Федеральное агентство по образованию
Байкальский государственный университет экономики и права

**В.В. Братищенко**

# БАЗЫ ДАННЫХ

Иркутск
Издательство БГУЭП
2006

УДК 681.3
ББК 32.973
Б 87

Печатается по решению редакционно-издательского совета
Байкальского государственного университета экономики и права

Рецензенты канд. физ.-мат. наук, доц. М.П. Климов
канд. пед. наук, доц. Л.В. Рожина,
зам. директора по информационным технологиям
Института математики, экономики информатики

Братищенко В.В.
Б 87 Базы данных: учеб. пособие. – Иркутск: Изд-во БГУЭП,
2006. – 98 с.

Пособие содержит сведения о базах данных, области их применения и технологиях использования. Описывается модель «Сущность-связь» в качестве основы для логического проектирования баз данных. Приведены сведения о иерархической, сетевой, реляционной моделях данных, применяемых для организации хранения информации в базах.

Предназначено для студентов специальности 351400 Прикладная информатика в экономике.

ББК 32.973

# ОГЛАВЛЕНИЕ

# 1. Базы данных. Основные определения

Базы данных на сегодняшний день являются наиболее распространенным способом хранения информации. Как форма хранения они обеспечивают быстрое проектирование информационной системы, достаточный уровень независимости программ и данных, надежность хранения, безопасность и защиту от несанкционированного доступа, разделение прав доступа для пользователей базы данных, эффективное выполнение запросов.

Уже первые применения вычислительной техники для решения задач управления привели к необходимости хранения на машинных носителях (в электронной форме) больших объемов информации. Жесткая привязка программ и структур данных приводила к необходимости переписывать программы при любых изменениях в форматах данных. Поэтому следующее поколение систем разрабатывалось по принципу «независимости программ от данных». Для этого необходимо хранить кроме данных еще и описание структур хранения – так называемые ***метаданные*** (данные о данных). Специальный компонент программного обеспечения использует метаданные для записи и извлечения данных из хранилища.

Распространение автоматизированных технологий управления привело к использованию разными подразделениями одной и той же информации. Хранение нескольких электронных версий данных вызвало помимо дублирования еще и появление и накопление противоречий в разных версиях. Решением этой проблемы стало совместное использование разными пользователями одной и той же коллекции данных – базы данных.

Таким образом, обеспечение независимости программ и данных с одной стороны и совместного использования данных с другой и привели к возникновению понятия ***базы данных (БД)*** – специальным образом организованной системы хранения данных и к созданию ***системы управления базами данных*** (**СУБД**) – программы, обеспечивающей доступ к базе для выполнения запросов на корректировку и извлечение данных. Широкое применение баз данных привело к появлению технологии Information engineering [1] построения информационных систем, в которой центральным ядром системы становятся данные (база данных). В таких системах пользователи получают доступ к базе данных в соответствие со служебными полномочиями при помощи приложений и посредничестве СУБД.

Известно несколько определений базы данных:

***База данных*** – поименованная, целостная, единая система данных, организованная по определенным правилам, которые предусматривают общие принципы описания, хранения и обработки данных [2].

***База данных*** есть совокупность данных, организованных в соответствии с некоторой концептуальной моделью данных, которая описывает характеристики этих данных и взаимоотношения между соответствующими им реалиями, и которая предназначена для информационного обеспечения одного или нескольких приложений [3].

При всей разнице общим в этих определениях является концептуальное единство. Это реализовано в виде использования однотипных структур хранения (например, все данные хранятся в таблицах) и применения соответствующих команд изменения и извлечения данных. Причем извлекаются данные в виде все тех же базовых структур хранения данных.

***Система управления базами данных (СУБД)*** – это программа (комплекс программ), которая обеспечивает хранение данных и доступ к базам данных. СУБД выполняет следующие функции:

- создание информационных структур, описание и поддержание целостности,
- выполнение запросов,
- поддержка многопользовательского режима,
- обеспечение надежности и безопасности хранения,
- управление полномочиями пользователей,
- выполнение сервисных функций (экспорт-импорт, резервное копирование и восстановление).

СУБД реализует общие принципы описания данных в виде ***ЯОД – языка описания данных (DDL – Data Definition Language)*** и принципы обработки данных в виде ***ЯМД – языка манипулирования данными (DML – Data Manipulation Language)***.

Для достижения независимости данных и программ выделяют три вида представления данных [4] (см. рис. 1):

- представление пользователя, или внешняя модель, или подсхема,
- логическое описание, или схема, или логическая БД,
- внутреннее описание или физическая БД.

Трехуровневая схема позволяет достичь высокой степени независимости данных и программ. При изменении либо представления пользователя, либо всей схемы БД достаточно изменить отображение схемы в соответствующую подсхему. Независимость схемы от физического представления позволяет безболезненно заменять одну технологию физического размещения данных другой. Все перечисленные возможности должны быть реализованы при помощи СУБД.

В соответствии с концепцией Information Engineering именно данные, а не функции являются наиболее стабильной и системообразующей составляющей. Следовательно, наиболее значимым становится проектирование

системы хранения данных. В проектировании информационного обеспечения традиционно выделяют два этапа проектирования:

1. Инфологическое проектирование – решение вопросов, связанных со смысловым содержанием данных:
   - о каких объектах и явлениях следует накапливать информацию;
   - какие их характеристики и взаимосвязи будут учитываться.
2. Даталогическое проектирование – решение вопросов связанных с представлением данных:
   - типы и форматы данных;
   - методы преобразования и смысловой интерпретации данных.

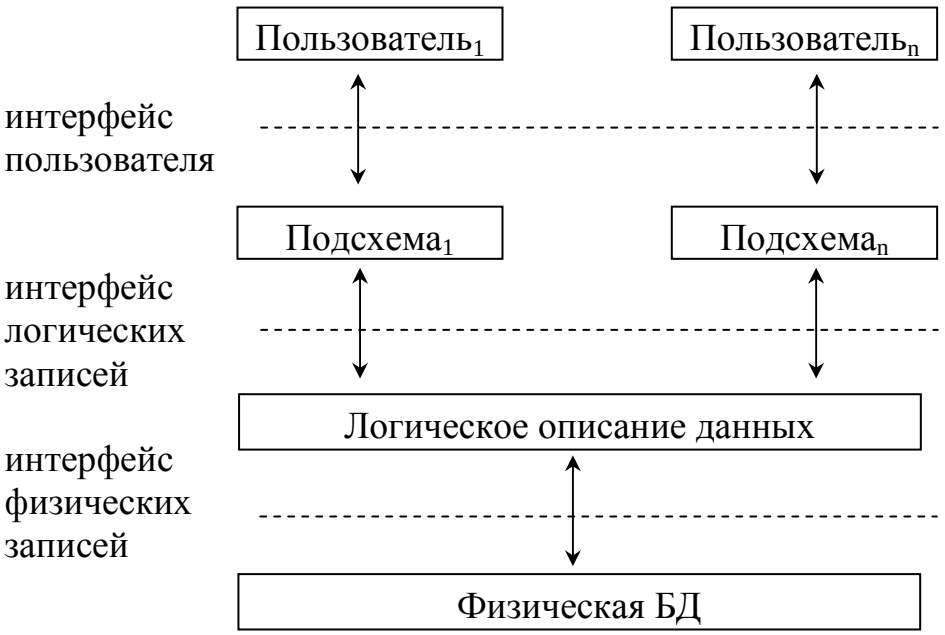



Рис. 1. Представления данных в базе

**Вопросы**

1. Приведите причины возникновения баз данных.
2. Что такое метаданные?
3. Дайте определение базы данных и системы управления базами данных.
4. Перечислите основные функции СУБД.
5. Перечислите и укажите назначение представлений данных в БД.
6. В чем заключается концепция Information Engineering?
7. Определите содержание инфологического и даталогического проектирований.

# 2. Инфологическое проектирование

Наиболее распространенной основой инфологического проектирования является модель «Сущность-связь». В отличии от структурной информационной модели, основанной на разбиении всех данных на составляющие вида структура (набор данных разных типов) или массив (набор данных одного типа), модель «Сущность связь» ориентирует проектировщика на выявление сущностей (наборов однотипных объектов или явлений, о которых необходимо сохранять информацию) и устойчивых связей между ними, необходимых для обеспечения целостности. Например, выделение связи договора с поставщиком означает, что в договоре указан поставщик, и сведения о нем должны быть зарегистрированы в БД.

## 2.1. Модель «Сущность-связь»

Инфологическое проектирование начинается с определения ***предметной области*** – части реально существующего мира, состояние которой БД должна адекватно описывать. Как уже упоминалось, ***модель «Сущность-связь»*** описывает предметную область в виде множества сущностей и связей между ними. Это описание отображается в виде диаграмм «сущность-связь» (Entity-Relationship Diagram – ERD). Для диаграмм «сущность-связь» применяется несколько систем обозначений. Мы будем использовать нотацию Мартина.

***Сущность*** – некоторая часть реального мира, которую можно выделить как самостоятельное целое, взаимодействующее с другими частями. Сущность в модели является обобщенным представителем множества объектов или явлений – экземпляров сущности. Сущности обычно именуются существительными. Например, для регистрации торговых операции можно выделить следующие сущности: продавцы, покупатели, товары. При этом сущность «Продавец» представляет в модели общие свойства всех продавцов. Деление на сущности достаточно произвольно. Например, можно объединить продавцов и покупателей в одну сущность – организации. Все объекты реального мира, соответствующие некоторой сущности, в информационной системе будут иметь одинаковый набор описателей – ***атрибутов (полей, реквизитов)***. Например, каждый продавец описывается наименованием, адресом, счетом в банке и т.д. На диаграмме сущность изображается прямоугольником (см. рис. 2).

Сущности можно классифицировать следующим образом:

- роли (люди, организации),
- реальные предметы,
- события (договора, поставки, оплаты),
- расположения.

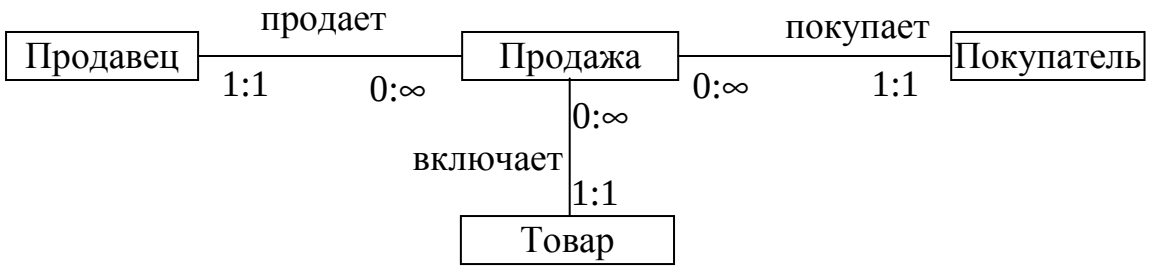



Рис. 2. Модель «Сущность-связь» для торговых операций

***Связь*** – это некоторая характеристика взаимодействия или ассоциации сущностей. Связь обычно именуется глаголом. Так же как сущность связь является обобщенным представителем экземпляров связей. На диаграмме связь изображается линиями, который соединяют связываемые сущности (см. рис. 2).

Наиболее распространенным видом связи является бинарный – когда сущности связаны попарно. Например, «Банк» – «Клиент». Если выделена связь нескольких сущностей, то, скорее всего, в виде связи выделено некоторое явление. Например, «продажу» можно рассматривать как связь «продавца», «товара» и «покупателя». Такие связи имеют набор атрибутов (цена, количество), и должны быть преобразованы в сущности (как это сделано на рис. 2) с бинарными связями с другими сущностями.

Связь обычно характеризует ссылочную целостность и не имеет атрибутов. Вместо атрибутов связь характеризуют количеством экземпляров связываемых сущностей. Например, у продажи один и только один продавец, а у продавца может быть несколько продаж (ноль или много). Различают связи «один к одному», «один ко многим» и «многие ко многим». Точнее, множество экземпляров каждой из связываемых сущностей описывают двумя числами: ординальное число указывает минимальное количество экземпляров сущности в связи, кардинальное – максимальное. Для связи «продажа включает товар» на рис. 2 установлено, что продажа включает один и только один товар, а товар включается в ноль или несколько продаж.

Связи многие ко многим, так же как связи более двух сущностей, требуют описания набором атрибутов, поэтому такие связи также преобразуют в некоторую сущность. Связь преподавателей с учебными группами можно описать в виде сущности «занятие», которое попарно связано с преподавателем и группой.

Для сущности можно установить ***ключ*** – множество атрибутов, которое позволяет однозначно идентифицировать один экземпляр сущности. Ключей может быть несколько. Например, работника можно идентифициро-

вать по серии и номеру паспорта, или по табельному номеру, или по индивидуальному номеру налогоплательщика. Один ключ из возможных выбирают для использования в качестве ссылки и называют первичным ключом, т.е. когда надо сослаться на экземпляр сущности используют соответствующее значение первичного ключа.

Рассматривая связь, можно выделить независимую (родительскую) сущность – существующую независимо от другой и зависимую – дочернюю, не существующую без родительской сущности. На рис. 2 независимыми сущностями являются покупатели, продавцы и товары, а зависимой является продажа. Дочерняя сущность содержит ссылку на соответствующую родительскую сущность. Обычно эта ссылка является набором атрибутов первичного ключа родительской сущности. Ссылочная целостность заключается в правильности значений всех ссылок. В частности не допускаются неопределенные («висячие») ссылки.

Сущности могут быть связаны попарно несколькими связями. Например, сущность «Валюта» может быть связана с сущностью «Обмен» двумя связями: «продается» и «покупается». Это означает неоднократно употребление ссылок с разными смыслами. Экземпляр обмена должен ссылаться и на продаваемую валюту, и на покупаемую валюту.

Возможен случай рекурсивной связи. Например, сущность «Сотрудник» может быть связана сама с собой связью «подчиняется» (описание каждого сотрудника содержит ссылку на другого сотрудника - непосредственного начальника). Рекурсия может быть непосредственной – сущность связывается сама с собой, как в рассмотренном примере, или опосредованной, если рекурсивная цепочка связей замыкается через другие сущности. Например, сущность «Сотрудник» связана с сущностью «Подразделение» связями «является руководителем» и «работает в», которые образуют рекурсивную связь (описание сотрудника содержит ссылку на подразделение, описание подразделения ссылку на сотрудника – начальника подразделения).

Связи могут отражать классификацию сущностей. Для этого вводят сущность – ***супертип***, которая объединяет характерные черты нескольких сущностей – ***подтипов*** (см. рис. 3). Например, роль супертипа может играть сущность «организация» по отношению к подтипам «продавец» и «покупатель». Соединение супертипа с подтипами на диаграмме образует категориальную связь, которая отображает-

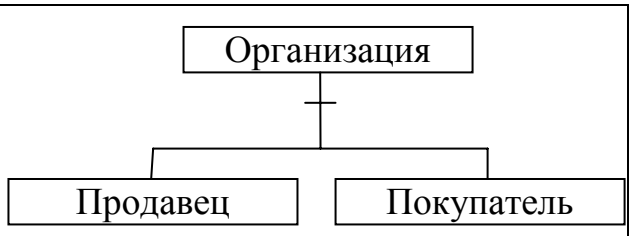


Рис. 3. Пример супертипа и подтипов

ся на диаграмме линиями, при этом для выделения супертипа используется черточка.

Рекомендуется следующая последовательность построения модели «Сущность – связь»:

1. Идентификация сущностей. Для идентификации полезными оказываются следующие приемы:
   - При интервьюировании пользователей выделяйте ключевые слова.
   - Просите пользователей определить объекты и явления, о которых необходимо собирать, хранить и извлекать информацию.
   - Изучайте формы документов.
   - Исследуйте уже созданные и применяемые ИС.

   Выясните количество экземпляров сущностей (десятки, сотни, ...). Придумайте простые звучные содержательные имена, имеющие точный смысл. Старайтесь избегать аббревиатур. Имена должны быть уникальны.
2. Идентификация атрибутов. Для этого выделяют свойства, характеристики и другие описатели сущностей. При этом важно, чтобы названия максимально точно отражали понятия (например, не просто «Дата», а «Дата Поставки»).
3. Описание каждой сущности набором атрибутов. При этом следует избегать включения в описание одной сущности атрибутов другой сущности, например в описание поставщика не следует включать номер договора. Также, не рекомендуется использовать атрибуты, являющиеся списками, например, атрибут «иностранные языки, которыми владеет работник».
4. Выбор первичных ключей. Значения первичного ключа используют для ссылки. Поэтому стараются выбирать первичные ключи минимальной длины для экономии памяти и увеличения скорости поиска по ключу. Не рекомендуется выбирать в качестве первичных ключей описатели, которые могут изменяться. Например, если выбрать серию и номер паспорта в качестве первичного ключа работника, то при замене паспорта придется менять в базе все ссылки на данного работника. Обычно, в качестве первичного ключа выбирают внутренний системный номер экземпляра сущности – код, который формируется автоматически при добавлении данных.
5. Построение диаграммы. В сложных случаях формируют несколько диаграмм, каждая из которых описывает определенную область учета. Например, в БД «Контингент студентов», одна диаграмма описывает деление контингента на факультеты, специальности, группы, другая представляет структуры данных для учета успеваемости, и т.д.

Важной частью работы по формированию модели данных всей информационной системы, объединяющей несколько подсистем, является согласование подмоделей данных (подсхем) с моделью данных организации в целом (см. рис. 4). Для этого используются следующие операции:

1) Установление идентичности. Два элемента идентичны, если они имеют одинаковое смысловое значение.
2) Агрегация – объединение связанных элементов модели в один сложный (адрес, банковские реквизиты).
3) Обобщение – разбиение множества сущностей на классы эквивалентности и введение супертипов – сущностей, соответствующих понятию класса. Таким образом, создается иерархия сущностей.

Проблема заключается в использовании разными пользователями разных терминов и описаний для одних и тех же понятий.

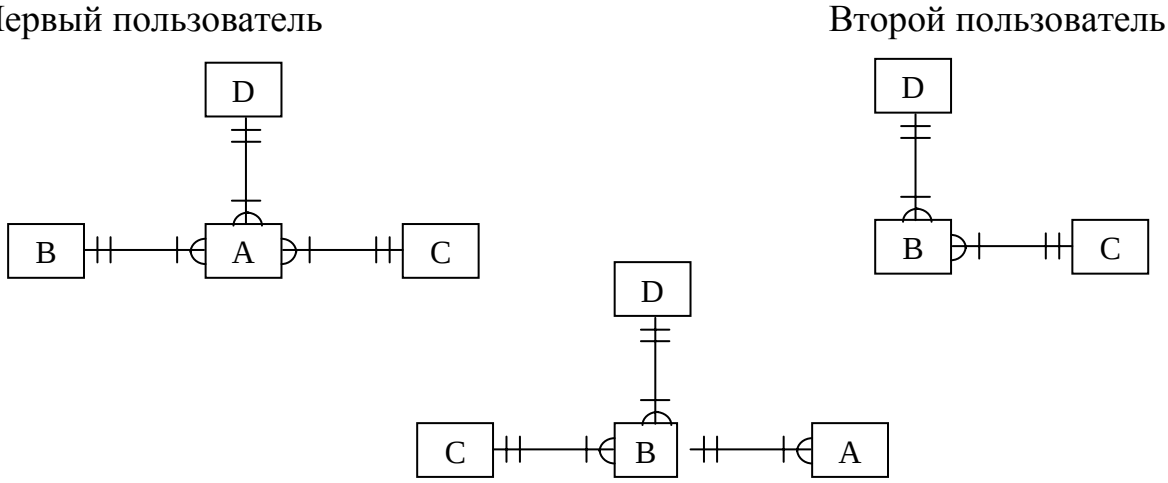



Рис. 4. Объединение представлений пользователей

## 2.2. Бизнес-правила

Технология использования БД включает процедуры изменения и использования данных при выполнении бизнес-функций. Для каждой функции указывается должностное лицо ответственное за выполнение функции, основания ее выполнения (документы, распоряжения, наблюдения, события), регламент выполнения функции, условия выполнения функции, особенности выполнения функции, правила регистрации данных, вытекающие из правил соответствующего учета. Все это объединяют в понятие «***бизнес-правила***». Например, в бухгалтерском учете для выполнения регистрации хозяйственной операции необходим соответствующим образом оформленный первичный документ, для которого указываются правила проводки и должностные лица, имеющие полномочия выполнять регистрацию.

Технологию работы с БД можно описывать в виде потока событий, приводящих к изменениям в БД и выборкам из БД. Анализ событий за-

ключается в выделении реальных экономических событий, которые могут происходить в предметной области, операций с данными, которые вызывает каждое событие, и обстоятельств, влияющих на выполнение операций. На товарно-сырьевой бирже такими событиями могут быть регистрация нового участника торгов, заявки на продажу или покупку, заключение сделки, оплата товара. Для каждого события перечисляются сущности и операции (выборка, добавление, изменение, удаление данных), которые события вызывают. В обстоятельства выполнения операций включают должностное лицо, ответственное за внесение данных и правила выполнения операций.

Рекомендуется следующая последовательность для проведения анализа событий:

**1-й шаг.** Идентификация событий для независимых сущностей. Результатом идентификации является таблица, состоящая из следующих колонок:

- имя события,
- краткое описание события,
- сущность, затрагиваемая событием,
- тип действия: добавление, изменение, удаление или выборка данных,
- условия выполнения действия.

Одно событие может влиять на много сущностей. Некоторые события могут обуславливаться более чем одни условием. Идентификация событий требует привлечения пользователей.

Условия можно разбить на два класса:

- вытекающие из экономической практики (известны пользователям),
- обусловленные связями структур данных (определяются системными аналитиками): оцениваются связи двух сущностей – выделяется независимая, родительская сущность и зависимая, дочерняя сущность. Для корректного добавления экземпляра дочерней сущности должен существовать экземпляр родительской. Удаление экземпляра родительской сущности возможно, если нет связанных с ним экземпляров дочерней.

Для событий, вызывающих изменения, должны указываться атрибуты, которые могут изменяться.

Затем выполняется идентификация событий для зависимых сущностей.

**2-й шаг.** Консолидация общих событий. Проверяется таблица – выделяются одинаково названные разные сущности (события) и по разному названные одинаковые. После этого выполняется реорганизация таблицы – для каждого события указываются сущности и действия.

По результатам построения модели данных и анализа событий пересматривается описание потоков данных. При этом возможно:

– добавление входных данных для проверки условий (например, на существование родителя);
– добавление атрибутов;
– добавление выходных данных;
– добавление новых операций (например, проверок).

Описание событий (технологии работы с БД) служит нескольким целям. Во-первых, можно проверить жизненный цикл каждой сущности: данные как минимум должны добавляться и использоваться, возможны также операции корректировки и удаления. Во-вторых, привязка процедур по рабочим местам позволяет определить полномочия на уровне базы данных каждого должностного лица, а также список приложений и функциональные возможности каждого приложений. В-третьих, создается описание бизнес-правил, которые становятся частью базы данных.

**Вопросы**

1. Дайте определение сущности и связи.
2. Укажите чем характеризуются сущности и связи.
3. Перечислите виды связей. Какие связи обычно отображают модели «Сущность-связь» и почему?
4. Определите понятия «ключ» и «первичный ключ».
5. Как выделяются «родительские» и «дочерние» сущности?
6. В чем заключается ссылочная целостность данных?
7. Опишите последовательность построения модели «Сущность-связь».
8. Что входит в технологию использования БД?
9. Приведите 2-3 примера бизнес-правил, которые необходимо выполнять для поддержания целостности БД.
10. Опишите технологию анализа событий.

## 3. Даталогическое проектирование. Модели данных

Содержанием даталогического проектирования является определение модели данных. ***Модель данных*** – это набор соглашений по способам представления сущностей, связей, агрегатов, систем классификации. Кроме этого каждая модель данных определяет особенности выполнения основных операций над данными: добавления, удаления, модификации, выборки.

Особое внимание при построении модели уделяют целостности и отсутствию избыточности данных. ***Избыточность*** – это многократное повторение одних и тех же сведений. Например, рассмотрим БД, в которой регистрируются торговые операции (см. рис. 2). Если в БД имеется несколько описаний одного и того же продавца, то все экземпляры этих описаний, кроме одного будут избыточными. Следует отличать описание от ссылки. Может быть несколько ссылок на один и тот же объект. Например, при регистрации нескольких продаж одного продавца в БД будет множество упоминаний (ссылок) данного продавца. Очевидно, что эти ссылки не являются избыточными. Кроме этого, несколько разных описаний одного и того же свойства могут отражать изменение свойства во времени, например смену фамилии или адреса. Приведенные примеры показывают, что проблема избыточности не является такой тривиальной, как это кажется на первый взгляд, и требует специальных методов решения. Например, если база данных допускает существование только одного адреса поставщика, то при регистрации нового адреса теряются сведения о старых адресах, а описание договора, в котором должен быть указан старый адрес, просто является не вполне корректным по причине указания нового адреса.

***Целостность*** – это полнота описания предметной области и отсутствие противоречий и неточностей данных. Полнота описания в компьютерных системах всегда имеет определенные границы. Полное описание человека является очевидно огромным, если не бесконечным. На самом деле такое описание не является необходимым, например, для учета кадров. Однако, описание сотрудника должно быть полным для организации учета кадров и управления персоналом. Таким образом, полнота понимается как достаточность сведений для решения определенного круга задач.

Целостность зависит от избыточности данных. Например, если в БД имеется несколько описаний одного продавца, то при изменении одного описания нарушается целостность данных. Однако, даже при отсутствии избыточности данных может возникнуть нарушение целостности. Пусть из БД удаляются сведения о некотором продавце. Теперь ссылки на этого продавца в зарегистрированных до удаления продажах будут неправильными и также квалифицируются как нарушение целостности. Кроме нару-

шения ссылочной целостности возможны нарушения бизнес-правил, например, нарушения баланса в бухгалтерском учете.

При проектировании БД учитывают возможные изменения структуры БД, скорость выполнения запросов к БД, а также необходимость разграничение доступа для разных групп пользователей.

Для построения БД могут использоваться три модели: иерархическая, сетевая, реляционная. Все эти модели отличаются структурами хранения данных и связей, а также особенностями выполнения выборок и корректировок.

### 3.1. Иерархическая модель

В иерархической модели данные представлены в виде множества деревьев (прадеревьев). ***Дерево*** состоит из вершины, связанной с произвольным количеством деревьев (поддеревьев). Вершину поддерева некоторой вершины называются ее ***потомком***, а вершину дерева – ***предком*** по отношению к своим потомкам. Вершина дерева, которое не является поддеревом ни для одной другой вершины, называется корнем дерева. Вершины, не имеющие поддеревьев, называются ***терминальными*** или ***листьями***. Количество вершин в пути максимальной длины в дереве называют ***высотой*** дерева.

Для определения связи в иерархической модели в дерево родительской сущности включают все описания связанных с ней сущностей. Для примера с рис. 2 в описание продавца, кроме его атрибутов должны быть вставлены поддеревья, описывающие его продажи (см. рис. 5).

При помощи деревьев можно описывать любые иерархические структуры. Проблемы начинаются, если в модели данных появляются зависимости «многие ко многим», как между продавцами и покупателями. Действительно в каждое дерево продаж необходимо включить поддерево описания покупателя. При этом описание покупателя будет дублироваться по числу его покупок.

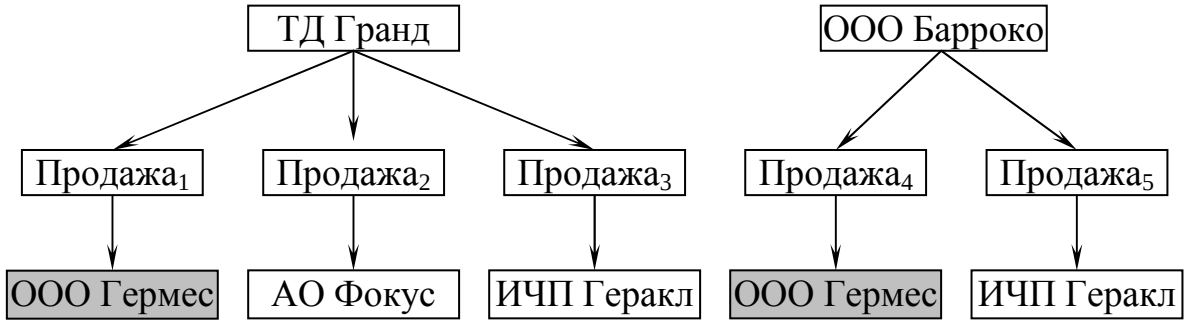



Рис. 5. Представление данных о продажах в иерархической модели

Для ликвидации избыточности, связанной с представлением связей многие ко многим вводят данные типа указатель для создания виртуальных поддеревьев в составе дерева. В примере на рис. 5 покупателей можно описывать виртуальными деревьями – ссылками на уникальные деревья, хранящие сведения каждое об одном покупателе.

Поиск и обработка данных в иерархической модели выполняются в виде процедур просмотра деревьев, а в дереве в виде процедур обхода дерева. Эти процедуры зависят от положения сущности в иерархии. Для обработки данных по продавцу достаточно найти и обработать описывающее его дерево. Для обработки данных по покупателю (расположенному на нижнем уровне иерархии) придется просматривать все деревья.

### 3.2. Сетевая модель

Основным элементом сетевой модели является набор, который описывает двухуровневую структуру (см. рис. 6), состоящую из одного владельца и нескольких связанных с ним компонентов. Компоненты могут быть связаны одновременно с владельцами разного типа. На рис. 6 продажи имеют в качестве владельца одного продавца и одного покупателя. Эта конструкция позволяет реализовать связи «многие ко многим» без присущей иерархической модели избыточности. Компонент некоторого набора может сам быть владельцем других компонентов и это позволяет описывать иерархические структуры. Например, владелец «предприятие» связан с компонентами – «цехами», которые являются владельцами компонент типа «бригада», а «бригада» владеет компонентами – «рабочими».

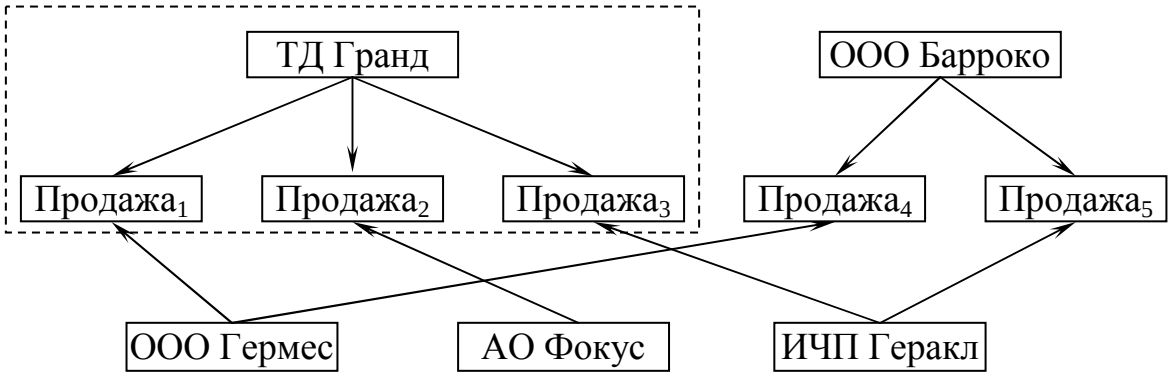



Рис. 6. Представление данных о продажах в сетевой модели. Пунктиром выделен набор данных

Операции поиска и обработки определяются в терминах просмотра наборов. Поиск должен начинаться с владельцев, расположенных на самом верхнем уровне иерархии, и продолжается среди компонент. Жесткая стратегия поиска является основным недостатком сетевой модели.

### 3.3. Реляционная модель

Основы реляционной модели данных были сформулированы Э.Ф.Коддом в 1970 г. Структурой хранения данных в этой модели является отношение – таблица со следующими свойствами:

1. Каждый столбец содержит информацию одного типа.
2. Ячейки – поля – таблицы не содержат агрегатов (структур или массивов) данных.
3. Таблицы не содержат одинаковых строк.
4. Порядок строк и столбцов не имеет значения. Все операции используют содержательную сторону данных, а не их расположение внутри таблицы.

Для описания связей вводятся первичные ключи – минимальные наборы полей, позволяющие указывать ровно одну строку (кортеж) таблицы. Значение ключа используется для ссылки в других таблицах, что и является описанием связей данных. Поскольку первичный ключ играет ведущую роль в описании связей и поиске данных, размер ключа стараются сделать минимальным для оптимизации поиска. Это приводит к использованию номеров или кодов в качестве первичных ключей. На рис. 7 приведено представление данных о продажах в реляционной модели. Стрелки на рисунке отражают ссылки данных одной таблицы на данные других таблиц. В главе 4 приведено формальное описание реляционной модели данных.

Для запросов к реляционным данным придумано несколько языков. Все они используют похожие операции в разных нотациях. Результатом каждой операции является отношение. Кроме операций над таблицами как над множествами записей, вводятся операция соединения, позволяющая из нескольких таблиц получить таблицу-результат, операция проекции для выделения таблицы с нужным набором полей (колонок), операция фильтрации для выбора записей (строк), удовлетворяющих некоторому условию и ряд других. Язык реляционной алгебры и исчисление отношений описаны в главе 4, а язык SQL в главе 6.

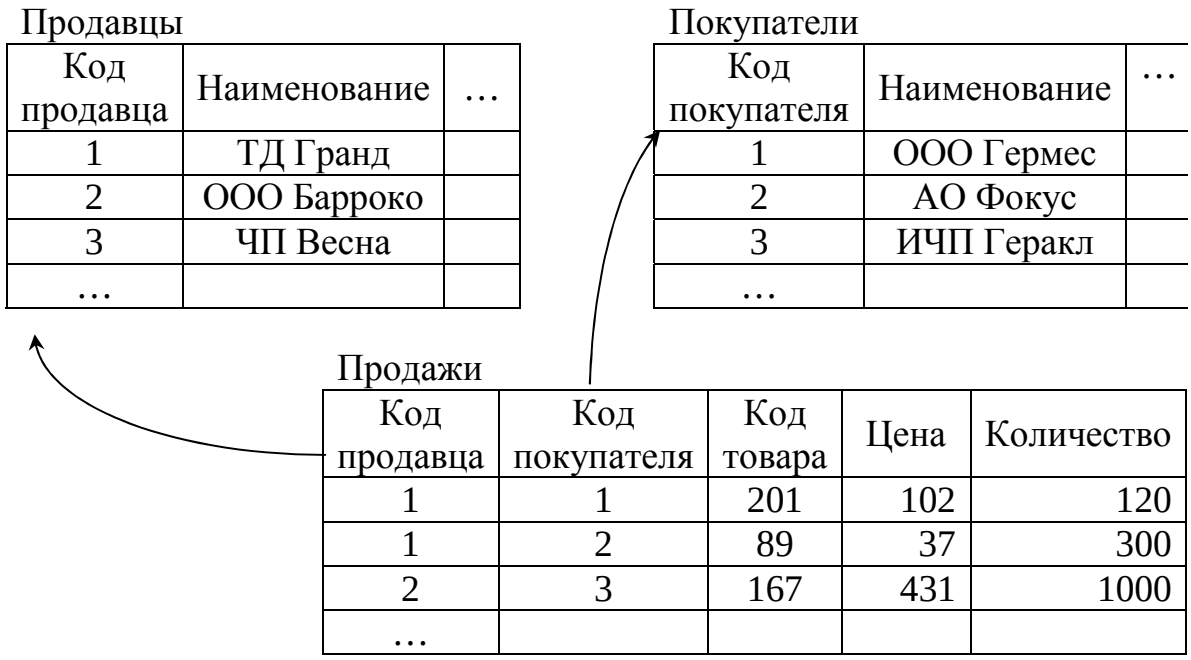



Рис. 7. Представление данных о продажах в реляционной модели данных. Стрелками показаны связи полей таблиц

**Вопросы**

1. Дайте определения целостности и избыточности данных.
2. Для каждой модели данных определите базовые структуры хранения данных и операции над ними, укажите достоинства и недостатки.
3. Перечислите свойства отношений.
4. Как поддерживается ссылочная целостность в каждой модели данных?

# 4. Реляционная модель данных

## 4.1. Определение реляционной модели

В реляционной модели данные представлены в виде набора отношений (relation). ***Отношение*** $r = \{(d_1,...,d_n),\ d_1 \in D_1,...,d_n \in D_n\}$ есть множество ***кортежей*** $(d_1,...,d_n)$ , значения которых принадлежат ***доменам*** $D_1,...,D_n$ . Таким образом, отношение есть подмножество декартова произведения доменов

$$r \subset D_1 \times ... \times D_n .$$

Число $n$ называют ***степенью отношения***. Количество кортежей в отношении $r$ называют ***мощностью отношения*** $r$.

Отношение является аналогом таблицы с приведенными в параграфе 3.3 свойствами. Колонки таблицы делят каждый кортеж (запись) на ***атрибуты*** – компоненты кортежа. Для указания атрибута используют уникальные в рамках одной таблицы имена. Термин «атрибут» используется в разных смыслах. Во-первых, атрибут определяет множество возможных значений соответствующей колонки таблицы, так как областью значений атрибута является соответствующий домен. Во-вторых, атрибут используется в программах для описания доступа к соответствующей компоненте кортежа – аналогичным понятием является ***поле*** записи. Атрибут $A_i$ – это функция, вычисляющая значение i-ой компоненты кортежа отношения $r$. Множеством значений атрибута $A_i$ является домен $D_i$. В алгоритмических языках атрибут обозначается чаще именем и реже номером атрибута в кортеже.

Схемой $r(R)$ отношения $r$ называют множество атрибутов $R = \{A_1,...,A_n\}$. Реляционная модель данных или схема БД – это множество схем отношений, дополненное различными ограничениями и правилами (бизнес-правилами).

Далее по тексту используются следующие обозначения

$t[A_i]$ – значение атрибута $A_i$ в кортеже $t \in r$ отношения $r$;

$t[X] = (t[A], t[B], ...)$ – $X$-значение кортежа $t$ на множестве атрибутов $X = \{A, B, ...\}$ ;

Важное значение в теории реляционных БД имеет ключ, позволяющий выделять ровно один кортеж из отношения. Ключом $K$ отношения $r(R)$ называют множество атрибутов $K=\{B_1,...,B_m\} \subset R$ такое, что в отношении $r$ не существует двух кортежей с одинаковым $K$-значением ключа и ни одно собственное подмножество ключа $K$ не обладает этим свойством.

## 4.2. Операции над отношениями. Реляционная алгебра

Над отношениями ***r*** и ***s*** с одинаковыми схемами можно выполнять теоретико-множественные операции

1. ***Объединение отношений*** – множество кортежей, принадлежащих отношению ***r*** или отношению ***s***

   $r$ **UNION** $s = \{ t : (t \in r) \cup (t \in s) \}$.

2. ***Пересечение отношений*** – множество кортежей, принадлежащих ***r*** и ***s*** одновременно

   $r$ **INTERSECT** $s = \{ t : (t \in r) \,\&\, (t \in s) \}$.

3. ***Разность отношений*** – множество кортежей принадлежащих ***r*** и не принадлежащих ***s***

   $r$ **MINUS** $s = \{ t : (t \in r) \,\&\, (t \notin s) \}$.

4. ***Дополнение отношения r*** можно определить как разность множества всех возможных записей отношения ***r*** и самого отношения ***r***

   **NOT** $r = (D_1 \times D_2 \times ... \times D_n)$ **MINUS** $r = \{ t : (t \in D_1 \times D_2 \times ... \times D_n) \,\&\, (t \notin r) \}$.

   За исключение тривиальных случаев мощность дополнения выражается огромным числом и выполнение этой операции физически невозможно. Для уменьшения количества получаемых кортежей вводят операцию «активное дополнение».

5. ***Активное дополнение***. Вместо декартова произведения доменов в этой операции используют декартово произведение множеств используемых значений $D'_i = \{ t[A_i] : t \in r \}$.

Кроме множественных операций вводят специфические операции для манипулирования отношениями.

6. ***Выбор*** или ***фильтрация*** – множество кортежей, обращающих логическую формулу $\theta(t)$ в истину

   $r$ **WHERE** $\theta(t) = \{ t : \theta(t) \}$

7. ***Проекция*** – из отношения вычеркиваются столбцы, не входящие в множество атрибутов ***X*** и повторяющиеся строки

   $r[X] = \{ t[X] : t \in r \}$

8. ***Естественное соединение***

   $$q(R \cup S) = r(R) \textbf{ JOIN } s(S)$$
   $$= \{ t : (\exists t_r \in r : (\exists t_s \in s : (t_r = t[R]) \cap (t_s = t[S])$$

получается как результат комбинирования каждого кортежа $t_r$ отношения ***r*** с каждым кортежем $t_s$ отношения ***s*** при условии совпадения одноименных атрибутов $t_r[R \cap S] = t_s[R \cap S]$ (см. рис.8).

Для различных прикладных аспектов важно разлагать при помощи проекции отношение на множества отношений, содержащих туже самую ин-

формацию. Отношение $q\ (R \cup S)$ разложимо без потерь на отношения $r\ (R) = q\ [R]$ и $s\ (S) = q\ [S]$ если $q = r$ **JOIN** $s$.

*r*

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |
| $a_3$ | $b_4$ |

*s*

| A | C |
|---|---|
| $a_1$ | $c_1$ |
| $a_1$ | $c_2$ |
| $a_3$ | $c_3$ |
| $a_3$ | $c_4$ |

*q*

| A | B | C |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_2$ | $c_2$ |
| $a_3$ | $b_4$ | $c_3$ |
| $a_3$ | $b_4$ | $c_4$ |

Рис. 8. Пример естественного соединения $q = r$ JOIN $s$.

9. ***θ-соединение***: пусть $\theta$ – логическая формула над соответствующими множествами атрибутов тогда

   $q\ (R \cup S) = r\ (R)$ **JOIN** $s\ (S)$ **ON** $\theta$ =

   $= \{ t : (\exists t_r \in r : (\exists t_s \in s: (t_r = t\ [R]) \cap (t_s = t\ [S]) \cap \theta (t_r [R], t_s [S]))) \}$

   $\theta$-соединение аналогично естественному соединению с той лишь разницей, что условие совпадения общих атрибутов кортежей заменяется выполнение логического условия $\theta(R,S)$.

10. Операция ***переименования***

    $r$ **RENAME** $A_1, A_2, \ldots$ **AS** $B_1, B_2, \ldots$

    просто изменяет наименование атрибута $A_1$ на $B_1$, $A_2$ на $B_2$ и т.д.

Реляционная алгебра – множество операций над отношениями, результатом которых является отношение. Доказано, что приведенные операции: выбора, естественного соединения, проекции, объединения, разности и переименования – образуют базис реляционной алгебры, то есть любую операцию над отношениями можно построить при помощи операций базиса и постоянных отношений.

### 4.3. Реляционное исчисление

Реляционное исчисление есть множество формул вида

$\{ t\ (R) : \phi(t) \}$,

где $t$ – переменная-кортеж со схемой $R$, $\phi(t)$ формула над атрибутами и кортежами отношений и константами, построенная при помощи скобок, операций сравнения, логических операций, кванторов общности и существования.

Для того, чтобы алгоритм вычисления отношения по формуле был конечным, множество формул ограничивают безопасными выражениями.

Например, формула $\{ t : t \notin r \}$ является опасной, так как приводит к построению множества бесконечной мощности.

Доказано, что реляционная алгебра и исчисление отношений эквивалентны по функциональной мощности: для любого выражения реляционной алгебры можно построить эквивалентную формулу исчисления отношений и наоборот.

Для реляционных БД разработаны специальные языки запросов. Среди них наиболее распространенным является Structured Query Language (SQL) – структурированный язык запросов.

**Вопросы**

1. Дайте определения отношения, кортежа, домена, порядка, мощности, атрибута, ключа отношения.
2. Перечислите и определите теоретико-множественные операции реляционной алгебры. Какие из этих операций трудно реализовать и почему?
3. Определите и приведите примеры операций фильтрации, проекции, естественного соединения, $\theta$-соединения, переименования.
4. Что такое реляционная алгебра и реляционное исчисление.
5. Приведите пример запроса к данным из нескольких таблиц и запишите его в реляционной алгебре и в реляционном исчислении.

# 5. Проектирование реляционной БД

## 5.1. Требования к схеме БД

Проектирование реляционной БД заключается в построении схемы БД и определении всех ограничений, которым должны удовлетворять данные. Кроме этого в проект БД включают бизнес-правила.

Схема БД должна удовлетворять следующим требованиям:

- БД должна быть целостной – набор отношений должен правильно описывать предметную область.
- БД не должна быть избыточной – одни и те же сведения не должна описываться дважды.
- Количество отношений должно быть минимальным.
- Первичные ключи должны быть минимальными.
- Изменения БД должны быть минимальными при изменении предметной области.
- Схема БД должна обеспечивать минимальное время выполнения запросов.

Стремление сохранить целостность и уменьшить избыточность привели к созданию нормальных форм отношений. Нормализация – приведение отношений к нормальным формам – является необходимым этапом проектирования БД.

Избыточность вызывает перечисленные ниже проблемы при выполнении корректировок данных в базе (для примера рассмотрим одно отношение, содержащее все данные для схемы с рис. 2):

- Добавление новых данных (например, нового поставщика) может приводить к неопределенным значениям некоторых полей (о поставке, товаре и получателе).
- Удаление одних данных (для исключения данных о невыполненной поставке) может приводить к потере других, размещенных в удаляемом кортеже (например о поставщике).
- Обновление кортежа (адреса поставщика) приводит к несоответствию с данными из других кортежей.

Каждая нормальная форма позволяет устранить некоторые из этих проблем.

## 5.2. Функциональные зависимости. Ключи

Причина перечисленных выше проблем корректировки данных заключается в наличии зависимостей между атрибутами в отношении. Наиболее распространенным видом зависимости является функциональная зависимость.

Пусть дано отношение ***r*** со схемой ***R*** и два его подмножества ***X, Y*** $\subset$ ***R*** атрибутов. Множество атрибутов ***Y*** функционально зависит от множества атрибутов ***X*** (обозначается ***X*** $\rightarrow$ ***Y***), если в отношении ***r*** не может быть двух кортежей, в которых одному и тому же набору ***X*** - значений соответствуют разные наборы ***Y*** – значений, т.е. функциональная зависимость требует, чтобы в случае совпадения значений аргументов совпадали и значения функций**.** Данное понятие функциональной зависимости соответствует табличному определению функции ***Y*** аргумента ***X*.**

Функциональная зависимость определяется предметной областью и не зависит от экземпляра отношения (должна выполняться для всех экземпляров отношений). Связь многие к одному в модели «Сущность-связь» определяет функциональную зависимость. Например, связь «поставщика» с «продажами» означает, что «продажа» является аргументом, а «поставщик» - функцией: по описанию конкретной продажи можно однозначно определить ровно одного «продавца» (но не наоборот).

Определение в качестве первичных ключей системных уникальных кодов также означает введение функциональных зависимостей – все атрибуты записи функционально зависят от такого ключа.

Одни и те же множества функциональных зависимостей могут быть заданы несколькими способами. Известно много правил и теорем вывода новых функциональных зависимостей. Ниже приведены некоторые из них.

Аксиомы Армстронга

- Аксиома рефлексивности $Y \subseteq X \Rightarrow X \rightarrow Y$ (тривиальные функциональные зависимости).
- Аксиома пополнения $X \rightarrow Y \Rightarrow X \cup Z \rightarrow Y \cup Z$.
- Аксиома транзитивности $X \rightarrow Y,\ Y \rightarrow Z \Rightarrow X \rightarrow Z$.

Теоремы-правила

- Правило объединения $X \rightarrow Y,\ X \rightarrow Z \Rightarrow X \rightarrow Y \cup Z$.
- Правило псевдотранзитивности $X \rightarrow Y,\ W \cup Y \rightarrow Z \Rightarrow W \cup X \rightarrow Z$.
- Правило декомпозиции $X \rightarrow Y,\ Z \subset Y \Rightarrow X \rightarrow Z$.

Все функциональные зависимости $F^+$, выводимые из множества ***F*** называют замыканием ***F*** или полным семейством зависимостей. Два множества функциональных зависимостей ***F*** и ***G*** эквивалентны, если $F^+ = G^+$. Для целей проектирования необходимо получить минимальное множество всех функциональных зависимостей предметной области. Таким множеством является неизбыточное множество функциональных зависимостей. Множество функциональных зависимостей неизбыточно, если оно не является эквивалентным любому своему собственному подмножеству.

Понятие функциональной зависимости позволяет определить ключ следующим образом. Множество атрибутов ***K*** $\subset$ ***R*** отношения ***r ( R )*** с множе-

ством функциональных зависимостей $F^+$ называют ключом отношения $r$, если

- $R$ функционально зависит от $K$ ($K \to R \in F^+$);
- $R$ не зависит функционально ни от какого собственного подмножества $K$.

Один из ключей выбирают для ссылок на кортежи отношения и называют первичным ключом. Все остальные ключи называют возможными ключами. ***Первичным атрибутом*** называют атрибут, входящий в состав какого-либо ключа и ***непервичным*** называют атрибут, не входящий в состав ни одного ключа.

## 5.3. Декомпозиция и соединение

Первым требованием к БД является сохранение целостности – всех фактов и зависимостей, необходимых для адекватного отображения предметной области. Для этого могут быть предложены различные схемы БД – различные наборы отношений. В процессе реструктуризации БД важно выделить операции, которые позволяют преобразовывать схему БД без потери целостности. Такими операциями являются

- декомпозиция – замена одного отношения несколькими,
- соединение – соединение нескольких отношений в одно.

Декомпозиция $\rho = \{r_1(R_1),..., r_n(R_n)\}$ отношения $r\ (R)$ должна содержать ту же информацию, что и отношение $r$. Для этого декомпозиция должна удовлетворять следующим условиям

1) $R_1$ **UNION ... UNION** $R_n = R$.
2) $r_1,..., r_n$ не должны содержать информацию, которой нет в $r$: $r_i = r[R_i]$.
3) $r_1,..., r_n$ должны содержать информацию, которая есть в $r$:
   $r = r_1$ **JOIN** $r_2$ **JOIN ... JOIN** $r_n$,
   Свойства 2 и 3 означают, что $\rho$ обладает свойством ***соединения без потерь***.
4) Декомпозиция должна сохранять множество функциональных зависимостей. Проекцией $\pi_Z(F)$ множества функциональных зависимостей $F$ на множество атрибутов $Z$ называется множество функциональных зависимостей $X \to Y$, где $X, Y \subset Z, (X \to Y) \in F$. Декомпозиция $\rho$ сохраняет множество функциональных зависимостей $F$, если $\pi_{R_1}(F) \cup \ldots \cup \pi_{R_n}(F)$ эквивалентно $F$.

Свойство соединения без потерь и сохранения множества функциональных зависимостей не следуют одно из другого.

## 5.4. Нормальные формы

Формализация процесса построения схемы реляционной БД, изучение аномалий, возникающих при корректировке БД, привели к формулировке ряда требований. Если отношение удовлетворяет требованиям, то говорят, что оно находится в соответствующей нормальной форме. Процесс приведения схем отношений к нормальной форме называют нормализацией.

***Первая нормальная форма (1НФ)*** является самой простой.

***Отношение находится в 1НФ, если каждый атрибут отношения является атомарным, т.е. неделимым в смысловом отношении.***

Примерами атрибутов, нарушающих 1НФ являются адреса, списки и вообще любые агрегаты данных.

***Вторая нормальная форма (2НФ)*** связана с исключением из отношения неполных (частичных) функциональных зависимостей непервичных атрибутов от ключа. ***Неполная*** или ***частичная*** функциональная зависимость в отношении ***r*** атрибута ***A*** от ключа ***K*** существует, если существует собственное подмножество ***Y*** ключа ***K*** и функциональная зависимость ***A*** от ***Y***. Если такого подмножества у нет, то зависимость ***A*** от ключа ***K*** называется ***функционально полной***.

***Отношение находится в 2НФ, если оно находится в 1НФ и каждый непервичный атрибут функционально полно зависит от ключа.***

В отношении с неполными функциональными зависимостями возникает избыточность, что приводит к проблемам сохранения целостности при добавлении записей и к возможно потере информации о неполной зависимости при удалении. Например, пусть итоги сессии хранятся в отношении «Оценки» с атрибутами «Номер студ.билета», «Фамилия студента», «Номер группы», «Предмет», «Преподаватель» и «Оценка». Пусть существуют следующие зависимости:

«Номер студ.билета» → «Фамилия студента»,

«Номер студ.билета» → «Номер группы»,

«Номер студ.билета», «Предмет» → «Оценка»,

«Номер группы», «Предмет» → «Преподаватель».

Ключом отношения является комбинация атрибутов «Номер студ.билета» и «Предмет». Зависимости «Номер студ.билета» → «Фамилия студента», «Номер студ.билета» → «Номер группы» являются неполными. Для приведения данного отношения в 2НФ необходимо вывести из него указанные зависимости, например выполнить декомпозицию на отношения
(«Номер студ.билета», «Фамилия студента», «Номер группы») и
(«Номер студ.билета», «Предмет», «Преподаватель», «Оценка»).

***Третья нормальная форма (3НФ)*** исключает не только неполные, но и транзитивные зависимости. ***Транзитивная зависимость*** $K \to A$ атрибута $A$ от ключа $K$ существует, если существует множество атрибутов $Y \subset R$ такое, что $Y \to A$.

***Отношение r( R ) находится в 3НФ, если оно находится в 2НФ и каждый непервичный атрибут A нетранзитивно зависит от ключа X.***

Транзитивные зависимости также приводят к избыточности и проблемам обновления. Поэтому транзитивные зависимости от ключа необходимо описывать отдельным отношением. Например, в отношении («Номер студ.билета», «Номер комнаты», «Телефон»), описывающем распределение студентов по комнатам общежития есть зависимости «Номер студ.билета»→«Номер комнаты», «Номер комнаты»→«Телефон». Атрибут «Телефон» транзитивно зависит от ключа «Номер студ.билета».

***Нормальная форма Бойса-Кодда (НФБК)*** применяется, если в отношении $r$ первичный атрибут транзитивно зависит от ключа. Определение НФБК использует понятие детерминанта. Множество атрибутов $X$ является ***детерминантом*** $B$, если $X \to B$ и ни одно собственное подмножество $X$ не обладает этим свойством.

***Отношение r ( R ) находится в НФБК, если оно находится в 2НФ и каждый детерминант из R является возможным ключом.***

Например, пусть значение кортежа $(a, f, o)$ заключается в том, что фирма $f$ покупает билеты в аэропорту $a$ и оплачивает билеты отделение фирмы $o$. Отношение содержит две зависимости $(a, f) \to o$, $o \to f$. Ключами отношения являются $\{a, o\}$ и $\{a, f\}$. Фирма $f$ является первичным атрибутом, но зависит от атрибута $o$, который является детерминантом, но не является ключом отношения. Зависимость $o \to f$ приводит к избыточности и должна быть выведена из отношения.

Нормальная форма Бойса-Кодда обеспечивает отсутствие аномалий корректировки, добавления и удаления данных, если все зависимости являются функциональными. Однако, кроме функциональных существуют зависимости других видов, в частности многозначная зависимость. Например, в отношении «Расписание» на рис. 9 нет функциональных зависимостей и оно находится в третьей нормальной форме, тем не менее его можно представить как результат соединения двух отношений «Дни вылетов» и «Самолеты рейсов».

Это возможно благодаря существованию многозначных зависимостей «Рейс»→→ «День недели» и «Рейс»→→ «Тип самолета». Объединение нескольких многозначных зависимостей приводит к появлению избыточности. Многозначная зависимость наступает тогда, когда значениями

функций являются множества. Более точным является следующее определение.

«Расписание»

| Рейс | День недели | Тип самолета |
|---|---|---|
| 106 | пн | 747 |
| 106 | чт | 747 |
| 106 | пн | 1011 |
| 106 | чт | 1011 |
| 204 | ср | 707 |
| 204 | ср | 727 |

«Дни вылетов»

| Рейс | День недели |
|---|---|
| 106 | пн |
| 106 | чт |
| 204 | ср |

«Самолеты рейсов»

| Рейс | Тип самолета |
|---|---|
| 106 | 747 |
| 106 | 1011 |
| 204 | 707 |
| 204 | 727 |

Рис. 9. Пример многозначной зависимости и её декомпозиции

В отношении ***r(A, B, C)*** **существует многозначная зависимость** ***A→→B,*** если множество значений ***{b∈B}*** соответствующих паре (***a, c), a∈A, c ∈C*** зависит только от ***a*** и не зависит от ***c***.

Из определения следует, что при наличии в отношении ***r*** кортежей $(a_1, b_1, c_1)$ и $(a_1, b_2, c_2)$ в нем (по причине многозначной зависимости) должны быть кортежи $(a_1, b_2, c_1)$ и $(a_1, b_1, c_2)$, в этом случае в отношении кроме зависимости $A \to\to B$ присутствует зависимость $A \to\to C$. Приведенное в примере отношение находится в ЗНФ (ключ $ABC$), но в тоже время разлагается без потерь на отношения со схемами $AB$ и $AC$.

**Отношение находится в четвертой нормальной форме (4НФ), если в случае существования многозначной зависимости *A→→B* все остальные атрибуты функционально зависят от *AB*.**

Существует еще и пятая нормальная форма, связанная с зависимостью соединения. Зависимость соединения обобщает многозначную зависимость на случай когда декомпозиция без потерь возможна не на два, а на большее число отношений (см. рис. 10)

В целом построение реляционной модели данных можно представить следующим образом.

1) Получение некоторого начального набора отношений на основе модели «Сущность-Связь».
2) Определение всех функциональных, многозначных и другого вида зависимостей.
3) Построение минимального покрытия – неизбыточного множества функциональных зависимостей.
4) Построение минимального покрытия – неизбыточного множества функциональных зависимостей.

5) На основе минимального покрытия выполняется нормализация отношений при помощи операций декомпозиции и соединения.
6) Полученная схема БД оценивается с точки зрения безопасности, секретности и эффективности использования и хранения данных. При необходимости предлагается иное деление БД на отношения и пункт 4) выполняется заново.

**r**

| A | B | C |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b2 | c2 |
| a2 | b3 | c3 |
| a3 | b3 | c4 |
| a4 | b4 | c5 |
| a5 | b5 | c5 |

**$r_1$**

| A | B |
|---|---|
| a1 | b1 |
| a1 | b2 |
| a2 | b3 |
| a3 | b3 |
| a4 | b4 |
| a5 | b5 |

**$r_2$**

| A | C |
|---|---|
| a1 | c1 |
| a1 | c2 |
| a2 | c3 |
| a3 | c4 |
| a4 | c5 |
| a5 | c5 |

**$r_3$**

| B | C |
|---|---|
| b1 | c1 |
| b2 | c2 |
| b3 | c3 |
| b3 | c4 |
| b4 | c5 |
| b5 | c5 |

**Рис. 10. Пример зависимости соединения**

## 5.5. Ограничения реляционных языков и некоторые ошибки проектирования

При проектировании БД необходимо учитывать специфику реляционных языков, в основе которых лежит реляционная алгебра. Первый пример [6] связан с отсутствием средств построения списков колонок с определенными свойствами. В таблице с рис. 11 каждому элементу соответствует одно поле. Запросы о составе вещества (элементы, составляющие не менее 1% вещества) должны возвращать список атрибутов записи, удовлетворяющих условию, а такая операция не предусмотрена реляционной алгеброй. Списками в реляционных БД являются строки, а не столбцы. Поэтому правильнее разбить это отношение на три различных отношения:

– ВЕЩЕСТВА(НОМ_ВЕЩЕСТВА, ВЕЩЕСТВО),
– ЭЛЕМЕНТЫ(НОМ_ЭЛЕМЕНТА, ЭЛЕМЕНТ),
– ХИМИЧЕСКИЙ_СОСТАВ_ВЕЩЕСТВ(НОМ_ВЕЩЕСТВА, НОМ_ЭЛЕМЕНТА, ПРОЦЕНТ).

Таблицы, аналогичные таблице с рис. 11, (их еще называют сводными или шахматными) довольно часто встречаются в качестве итоговых. Например, продавцы в качестве заголовков строк, потребители, в качестве заголовков колонок, а каждая клетка содержит стоимость приобретенных товаров. Такие данные в реляционной модели удобнее (с точки зрения построения запросов) хранить в отношении с атрибутами «продавец», «покупатель», «стоимость».

**Химический состав веществ**

| Наименование вещества | Водород | Гелий | … | 105 элемент |
|---|---|---|---|---|
| Дезоксирибонуклеиновая кислота | 5 | 3 | … | 0.01 |
| Бензин | 50 | 0 | … | 0 |
| … | … | … | … | … |

**Рис. 11. Описание состава веществ по элементам**

Вторая достаточно распространенная ошибка – это разбиение множества объектов на классы и каждый класс описывать отдельным отношением. Например, таблицу на рис. 11 можно разбить на 105 отношений:

– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ВОДОРОД(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ),
– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ГЕЛИЙ(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ),
– …
– ВЕЩЕСТВА_СОДЕРЖАЩИЕ_ЭЛЕМЕНТ105(НОМ_ВЕЩЕСТВА, ПРОЦЕНТ).

В этом случае для ряда запросов придется формировать список не атрибутов, а отношений. Такие возможности также не предусмотрены реляционной алгеброй и создают алгоритмические проблемы. Даже простое разбиение таблицы товаров на две – проданные и не проданные – требует неоправданного усложнения запросов в некоторых случаях.

**Вопросы**

1) Перечислите требования к схеме БД. Какие из этих требований являются более важными и почему?
2) Дайте определение понятий «функциональная зависимость», «ключ», «первичный атрибут», «непервичный атрибут», «неизбыточное множество функциональных зависимостей», «функционально полная зависимость», «функционально неполная зависимость», «частичня зависимость», «транзитивная зависимость», «детерминант», «многозначная зависимость».
3) Когда два множества функциональных зависимостей считаются эквивалентными?
4) Может ли атрибут быть первичным в одном отношении и непервичным в другом? Приведите примеры.
5) В чем смысл декомпозиции и соединения?
6) Как выполнить декомпозицию без потери информации?
7) Дайте определение каждой нормальной формы, укажите какие проблемы изменения данных, связаны с нарушением каждой из них.
8) Перечислите шаги построения реляционной модели.

# 6. Структурированный язык запросов SQL

Прообраз языка SQL возник в 1970 году в рамках научно-исследовательского проекта System/R, работа над которым велась в лаборатории Санта-Тереза фирмы IBM. Сегодня SQL – это фактический стандарт интерфейса с современными реляционными СУБД. Язык SQL имеет официальный стандарт – ANSI/ISO. Большинство разработчиков СУБД придерживается этого стандарта, однако часто расширяют его для реализации новых возможностей обработки данных. Новые механизмы управления данными могут быть использованы только через специальные операторы SQL, в общем случае не включенные в стандарт языка.

SQL не является языком программирования в традиционном представлении. На нем пишутся не программы, а запросы к базе данных. Поэтому SQL – декларативный язык. Это означает, что с его помощью можно сформулировать, что необходимо получить, но нельзя указать, как это следует сделать.

## 6.1. Типы данных

Язык SQL / 89 поддерживает следующие типы данных:

- BLOB – Binary large object произвольной величины для хранения графики, текстов;
- CHAR(n) – строки ровно из n знаков: строки длиной меньше n дополняются пробелами;
- VARCHAR (n) -строки не более n знаков (n может быть от 1 до 32767), хранится реально указанное число знаков;
- DATE – длинная дата (8 байт) для дат из интервала 1 января 100 – 11 января 5941;
- SMALLDATETIME – короткая дата (4 байта) от 1.01.1900 до 6.6.2079;
- DECIMAL (p, s), NUMERIC (p, s) – числа, с указанным общим количеством цифр p и количеством дробных знаков s;
- REAL – вещественное число (4 байта);
- FLOAT – вещественное число двойной точности (8 байт);
- INTEGER – целое (4 байта) от -2 147 483 648 до +2 147 483 648;
- SMALLINT – целое (2 байта) от -32768 до +32767.

## 6.2. Создание и изменение структуры таблицы

Команда создания таблицы записывается следующим образом

*CREATE TABLE <имя таблицы> (<колонка> [, <колонка>…]*
*[, <табличное ограничение>[,<табличное ограничение> ...]])*

Пункт <колонка> задает описание одной колонки (поля) таблицы
*<колонка> ::= <имя колонки > <Тип данных>*
*[ <свойства и ограничение колонки>]*

Типом данных может быть любой из перечисленных в предыдущем параграфе.

Свойства и ограничения могут быть следующими:

– Пункт DEFAULT задает значения по умолчанию, например
COUNTRY VARCHAR(20) default ‘Российская федерация’.
– Пункт IDENTITY(n,s) определяет автоинкрементное поле, значения которого формируется автоматически с начальным значением n и увеличивается при добавлении записи на s.
– Свойство NULL (NOT NULL) разрешает (запрещает) задавать пустые значения поля.
– Ограничение колонки *CHECK (<условие>)* задает условие, образованное при помощи операций: конъюнкции (and) дизъюнкции (or) и отрицания (not), скобок и сравнений вида
*<выражение> <знак сравнения> <выражение>*

Кроме этого, возможно употребление условий, использующих команду выбора *SELECT* (см. описание команды ниже).

Значение NULL является специальной меткой и не совпадает с пустой строкой или нулем. Пустые значения могут породить проблемы при вычислении, когда один из операндов равен нулю. Результат выражения с NULL-значением может быть снова NULL или NULL-значение интерпретируется как ноль, в зависимости от параметров сервера. Другая ситуация возникает при сравнении двух NULL-значений: NULL-значения считаются не равными друг другу.

Табличное ограничение
*CONSTRAINT <имя ограничения> < ограничение>*
может быть следующим:

– Ограничение «первичный ключ»
*PRIMARY KEY (<список имен колонок через запятую>)*
указывает, что данная последовательность полей является первичным ключом, т.е. не допускается существования двух записей с одинаковыми значениями первичного ключа. В базе данных создает индекс для повышения скорости поиска по первичному ключу. Поля, составляющие первичный ключ не могут принимать пустые значения (обязательно должны быть NOT NULL).
– Ограничение «Внешний ключ»
*FOREING KEY (<имена колонок таблицы через запятую>)*
*REFERENCES <имя справочной таблицы>*
*(< имена колонок справочной таблицы через запятую >)*

указывает, что значения перечисленных полей обязательно должны быть в одной из записей справочной таблицы. В таблице нельзя определить запись с несуществующими значениями, а в справочной таблице нельзя удалить запись, если на значения указанных полей в записи есть ссылки. В справочной таблице должен быть определен первичный ключ по указанным полям.

– Ограничение
*CHECK(<условие>)*
позволяет определить ограничение в виде произвольного логического выражения.

Команда изменения структуры таблицы имеет следующий вид

*ALTER TABLE <таблица> <изменения через запятую>*

Изменение описывается фразой

*ADD <колонка>* |
*ADD <Табличное ограничение >* |
*DROP <колонка>l*
*DROP CONSTRAINT <ограничение>*

Пример, команд (скрипта) создания таблицы [Договоры] с рис. 12:

```
CREATE TABLE [Договоры] (
    [Дата] [smalldatetime] NOT NULL ,
    [Номер договора] [int] NOT NULL ,
    [Код Продавца] [int] NOT NULL ,
    [Предоплата] [float] NOT NULL )
ALTER TABLE [Договоры] ADD
    CONSTRAINT [PK_Договоры] PRIMARY KEY  CLUSTERED ([Номер
договора])
ALTER TABLE [Договоры] ADD
    CONSTRAINT [FK_Договоры_Продавцы] FOREIGN KEY
    ([Код Продавца]) REFERENCES [Продавцы] ([Код продавца])
```

Команда удаления таблицы

*DROP TABLE <имя таблицы>*

удаляет таблицу со всеми записями и ограничениями.

Для таблицы можно определить несколько индексов. При определении первичного ключа индекс строится автоматически. Индекс является высоко эффективным средством поиска записей в таблице по значению. Основой организации индекса обычно является конструкция B-дерева. B-дерево состоит из страниц. Каждая страница содержит n ключей и n+1 указателей на страницы-потомки, причем 0-й потомок содержит ключи меньшие 1-го ключа, k-й потомок содержит ключи из интервала (k-й ключ, (k+1)-й ключ), (n+1)-й потомок содержит ключи большие n-го ключа (см. рис. 13).


**Рис. 12. Пример схемы базы данных**

Страница может быть незаполненной. Если все страницы заполнены полностью, то при добавлении значения ключа приходится выполнять добавление страниц и реорганизацию дерева. Неполные страницы позволяют уменьшить количество реорганизаций дерева.

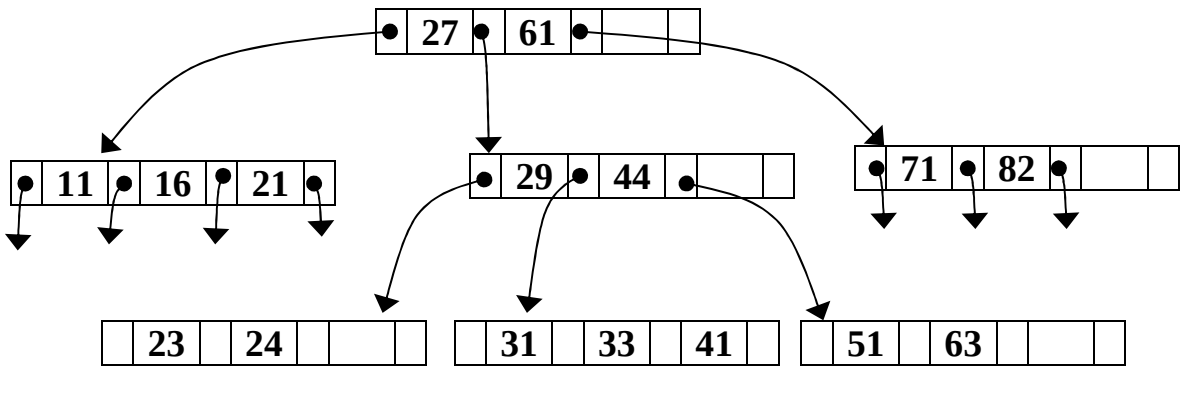



Рис. 13. Фрагмент B-дерева

Поиск ключа начинается с корневой страницы. Если ключ на странице не найден, то однозначно определяется станица-потомок, на которой поиск следует продолжить. Максимальное количество просматриваемых страниц равно высоте дерева. С увеличением количества сложность поиска растет как логарифм количества ключей, причем основанием логарифма является среднее количество ключей на странице. Например, при среднем количестве ключей на странице 10 для поиска ключа среди миллиона достаточно просмотреть всего lg(1 000 000)=6 страниц, а при увеличении количества ключей в 10 раз придется просматривать всего на одну страницу больше.

Один из индексов может быть кластерным –на терминальных страницах ключи хранятся вместе с записью. Кластерный индекс не требует дополнительных расходов памяти, но для таблицы можно определить только один кластерный индекс. Некластерные индексы требуют память для хранения.

Индекс создается командой

*CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX <индекс>*

*ON <таблица>* ***(*** *<поля>* ***)***

*[ WITH < опции > [ ,...n] ]*

*[ ON <группа_файлов> ]*

Параметры команды означают следующее:

- *UNIQUE* – обеспечивает создание уникального индекса,
- *CLUSTERED* – записи хранятся в индексном порядке, страницы-листья содержат данные. У таблицы может быть только один кластерный индекс.
- *NONCLUSTERED* – индекс хранится в отдельных страницах. Последовательность записей не совпадает с индексной.
- *<индекс>* – уникальное имя в рамках БД для указания индекса.
- *<поля>* – список полей через запятую. После каждого поля можно указывать направление сортировки *ASC* – по возрастанию или DESC – по убыванию. Индексный порядок записей соответствует их сортировке сначала по первому полю, затем по второму и так далее.
- *< Опции >* могут быть следующими
  - *PAD_INDEX* – резервирует свободное пространство для индексных записей в момент создания, употребляется вместе с параметром *FILLFACTOR.*
  - *FILLFACTOR = <процент заполнения>* – указывает долю заполнения пространства индексных записей на каждой странице. Используется при создании таблицы. Если эта доля равна 100%, то ключами заполняется все пространство. Это экономит память, однако, при добавлении или удалении записей потребуется перекомпоновка страниц на пути от листовой страницы до корневой. Если доля меньше 100%, то такая перекомпоновка требуется реже.
  - *IGNORE_DUP_KEY* – игнорирует появление дублирующих значений ключа.
  - *DROP_EXISTING* – новый ключ заменяет одноименный существующий.
  - *STATISTICS_NORECOMPUTE* – отменяет рекомпиляцию статистики, используемой для оптимизации выполнения запроса.

– *SORT_IN_TEMPDB* – указывает использование БД временных данных для построения индекса. Это может повысить скорость создания индекса, если эта БД расположена о другом диске или в оперативной памяти.
– *ON <группа_файлов>* – физически размещает индекс в указанной группе.

Индекс также можно создать при помощи диалоговых команд в программе Enterprise Manager.

При правильном построении и использовании индексы обеспечивают более высокую производительность выполнения запросов.

## 6.3. Команды изменения содержания таблицы

Команда добавления записей в таблицу

*INSERT INTO <таблица> [(<колонка> [, <колонка> ...])]*
*{VALUES (<значение> [, <значениеl> ...]) | <select - выражение>}*

может не иметь списка полей добавляемых записей. В этом случае должны быть заданы значения всех полей. Добавление может быть задано списком констант. Например,

*INSERT INTO [Продавцы]*
*([Код продавца],[Форма собственности],[Продавец],[Город],[Банк])*
*VALUES (12, 'АОЗТ', 'Прогресс', 'Ангарск', 'Надежный')',*

Добавляемые записи также могут быть определены командой выбора SELECT.

Команда

*DELETE FROM <таблица> [WHERE <условие>]*

удаляет все записи удовлетворяющие условию. Например,

*DELETE FROM [Продавцы] WHERE [Код продавца]=12*

Команда модификации записей

*UPDATE <Таблица> SET <колонка> = <выражение> [,<колонка> = <*
*выражение >...] [WHERE <условие>]*

изменяет значения полей указанными выражениями для всех записей, удовлетворяющих условию. Например,

*UPDATE [Продавцы] SET [Банк] = 'Универсал'*
*WHERE [Банк] = 'ВСКБ'*

переводит продавцов-клиентов банка «ВСКБ» в банк «Универсал»

## 6.4. Выбор данных

Команда SELECT выбирает данные в виде некоторой таблицы и имеет следующий формат

*SELECT [DISTINCT] <список выражений>*
*[INTO <таблица-результат>]*

*FROM < соединение таблиц>*
*[WHERE <условие>]*
*[Group BY <список выражений группировки>*
*[HAVING <условие выбора группы>]]*
*[ORDER BY <список полей сортировки>]*

### 6.4.1. Определение полей таблицы, формируемой запросом

Список выражений, приведенных после ключевого слова *SELECT* определяет состав и последовательность колонок в результирующей таблице. Ключевое слово *DISTINCT* означает, что в этой таблице все записи будут уникальны – дубликаты записей будут вычеркнуты. Выражения (в простейшем случае поля) перечисляются через запятую. Выражение определяется фразой

*<выражение> [AS <имя поля>]*

в котором после слова «AS» указывается имя поля в итоговой таблице (можно не указывать, если выражение является просто полем).

Выражение может быть представлено

- *полем* базы данных в формате [<имя | псевдоним таблицы>.]<поле>. Имя или псевдоним таблицы необходимо указывать, если поле входит состав нескольких соединяемых таблиц;
- *константой* (указанное значение константы появится в каждой записи выходной таблицы);
- *выражением*, построенным при помощи констант, полей, функций, знаков операций и скобок;
- *статистической функцией*, вычисляемой по группе записей:
  *AVG(<выражение>)* – среднее значение выражения;
  *COUNT(*)* – число записей, *COUNT(<поле>)* – число непустых значений поля, *COUNT(DISTINCT <поле>)* – число различных значений поля;
  *MIN(<выражение>)* – наименьшее значение выражения;
  *MAX(<выражение>)* – наибольшее значение выражения;
  *SUM(<выражение>)* – сумма выражений.

Статистическая функция вычисляется либо по группе записей исходной таблицы в случае группировки данных, либо по всей исходной таблице в запросе без группировки.

Выражение может быть арифметическим – для его построения используют операторы + , -, *, /, % (остаток от деления), скобки и функции. Набор арифметических функций является обычным для языков программирования, например

*CEILING(<число>)* – возвращает ближайшее большее целое число,
*FLOOR(<число>)* – возвращает ближайшее меньшее целое число,

*RAND([<число>])* – возвращает псевдослучайное равномерно распределенное в интервале [0,1) число, следующее за указанным,

*ROUND (<число> , <число знаков дробной части> [ , <числовой тип>] )* – возвращает число, округленное до указанного числа знаков.

Для работы с датами предусмотрены следующие функции:

*DATEADD (<единица измерения>, <число>, <дата>)* – возвращает дату увеличенную на заданное число единиц (единица может быть годом (y), кварталом (q), месяцем (m), днем (d) или более мелкой).

*DATEDIFF (<единица измерения> , <начальная дата>, <конечная дата>)* – возвращает количество единиц времени прошедших с начальной до конечной даты.

*DATEPART (<единица измерения>, < дата>)* – возвращает указанную часть даты.

*GETDATE ( )* – возвращает дату сервера.

*YEAR(< дата>), MONTH(< дата>), DAY(< дата>*) – возвращают год, месяц, день месяца указанной даты.

Для формирования строковых выражений используется операция сцепления (конкатенации) обозначаемая знаком «+» и следующие функции:

*CHARINDEX ( <подстрока> , <строка> [ , <начальная позиция> ] )* – возвращает позицию подстроки в строке. Поиск начинается с начальной позиции или с первого знака, если начальная позиция не указана. Если подстрока не найдена функция возвращает ноль.

*SUBSTRING ( <строка> , <начало> , <длина> )* – возвращает подстроку указанной длины с указанного начального знака.

Есть функции, предназначенные для определения длины строки *LEN* преобразования прописных букв в строчные *LOWER* и наоборот *UPPER*, замены одной подстроки на другую *REPLACE* и другие.

Следующие функции связаны с проверками значений:

*ISNULL ( <выражение> , <замена> )* – вместо выражение возвращается указанное значение замены, если значение выражения равно NULL.

*COALESCE (<выражение> [ , <выражение> ]* ) – возвращает первое не NULL-значение из списка.

*CASE WHEN <условие> THEN <выражение>*
*[WHEN <условие> THEN <выражение> … ]*
*[ELSE <else-выражение> ] END* – возвращает первое значение, для которого условие окажется истинным. Если такового не окажется, возвращается значение else-выражения.

Функция CONVERT(<тип данных>,<выражение>[,<тип даты>]) преобразует выражение к указанному типу.

### 6.4.2. Соединение таблиц в запросе

В пункте *FROM* определяется соединения таблиц, из которых извлекается информация. Для этого может использоваться оператор соединения

*<табличный источник> <оператор соединения> <табличный источник>*
*ON <условия соединения>*

Табличный источник состоит из имени таблицы, за которым можно через пробел указать локальный псевдоним, заменяющий имя таблицы только внутри команды SELECT. Псевдонимы позволяют использовать несколько экземпляров одной и той же таблицы в одном запросе. Вместо таблицы может быть другая команда SELECT, заключенная в круглые скобки. В этом случае псевдоним должен быть обязательно. Если псевдоним не указан, то он полагается равным имени таблицы. Условие соединения обычно имеет вид равенств

*<псевдоним таблицы1>.<поле1>=<псевдоним таблицы2>.<поле2>*

Для формулировки условия используются логические операторы AND, OR, NOT.

Условия соединения могут быть указаны и в пункте WHERE. В этом случает в пункте FROM таблицы просто перечисляются через запятую. Соединение в SELECT аналогично оператору соединения в реляционной алгебре, то есть каждая запись первой таблицы комбинируется с теми записями второй таблицы, которые удовлетворяют условию соединения. Каждая такая комбинация является основой для построения записи результирующей таблицы. Например, соединение

*Договоры O INNER JOIN Продавцы S*
*ON O.[Код Продавца] = S.[Код продавца]*

к каждой записи о договоре добавляет запись о соответствующем продавце.

Оператор соединения имеет следующие варианты

*[INNER] JOIN* – обычное (внутреннее) соединение в описанном выше смысле,

*[LEFT | RIGHT | FULL] [OUTER] JOIN* – левое | правое | полное внешнее соединение – в результат включаются даже те записи левой | правой | обеих таблиц, которым не соответствую записи другой таблицы,

*CROSS JOIN* – декартово произведение таблиц.

Для указания последовательности соединения можно использовать скобки.

### 6.4.3. Фильтрация строк

Условие из пункта WHERE должно содержать условия выбора строк – результата соединения таблиц (и условия соединения таблиц, если они не указаны в пункте FROM). Для формулировки условия могут использовать-

ся сравнения и логические операции: AND – конъюнкция (“логическое И”); OR – дизъюнкция (“логическое ИЛИ”); NOT – отрицание.
Например, в запросе

```
SELECT S.Продавец, O.Дата, O.[Номер договора],
 T.Товар, C.Количество, T.[Единица измерения],
 T.[Вес ЕдИзм(Кг)], C.Цена, C.Количество*C.Цена AS [Стоимость]
FROM Договоры O
 INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи C
  ON O.[Номер договора] = C.[Номер договора]
 INNER JOIN Товары T ON C.[Код товара] = T.[Код товара]
WHERE (S.Город = 'Иркутск') AND (T.Товар = 'Рис')
```

из базы данных извлекается информация о поставках риса из Иркутска.

Для формулировки условий применяются следующие специальные формы сравнений.

Условие

*<выражение> [NOT] BETWEEN <начало> AND <конец>*

истинно, если значение выражения лежит (не лежит в случае указания NOT) в указанном диапазоне. Например, для выделения договоров с номерами от 101 до 200 можно было бы воспользоваться условием

*[Номер договора] BETWEEN 101 AND 200*

Условие

*< выражение > [NOT] LIKE <шаблон>*

проверяет для значения выражения соответствие шаблону (несоответствие в случае указания NOT). В шаблоне можно использовать знаки: “_” (подчеркивание), означающий подстановку одного любого символа, “%”, заменяющий произвольную последовательность символов. Например, условие

*Товар LIKE “Конфет%”*

выбирает все товары, названия которых начинаются с "Конфет".

Следующие формы сравнения используют ***подзапросы*** – *запросы, вложенные в основной запрос для вычисления некоторых значений*. Подзапросы определяются командой SELECT.

Условие

*<выражение> <сравнение> ALL (<подзапрос>)*

истинно, если сравнение выполняется для всех данных, извлеченных подзапросом. Например, запрос

```
SELECT Продавцы.Продавец, Товары.Товар, Продажи.Количество,
 Товары.[Единица измерения], Продажи.Цена
FROM  Договоры
 INNER JOIN Продавцы
```

```
 ON Договоры.[Код Продавца] = Продавцы.[Код продавца]
 INNER JOIN  Продажи
 ON Договоры.[Номер договора] = Продажи.[Номер договора]
 INNER JOIN Товары
 ON Продажи.[Код товара] = Товары.[Код товара]
WHERE Продажи.Цена< ALL
(
SELECT C.Цена
FROM Договоры O
 INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи C
ON O.[Номер договора] = C.[Номер договора]
WHERE (S.Продавец='Гермес') AND
 C.[Код товара] =  Продажи.[Код товара]
)
```

выбирает поставки, в которых цена товаров ниже, чем любые цены на такие же товары поставщика 'Гермес'.

Условие

*<поле> <сравнение> ANY (<подзапрос>)*

истинно, если сравнение выполняется хотя бы для одного значения, извлеченного подзапросом. Для извлечения поставок, в которых цена товара не превышает цен на такие же товары поставщика с 'Гермес'., можно воспользоваться предыдущим запросом, заменив в нем *ALL* на *ANY*.

Результат сравнения

*EXIST (<подзапрос>)*

истина, если результат подзапроса является непустым множеством. Команда

```
 SELECT *
 FROM Товары
 WHERE NOT EXISTS
 (
 SELECT *
 FROM Договоры O
  INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
  INNER JOIN Продажи C
 ON O.[Номер договора] = C.[Номер договора]
 WHERE (S.Продавец='Гермес') AND
  C.[Код товара] =  Товары.[Код товара]
 )
```

выделят все товары, которые не поставлял «Гермес».

Сравнение

*<поле> [NOT] IN (<набор значений>)*

истинно, если значение поля входит (не входит, если указано NOT) в набор значений. Набор значений может быть указан перечислением, например,

*[Код продавца] IN (5, 7)*

или подзапросом, например, условие

```
[Код товара] IN
(
SELECT DISTINCT C.[Код товара]
FROM Договоры O
 INNER JOIN Продавцы S ON O.[Код Продавца] = S.[Код продавца]
 INNER JOIN Продажи C
ON O.[Номер договора] = C.[Номер договора]
WHERE (S.Продавец='Гермес')
)
```

означает принадлежность кода множеству кодов товаров, проданных «Гермесом».

### 6.4.4. Группировка

Фраза *GROUP BY* позволяет сгруппировать записи после объединения таблиц и фильтрации. Список выражений группировки (после фразы *GROUP BY*) состоит из выражений, разделенных запятыми. В группу попадают записи с одинаковыми значениями выражений группировки. В выходной таблице каждая группа представлена одной записью. По каждой группе можно вычислить агрегированные показатели. Запрос

```
SELECT Товары.Товар, SUM(Продажи.Количество) AS Количество,
 Товары.[Единица измерения],
 AVG(Продажи.Цена) AS [Средняя цена],
 SUM(Продажи.Количество*Продажи.Цена) AS Стоимость
FROM  Продажи INNER JOIN  Товары
ON Продажи.[Код товара] = Товары.[Код товара]
GROUP BY Товары.Товар, Товары.[Единица измерения]
```

создает таблицу, в которой для каждого товара вычисляется количество, средняя цена и стоимость по всем поставкам данного товара.

Условие выбора группы, указанное в пункте HAVING, определяет условие включения итогов вычисления по группе в результат запроса. Изменим запрос из предыдущего примера, так чтобы выбрать товары, для которых количество поставок было больше 5, добавив фразу

*HAVING COUNT(*)>5*

Условие выбора группы предназначено прежде всего для проверки агрегированных значений. Для уменьшения затрат на вычисление запроса

рекомендуется обычные условия (не связанные с агрегированными значениями) указывать в пункте *WHERE*. В список колонок запроса с группировкой можно помещать только те выражения, которые представлены в списке полей группировки и результаты агрегирования.

Если фраза *GROUP BY* отсутствует, а список выражений, определяющих колонки таблицы-результата запроса, включает только константы и функции агрегирования, то вычисления выполняются по всем записям и результат будет представлен одной записью.

### 6.4.5. Сортировка записей результата

Записи выходной таблицы можно упорядочить, если в команду включить пункт ORDER BY. Элементом списка является выражение, за которым может следовать слово ASC для упорядочения по возрастанию или DESC для упорядочения по убыванию. Порядок записи выражений сортировки определяет последовательность сортировки: сначала выполняется сортировка по первому выражению, затем записи с одинаковым значением первого выражения сортируются по второму выражению и так далее.

### 6.4.6. Объединение записей нескольких запросов

Для объединения таблиц предусмотрен оператор UNION

*< запрос> UNION [ ALL ] <запрос>[UNION [ ALL ] <запрос>…]*
*[ORDER BY <выражения сортировки> ]*

объединяющий строки всех запросов в одну таблицу. Запросы должны удовлетворять ограничениям:

- Количество и порядок полей в запросах должны быть идентичными.
- Типы данных полей должны быть сопоставимы.

Дублирующие записи вычеркиваются из объединение, если только не указано ключевое слово ALL. Если предусмотрена сортировка, то упорядочивается результат объединения.

### 6.4.7. Сохранение результата запроса

Наличие фразы

*[INTO <таблица-результат>]*

определяет сохранение таблицы-результата под указанным именем.

Обычно используют сохранение во временные таблицы локальные или глобальные. Имя локальной временной таблицы должно начинаться с одного диеза («#»), имя глобальной – с двух диезов. Временные таблицы существуют на протяжении сеанса соединения сервера с пользователем (или в течение работы процедуры, функции или триггера, в которых они определены). Локальные таблицы доступны только в рамках сеанса (процедуры, функции или триггера), глобальные таблицы доступны всем пользователям. Временные таблицы и локальные, и глобальные автоматически

уничтожаются по окончании сеанса связи (процедуры, функции или триггера).

Гораздо реже результат запроса сохраняется как обычная (не временная) таблица базы данных. Во-первых, для этого необходим специальный режим использования базы данных, во-вторых, пользователь, выполняющий такой запрос должен иметь права на создание таблиц в базе данных.

## 6.5. Определение представлений пользователей

Представление пользователя *View* – поименованный запрос, который компилируется, оптимизируется и хранится в БД. Представление может использоваться вместо таблицы в командах выбора и изменения данных. Представления обычно применяются для хранения алгоритмов выбора данных. Кроме этого, представления являются средством достижения гибкости – при изменении схемы данных достаточно разработать представления для отображения «новых» таблиц в «старые», и избежать, таким образом, переделки «старых» запросов.

Представление пользователя View создается командой

*CREATE VIEW <имя представления> [(<колонки через запятую>)]*
*AS <запрос> [WITH CHECK OPTIONS]*

Представление пользователя позволяет выполнять изменения (DELETE, UPDATE, INSERT) если выполнены следующие условия запроса-определения View:

- нет выбора уникальных значений (отсутствует ключ DISTINCT),
- в качестве колонок таблицы-результата используются только имена,
- в пункте FROM используется только одна таблица,
- при фильтрации в пункте WHERE не использует подзапросы,
- в запросе нет группировки.

Вариант WITH CHECK OPTIONS – запрещает ввод данных, не удовлетворяющих условию WHERE.

## 6.6. Transact – SQL

Язык Transact – SQL является расширением языка SQL для разработки процедур, функций и триггеров. Эти объекты позволяют определять сложные алгоритмы модификации и выборки данных.

Кроме описанных выше команд Transact – SQL включает следующие конструкции:

- Объявление локальных переменных (переменные в MS SQL должны начинаться с символа «@»)

  *DECLARE <переменная> <тип данных>*
- Оператор присваивания

  *SELECT | SET <переменная>=<выражение>,…*

– Операторные скобки (блок команд)

*BEGIN*
*<Команды>*
*END*

– Проверка условия

*IF <условие>*
*{<команда> | <блок>}*
*[ELSE*
*{<команда> | <блок>}]*

– Цикл

*WHILE <условие>*
*{<команда> | <блок>}*

Внутри блока команд цикла можно употреблять команду *BREAK* для выхода из цикла и команду *CONTINUE* для повторения цикла с начала (с проверки условия выполнения цикла).

– Безусловный переход

*GOTO <метка>*
выполняет переход на помеченный оператор
*<метка>: <оператор>*

– Комментарий

*/* <Комментарий>*/*
*--* <Комментарий>

– Средства описания и вызова хранимых процедур, триггеров, функций.
– Команда выполнения команд SQL, заданных строковой переменной

*Exec (<выражение строкового типа> )*

Для получения результатов выполнения команд можно использовать системные функции, начинающиеся со знаков «@@»:

– @@ERROR – код завершения последней команды (0 – нормальное завершение команды),
– @@ROWCOUNT – количество записей обработанных последней командой.

## 6.7. Хранимые процедуры

Хранимые процедуры – это скомпилированные подпрограммы на языке Transact – SQL. Они делятся на два класса:

– процедуры выбора, формирующие таблицу как результат вызова;
– выполняемые процедуры, осуществляющие некоторую модификацию данных и/или возвращающие вычисленные выходные параметры.

Иногда процедура выполняет и то, и другое.

Процедура создается командой

*CREATE PROCEDURE <имя процедуры> [*
*(<@параметр> <тип> [=<значение_по умолчанию>] [OUTPUT], …)]*
*AS <команда> / <блок>*

Процедура вызывается командой

*[[EXEC[UTE]] <имя процедуры>*
*[[<@параметр> =] {<значение> | <@переменная> [OUTPUT]]*

Для формирования таблицы в качестве результата работы процедуры среди ее команд должна быть команда выбора SELECT.

Пример: процедура для заполнения поля «Стоимость договора» в таблице «Договоры» на основании включенных в договор товаров из таблицы «Продажи»

```
/************
  Процедура формирует поле "Стоимость" в таблице "Договоры",  вычисляя суммарную стоимость продаж договора с номером, заданным параметром @N. Пусто (NULL) значение параметра приводит к формированию  стоимости всех договоров
**************/
CREATE PROCEDURE  UpdateCost (@N int = NULL) AS
BEGIN
--Создание временной таблицы стоимостей
SELECT [Номер договора], SUM(Количество*Цена) AS Стоимость
  INTO #A
FROM Продажи
WHERE (@N is NULL) --вычисляется стоимость для всех договоров
 OR ([Номер договора]=@N) -- или указанного
GROUP BY [Номер договора]
-- Обновление поля стоимость по данным временной таблицы
  UPDATE Договоры SET Стоимость =#A.Стоимость
    FROM #A
    WHERE Договоры. [Номер договора] = #A. [Номер договора]
END
```

В процедуре создается временная таблицы *#A* содержащая стоимости договоров, определенных параметром *@N* процедуры

## 6.8. Триггеры

Триггер – процедура без параметров, определяемая для операции Update, Insert, Delete или их комбинации над определенной таблицей. Команда определения триггера имеет следующий формат

```
CREATE TRIGGER <имя триггера>
    ON { <имя таблицы>| <имя представления>}
    { FOR | AFTER | INSTEAD OF }
```

*{*[DELETE] [,] [INSERT] [,] [UPDATE]*}*

*AS*

*<команды Transact – SQL >*

Триггер запускается при выполнении перечисленных в его определении изменений до их фиксации в базе данных (триггер «For» или «After») или вместо фиксации (триггер «Instead of»). Для программирования изменений в зависимости от произведенных действий в теле триггера можно использовать две таблицы:

– *Deleted* – содержит удаляемые или изменяемые записи (данные до операции),
– *Inserted* – содержит измененные или вставленные записи (данные после операции).

До выполнения операции проверяются ограничения целостности, которые могут и не допустить выполнения триггера. Следующий триггер «For»

```
CREATE TRIGGER КаскадноеУдалениеДоговора ON [Договоры]
FOR DELETE
AS
  DELETE FROM Продажи WHERE [Номер договора] IN
      (SELECT [Номер договора] FROM Deleted)
```

выполняющий каскадное удаление из таблицы «Продажи» записей, связанных с удаляемыми договорами, не будет выполнен, если определен внешний ключ для таблицы «Продажи» на таблицу «Договоры» и есть записи, ссылающиеся на удаляемый договор.

При наличии упомянутого внешнего ключа можно воспользоваться триггером «Instead of»

```
CREATE TRIGGER КаскадноеУдалениеДоговора ON [Договоры]
INSTEAD OF DELETE
AS
  DELETE FROM Продажи WHERE [Номер договора] IN
      (SELECT DISTINCT [Номер договора] FROM Deleted)
  DELETE FROM Договоры WHERE [Номер договора] IN
      (SELECT DISTINCT [Номер договора] FROM Deleted)
```

В этом случае приходится программировать удаление записей из таблицы «Договоры» (вторая команда DELETE), так как команда пользователя по удалению договоров не выполняется, а только моделируется: записи определенные пользователем для удаления помещаются в таблицу Deleted.

Для проверки изменения некоторого поля применяется условие

*UPDATE ( <имя поля> )*

которое возвращает значение «истина», если указанное поле было изменено командой UPDATE.

Триггеры являются эффективным средством для поддержки бизнес-правил. Например, при проведении бухгалтерских операций должно выполняться равенство сумм по дебету и кредиту. Такие правила не могут быть реализованы на основе механизма ссылочной целостности. Для регистрации и проводки хозяйственной операции в таблице «Операции» можно разработать триггер, который буде выполнять проводки, модифицируя соответствующие счета.

Триггеры допускают вложенный вызов. Например, при удалении поставщика запускается триггер, который удаляет договоры данного поставщика. Удаление договоров вызывает в свою очередь триггер для удаления продаж, включенных в удаляемые договоры.

Рекурсивный вызов триггера означает, что триггер определяет модификации, которые снова приводят к его вызову. Рекурсивный вызов возможен при включении соответствующего параметра базы данных.

## 6.9. Курсоры

Курсоры предоставляют возможность обрабатывать таблицы по записям. Существует определенная схема использования курсоров:

1) **Сначала курсор надо объявить**

*DECLARE <имя курсора> CURSOR FOR <команда выбора select>*

2) **Для использования курсора он открывается**

*OPEN <имя курсора>*

При этом выполняется запрос и создается его результат – временная таблица, которую можно просматривать.

3) **Просматривается курсор по записям обычно в цикле**

Команда просмотра выполняет позиционирование записи и извлечение полей в переменные

*FETCH {NEXT | PRIOR | FIRST | LAST | ABSOLUTE <номер записи>*
*| RELATIVE <смещение>}*
*FROM <имя курсора> INTO <@переменная_1>,…, <@переменная_n>*

где n – число полей. Команда извлекает поля указанной записи и присваивает указанным переменным. В цикле просмотра записей обычно используется глобальная переменная @@FETCH_STATUS, которая содержит код завершения команды FETCH:

| | |
|---|---|
| 0 | успешно, |
| -1 | конец таблицы, |
| -2 | прочитана удаленная запись. |

4) **После окончания обработки записей курсора необходимо закрыть временную таблицу командой**

*CLOSE <имя курсора>*

При этом снимаются все блокировки, освобождается таблица курсора и ее снова можно открыть командой *OPEN*.

**5) Удаление ссылки на курсор.**

При удалении последней ссылки освобождаются ресурсы.

*DEALLOCATE <имя курсора>*

Команда

*DEALLOCATE <имя курсора>*

разрушает связь переменной *<имя курсора>* и определением курсора.

Пример:

```
/*******************
  Процедура вычисляет изменение объема продаж
  в процентах по отношению к предыдущему месяцу
*******************/
CREATE PROCEDURE IncreaseByMonth AS
BEGIN
-- Временная таблица для хранения вычислений
CREATE TABLE #I
(       Year int,
        Month int,
        Cost float,
        Increment float)
-- Курсор для вычисления объемов продаж по месяцам
DECLARE Costs CURSOR FOR
SELECT Year(Договоры.Дата) AS [Year],
   Month(Договоры.Дата) AS [Month],
   SUM(Количество*Цена) AS Cost
FROM Продажи INNER JOIN Договоры
   ON Продажи.[Номер договора] = Договоры.[Номер договора]
GROUP BY Year(Договоры.Дата), Month(Договоры.Дата)
ORDER BY Year(Договоры.Дата), Month(Договоры.Дата)
-- Открывается курсор – вычисляется запрос
OPEN Costs
-- Объявление переменных
DECLARE @Y int, @M int, @C0 float, @C float
-- извлечение первой записи курсора
FETCH NEXT FROM Costs INTO @Y, @M, @C0
-- формирование первой записи результата
INSERT #I VALUES( @Y, @M, @C0, NULL)
-- Цикл по записям курсора
FETCH NEXT FROM Costs INTO @Y, @M, @C
WHILE (@@FETCH_STATUS <> -1)
```

```
BEGIN
        -- формирование очередной записи результата
        INSERT #I
        VALUES( @Y, @M, @C,(@C-@C0)/@C0*100)
        -- извлечение очередной записи курсора
        FETCH NEXT FROM Costs INTO @Y, @M, @C
END
-- закрытие курсора
CLOSE Costs
-- Ликвидация курсора
DEALLOCATE Costs
-- Вывод результата
SELECT * FROM #I
END
```

### 6.10.Переменные табличного типа и функции

В MS SQL Server 2000 появились новые средства для увеличения гибкости и выразительности языка. В нем можно использовать локальные переменные, которые имеют табличный тип.

*DECLARE <@переменная> TABLE ( <определение полей и ограничений как в команде CREATE TABLE> )*

Областью действия переменной, как обычно, являются процедура, функция или пакет команд.

Эти нововведения позволили определять функции, которые возвращают таблицы. Определение табличной функции с помощью нескольких команд выглядит следующим образом

```
CREATE FUNCTION <имя функции>
   ( [<@параметр> [AS] <тип> [ = <значение по умолчанию>] [,…]] )
RETURNS <@переменная> TABLE <определение таблицы>
 [ AS ]
BEGIN
  <команды>
   RETURN
END
```

Следующая функция заменяет систему представлений – одно представление для каждого города поставщика.

```
CREATE FUNCTION ПродавцыИзГорода (@Town nVarChar(50))
RETURNS @s TABLE (
        [Код продавца] [int] NOT NULL ,
        [Форма собственности] [nvarchar] (10) NULL ,
        [Продавец] [nvarchar] (30) NULL ,
```

```
    [Город] [nvarchar] (30) AS NULL ,
    [Банк] [nvarchar] (30) NULL) AS
BEGIN
    INSERT @s
    SELECT * FROM Продавцы Where Город=@Town
    RETURN
END
```

Вызов функции, вычисляющей таблицу, возможен вместо употребления имени таблицы, например

```
select * from ПродавцыИзГорода ('Иркутск')
```

## 6.11.Ограничения языка SQL

Простота реляционной модели является основой ее эффективности. Она же определяет ее основные ограничения.

Прежде всего страдает выразительность языка – в нем нет агрегатов (обобщений). Нельзя объединить набор атрибутов в единое понятие (например, адрес) и манипулировать им как единым целым.

Проблематично в SQL напрямую реализовывать современные тенденции, например объектное описание данных. Отчасти по причине отсутствия обобщений, отчасти по причине отсутствия средств наследования и инкапсуляции.

Ряд проблем создает ограниченность самого языка. Можно найти максимальную стоимость продаж, но найти продавца с максимальной стоимостью продаж алгоритмически сложнее. Другой проблемой SQL является невыразимость транзитивного замыкания, т.е. невозможно построить запрос, когда нужно найти для определенного элемента все связанные с ним элементы. Например, всех вышестоящих начальников, а не только непосредственного, всех потомков, а не только детей, и т.д. В таких запросах количество соединений является неопределенным, что могло бы быть реализовано отсутствующим в SQL рекурсивном соединении.

### Вопросы

1. Почему SQL является декларативным языком?
2. Перечислите типы данных языка SQL и укажите область применения каждого.
3. Что включает определение таблицы в языке SQL?
4. Перечислите и определите свойства и ограничения колонок. Приведите примеры.
5. Почему пустые значения могут привести к ошибкам?
6. Перечислите и определите табличные ограничения. Приведите примеры.

7. Какие средства предлагает язык SQL для поддержания ссылочной целостности и реализации бизнес-правил?
8. Что такое индекс таблицы? Зачем и когда создаются индексы таблицы?
9. Перечислите и определите команды изменения содержания таблицы. Приведите примеры.
10. Перечислите основные фразы команды SELECT. Что, в какой форме, на основании чего выбирает эта команда?
11. Как определяются поля выбираемой таблицы в команде SELECT?
12. Перечислите типы выражений в команде SELECT, укажите правила их построения.
13. Для чего и как применяются статистические функции (функции агрегирования) в команде SELECT?
14. Перечислите правила и возможности соединения таблиц в команде SELECT.
15. Как выполняется фильтрация в команде SELECT?
16. Перечислите условия с подзапросами. Приведите примеры.
17. Как определяется группировка в команде SELECT? Что и как группируется? Что является результатом группировки?
18. В чем разница условий, задаваемых в пунктах *WHERE* и *HAVING*?
19. Перечислите возможности сортировки в команде SELECT.
20. Как можно осуществить объединение результатов нескольких запросов?
21. Укажите назначение и использование представлений пользователя в языке SQL.
22. Для чего используется язык Transact – SQL? Какие команды он включает?
23. Укажите назначение и использование хранимых процедур, триггеров, курсоров и функций в языке SQL.
24. Как в теле триггера можно определить, какие изменения выполнены пользователем?
25. Когда запускается триггер?

# 7. Технология «Клиент-Сервер»

## 7.1. Технология и модели сетевой обработки данных

"Клиент-сервер" – это модель взаимодействия компьютеров в сети. Компьютер (процесс), управляющий тем или иным ресурсом, принято называть сервером этого ресурса, а компьютер (процесс), использующий ресурс – клиентом.

В технологии «клиент-сервер» в качестве сервера подразумевается сервер баз данных. Взаимодействие клиента и сервера представлено на рис. 14. Клиентская программа преобразует действия клиента в команду SQL и передает команду посредством клиентской части SQL сервера и сетевых библиотек в сеть. Сервер через сетевые библиотеки получает команду и передает ее ядру SQL сервера, которое и выполняет SQL-команды и обратным порядком отправляет результат клиенту для отображения результата на компьютере пользователя.

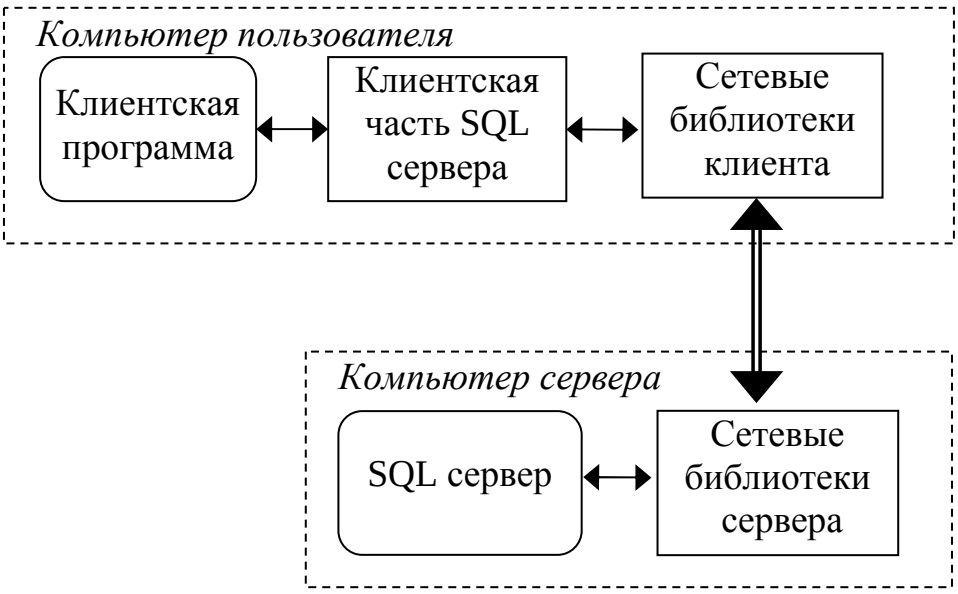



Рис. 14. Взаимодействие клиента и сервера баз данных в технологии «клиент-сервер»

В технологии "клиент-сервер" часть функций прикладной программы реализована в программе-клиенте, другая – в программе-сервере. Функции стандартного интерактивного приложения можно разделить на три группы:

– презентационная логика – включает интерфейсные функции (задание команд и получение результатов их выполнения) – реализуется на клиенте,

– бизнес-логика – правила корректировки и обработки данных, вытекающие из логики учета и управления – может быть реализована как на клиенте, так и на сервере,
– логика доступа – функции доступа к данным – реализуется сервером.

Варианты архитектуры, основанные на делении работ между клиентом и серверами, представлены в таблице 1.

Таблица 1

Многоуровневые модели «Клиент-сервер»

<table>
<tr><td rowspan="2">Уровни логики</td><td rowspan="2">«Файл-сервер»</td><td colspan="2">«Клиент-сервер»</td></tr>
<tr><td>Толстый клиент</td><td>Тонкий клиент</td></tr>
<tr><td>Презентационная логика</td><td rowspan="3">Клиент</td><td rowspan="2">Клиент</td><td>Клиент</td></tr>
<tr><td>Бизнес - логика</td><td rowspan="2">Сервер приложений, Сервер БД</td></tr>
<tr><td>Логика доступа к ресурсам</td><td>Сервер БД</td></tr>
</table>

В модели «Файлов-сервер» в приложении совмещены все виды логики. Протокол обмена представляет собой набор низкоуровневых вызовов, обеспечивающих приложению доступ к файловой системе на файл-сервере.

В модели клиент-сервер БД презентационная логика и часть бизнес-логики совмещены и выполняются на компьютере-клиенте. Другая, общезначимая часть бизнес-логики оформлена как набор хранимых процедур и триггеров и функционирует на компьютере-сервере БД. Доступ к информационным ресурсам обеспечивается операторами языка (SQL).

Реализация бизнес-логики на сервере имеет следующие достоинства (знак «+») и недостатки (знак «-»):

+ гарантия выполнения,
+ доступ к правилам и их использование всеми клиентами,
+ быстродействие за счет применения мощных серверов и уменьшения трафика сети,
+ мобильность,
+ удобство администрирования,
– правила не могут использовать несколько БД,
– сложность интерпретации сообщений сервера об ошибках,

Достоинства и недостатки реализации бизнес-логики в программе-клиенте в определенном смысле противоположны предыдущему варианту:

+ возможен высокий уровень оперативного контроля, использующий в том числе и диалог с пользователем,

– замедление работы клиента и увеличение трафика сети за счет передачи данных клиенту для обработки,
– повторная реализация бизнес-правил для разных приложений-клиентов,
– неудобно собирать все бизнес-правила в один свод, так как они разбросаны по всему тексту приложения или по текстам нескольких приложений,
– незащищенность БД от изменений без соблюдения правил некорректно разработанными клиентами.

## 7.2. Обработка транзакций

Для описания единицы работы в БД вводится понятие транзакции. *Транзакция* – это логическая единица работы с базой данных. Для SQL-сервера транзакция представляет собой последовательность операторов языка SQL, которая рассматривается как некоторое неделимое действие над базой данных.

Транзакция характеризуется четырьмя классическими свойствами ACID (Atomicity, Consistency, Isolation, Durability): атомарности, согласованности, изолированности, долговечности (прочности):

1. Свойство ***атомарности*** выражается в том, что транзакция должна быть выполнена целиком или не выполнена вовсе.
2. Свойство ***согласованности*** гарантирует, что выполнение транзакции переводит данные из одного согласованного состояния в другое – транзакция не разрушает согласованности данных, хотя может ее нарушать в процессе выполнения.
3. Свойство ***изолированности*** означает, что результат выполнения транзакции не зависит от выполняющихся в тоже время других транзакций. Для этого конкурирующие за доступ к данных транзакции обрабатываются по определенным правилам.
4. Свойство ***долговечности*** трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны в результате сбоев.

Команда:

*BEGIN TRAN[ACTION]*

начинает транзакцию – все последующие команды образуют блок команд транзакции. Заканчивается транзакция командой

*COMMIT TRAN[SACTION]* или *COMMIT WORK*

которая предписывает серверу зафиксировать все изменения выполненные командами транзакции или всех начатых клиентом транзакций. Второй вариант – завершение транзакции командой

*ROLLBACK TRAN[SACTION]* или *ROLLBACK WORK*
заставляет сервер отменить все изменения, сделанные транзакцией или всеми начатыми клиентом транзакциями.

Сервер работает по умолчанию в режиме *автоматического начала транзакций*, в котором каждая команда рассматривается как транзакция, если нет явных команд определения транзакции. Этот режим устанавливается командой

*SET IMPLICIT_TRANSACTION OFF*

Другой режим *неявного начала транзакций* (устанавливается командой *SET IMPLICIT_TRANSACTION ON*) автоматически начинает новую транзакцию, если закончена предыдущая командой фиксации или отката.

Откат и фиксация транзакций становятся возможными благодаря журналу транзакций. При выполнении любого оператора SQL, который вносит изменения в базу данных, СУБД автоматически заносит сведения об этом в журнал транзакций. Только после записи в журнал транзакций СУБД действительно вносит изменения в базу данных. Если после очередного оператора SQL был выполнен оператор COMMIT, то в журнале транзакций делается отметка о завершении текущей транзакции. Если же после оператора SQL следовал оператор ROLLBACK, то СУБД просматривает журнал транзакций и отыскивает в нем записи, отражающие состояние измененных строк до внесения изменений. Используя их, СУБД восстанавливает те строки в таблицах базы данных, которые были изменены текущей транзакцией, и таким образом отменяет все изменения, выполненные транзакцией.

Одной из основных задач многопользовательских СУБД является организация одновременного доступа к одним и тем же данным множества пользователей. Проблемы коллективного доступа разбивают на следующие группы: потеря изменений, чтение грязных данных (незафиксированных изменений), неповторяющееся чтение, появление данных-«фантомов».

***Потеря изменений*** происходит в ситуации, когда две или несколько транзакций читают одни и те же данные из базы данных, вносят в них какие-либо изменения и затем пытаются одновременно записать результат по прежнему месту. Разумеется, в базе данных могут быть сохранены изменения, выполненные только одной транзакцией, – другие изменения будут потеряны.

***Проблема чтения грязных данных (незафиксированных изменений)*** возникает в случае, когда в процессе выполнения одной транзакции в данные были внесены изменения и тут же прочитаны другой транзакцией, однако затем в первой программе транзакция была прервана оператором ROLLBACK. Получается, что вторая программа прочитала неверные, «грязные», незафиксированные данные.

***Неповторяющиеся чтение*** встречается когда первая транзакция выполняет повторное чтение данных, которые были измены второй транзакцией, между первым и вторым чтением. Это повторное чтение первой транзакцией считывает данные, отличные от прочитанных в первый раз.

***Появление данных-«фантомов»*** возникает, когда одна транзакция читает данные, а другая добавляет данные. Добавленные данные являются невидимыми – «фантомами» – для первой транзакции.

Стандарт ANSI предусматривает четыре уровня изолированности транзакций:

- ***Read Uncommited***. Нулевой уровень изолированности – запрет потерянных изменений. Не допускается обновление данных, пока не закончится первая транзакция, обновляющая эти данные.
- ***Read Commited***. Первый уровень изолированности – нет потерянных и незафиксированных изменений и «грязного чтения». Не допускается чтение данных, пока не закончится первая транзакция, обновляющая эти данные.
- ***Repeatable Read***. Второй уровень изолированности – запрещено неповторяемое чтение незафиксированных данных и потерянные изменения. Не допускается обновление и чтение данных, пока не закончится транзакция, прочитавшая эти данные.
- ***Serializable***. Третий уровень изолированности – нет фантомов и других конфликтов. Не допускается обновление и чтение таблиц, пока не закончится транзакция, обновляющая или читающая эти таблицы.

Уровень изолированности транзакций устанавливается командой:

```
SET TRANSACTION ISOLATION LEVEL
{  READ UNCOMMITED
   READ COMMITED
   REPEATABLE READ
   SERILIZABLE  }
```

Автоматический откат транзакции устанавливается командой:

```
SET XACT_ABORT ON
```

При возникновении ошибки будет произведен откат всех команд транзакции. Если автоматический откат транзакций отключен, то будет отменена только последняя команда, которая вызвала ошибку.

Чем выше уровень изолированности, тем меньше возможностей для параллельного выполнения транзакций. Основным механизмом достижения изолированности является механизм блокировок. Транзакция запрашивает захват объекта в одном из режимов: совместный (S – Shared) или монопольный (X – eXclusive). Для реляционных БД объектами захвата могут быть

- база данных;

– файл: несколько отношений с индексами;
– отношение;
– страница данных: несколько кортежей, индексная страница;
– запись.

Чем крупнее объект, тем меньше возможностей распараллеливания выполнения транзакций и меньше расходы на управление блокировками.

Одной из проблем механизма блокирования является «тупик» или «мертвая блокировка» – DeadLock. «Тупик» возникает, если две транзакции захватили разные объекты и каждая пытается захватить объект, заблокированный другой транзакцией. Сервер БД имеет встроенный механизм разрешения этой проблемы, если блокировки не заданы явно специальной командой.

## 7.3. Обработка распределенных данных

Главная проблема больших систем – организация обработки распределенных данных. Данные могут находятся на компьютерах различных моделей и производителей, функционирующих под управлением различных операционных систем, а доступ к данным может осуществляться разнородным программным обеспечением. Сами компьютеры могут быть территориально удалены друг от друга и находиться в различных географических точках. Решение этой проблемы предлагают две технологии: технология распределенных баз данных (транзакций) и технология тиражирования данных.

**Технология распределенных баз данных.**

Под распределенной базой данных подразумевают базу данных, включающую фрагменты, которые располагаются на различных серверах компьютерной сети, и, возможно, управляются различными СУБД. Тем не менее, распределенная база данных должна выглядеть с точки зрения пользователей и прикладных программ как обычная база данных.

В распределенных базах транзакция, выполнение которой заключается в обновлении данных на нескольких серверах сети, называется глобальной или распределенной транзакцией.

Для пользователя распределенной базы данных глобальная транзакция выглядит как обычная Для этого в современных СУБД предусмотрен так называемый протокол двухфазной фиксации транзакций.

Фаза 1 начинается, когда транзакция фиксируется. Специальный монитор распределенных транзакций направляет уведомление «подготовиться к фиксации» всем серверам, выполняющим распределенную транзакцию. Если все серверы приготовились к фиксации, монитор распределенных транзакций принимает решение о фиксации. Если хотя бы один из серверов не откликнулся на уведомление в силу каких-либо причин, будь то ап-

паратная или программная ошибка, то монитор распределенных транзакций откатывает транзакции на всех серверах, включая даже те, которые подготовились к фиксации и оповестили его об этом.

Фаза 2 – монитор распределенных транзакций направляет команду «зафиксировать» всем серверам, затронутым транзакцией, и гарантирует, что транзакции на них будут зафиксированы. Если связь с сервером потеряна в интервал времени между моментом, когда монитор распределенных транзакций принимает решение о фиксации транзакции, и моментом, когда сервер подчиняется его команде, то монитор распределенных транзакций продолжает попытки завершить транзакцию, пока связь не будет восстановлена.

**Технология тиражирования данных**

Принципиальное отличие технологии тиражирования данных от технологии распределенных баз данных заключается в отказе от синхронной обработки транзакции всеми серверами. Ее суть состоит в том, что любая БД (как для СУБД, так и для работающих с ней пользователей) всегда является независимой от данных, расположенных на других серверах. Все транзакции в системе завершаются на одном сервере. Это приводит к хранению на каждом сервере своей копии общих данных и периодической синхронизации изменений, внесенных на разных серверах.

Тиражирование данных – это асинхронный перенос изменений исходной базы данных в базы данных, размещенные на различных серверах распределенной системы. Функции тиражирования данных выполняет специальный модуль СУБД – сервер тиражирования данных, называемый репликатором. Его задача – поддержка идентичности данных в принимающих базах данных данным в исходной базе.

В качестве базиса для тиражирования выступает транзакция. В то же время возможен перенос изменений группами транзакций, периодически или в некоторый момент времени.

Технология распределенных БД и технология тиражирования данных – в определенном смысле антиподы. Краеугольный камень первой – синхронное завершение транзакций одновременно на нескольких узлах распределенной системы, то есть синхронная фиксация изменений в распределенной БД. "Ахиллесова пята" этой технологии – жесткие требования к производительности и надежности каналов связи. Если БД распределена по нескольким территориально удаленным узлам, объединенным медленными и ненадежными каналами связи, а число одновременно работающих пользователей составляет десятки и выше, то вероятность того, что распределенная транзакция будет зафиксирована в обозримом временном интервале, становится чрезвычайно малой. В таких условиях обработка распределенных данных практически невозможна.

Реальной альтернативой технологии распределенных БД становится технология тиражирования данных, не требующая синхронной фиксации изменений (и в этом ее сильная сторона). В действительности далеко не во всех задачах требуется обеспечение идентичности БД на различных узлах в любое время. Достаточно поддерживать тождественность данных лишь в определенные критичные моменты времени. Следовательно, можно накапливать изменения в данных в виде транзакций в одном узле и периодически копировать эти изменения на другие узлы.

Просуммируем очевидные преимущества технологии тиражирования данных. Во-первых, данные всегда расположены там, где они обрабатываются – следовательно, скорость доступа к ним существенно увеличивается. Во-вторых, передача только операций, изменяющих данные (а не всех операций доступа к удаленным данным, как в технологии распределенных БД), и к тому же в асинхронном режиме, позволяет значительно уменьшить трафик. В-третьих, со стороны исходной БД для принимающих БД репликатор выступает как процесс, инициированный одним пользователем, в то время как в физически распределенной среде с каждым локальным сервером работают все пользователи распределенной системы, конкурирующие за ресурсы друг с другом. Наконец, в-четвертых, никакой продолжительный сбой связи не в состоянии нарушить передачу изменений. После восстановления связи передача возобновляется с той транзакции, на которой тиражирование было прервано.

Технология тиражирования данных не лишена некоторых недостатков, вытекающих из ее специфики. Например, невозможно полностью исключить конфликты между двумя версиями одной и той же записи. Он может возникнуть, когда вследствие все той же асинхронности два пользователя на разных узлах исправят одну и ту же запись в тот момент, пока изменения в данных из первой базы данных еще не были перенесены во вторую. Следовательно, при проектировании распределенной среды с использованием технологии тиражирования данных необходимо предусмотреть конфликтные ситуации и запрограммировать репликатор на какой-либо вариант их разрешения.

**Вопросы**

1. Перечислите и укажите назначение видов логики в сетевой системе регистрации и обработки данных. Где может быть реализован каждый вид логики в системах «файл-сервер» и «клиент-сервер»?
2. Перечислите достоинства и недостатки реализации бизнес-логики на стороне клиента и на стороне сервера.
3. Понятие и свойства транзакции.

4. Перечислите и опишите использование команд начала и завершения транзакций.
5. Как реализуется механизм отката транзакций?
6. Опишите основные проблемы коллективного доступа к данным.
7. Перечислите и опишите уровни изолированности пользователей. Какие проблемы коллективного доступа они решают?
8. Как и что блокируется транзакциями в БД для обеспечения каждого уровня изолированности?
9. Опишите основные принципы, достоинства и недостатки технологий распределенных БД и тиражирования.

# 8. MS SQL сервер. Общие сведения

## 8.1. Компоненты MS SQL сервер

MS SQL сервер работает в среде операционной системы MS NT server и реализован в виде нескольких сетевых служб:

1) MSSQLServer – основное ядро сервера, реализует функции доступа к данным, только эта служба необходима для доступа к данным.
2) SQLServerAgent обеспечивает автоматическое выполнение заданий, извещение персонала об ошибках.
3) Microsoft Search (MSSearch) выполняет поиск символьной информации в полях таблиц БД.
4) Microsoft Distributed Transaction Coordinator (MS DTC) управляет распределенными транзакциями (любая транзакция, использующая таблицы из разных БД, реализована как распределенная).

Каждая служба сервера запускается под некоторой учетной записью. Рекомендуется такие учетные записи включать в группу Administrators и для автоматического запуска и давать учетным записям права Log on as a service, Act as a part of the operating system.

Предусмотрены следующие системные БД:

– Master – хранение параметров конфигурации сервера, login-ов пользователей и других системных данных;
– Msdb – используется SQLServerAgent для хранения информации о заданиях, операторах и оповещениях;
– Model – шаблон БД, который копируется в каждую создаваемую БД;
– Tempdb – БД для хранения временных таблиц.

Любая БД содержит 18 системных таблиц для хранения метаданных.

Взаимодействие клиента и MS SQL сервер выполняется через специальные интерфейсы программирования (API) доступа к базам данных:

– ODBC (Open Database Connectivity) набор функций для доступа к табличным источникам данных. ODBC был использован для построения более совершенных и удобных драйверов Remote Data Objects (RDO), DataAccess Objects (DAO),
– DB-Library – библиотека для доступа к серверу,
– OLE DB – новая открытая спецификация для доступа к данным,
– ADO (ActiveX Data Object) – эволюционное развитие RDO и DAO.

Все API реализованы в виде dll-файлов, которые взаимодействуют с MS SQL сервер через сетевые библиотеки клиента и сервера.

Службы сервера используют следующие каталоги:

– Backup – резервное копирование,
– Binn – программы,

– Books – документация,
– Data – базы данных,
– FtData – индексы MS Search,
– DevTolls – средства разработки ODBC, DB-library,
– Html – документы html, используемые Enterprise Manager,
– Install – описание инсталляции сервера,
– Jobs – временные файлы SQLServerAgent,
– Log – сообщения об ошибках,
– Repldata – для тиражирования,
– Upgrade – для обновления предыдущих версий.

Группа программ Microsoft SQL Server позволяет быстро получить доступ к основным инструментам и вспомогательным утилитам SQL-сервера:

– Books Online – подробная документация по SQL-серверу;
– Client Network Utility – утилита, позволяющая выполнить настройку клиентских сетевых библиотек;
– Enterprise Manager – основная утилита, при помощи которой администратор осуществляет управление серверами баз данных;
– Import and Export Data – мастер Data Transformation Service Wizard, который можно использовать для интеграции SQL-сервера с другими источниками данных;
– Performance Monitor – системная утилита, позволяющая реализовать наблюдение за использованием SQL-сервером системных ресурсов (таких, например, как процессор, оперативная память, диск и т. д.);
– Profiler – инструмент, предоставляющий администратору широкие возможности для осуществления мониторинга отдельных аспектов функционирования SQL-сервера;
– Query Analyzer – инструмент, при помощи которого администратор может выполнять управление базами данных средствами Transact-SQL;
– Server Network Utility – утилита, позволяющая выполнить настройку серверных сетевых библиотек;
– Service Manager – утилита, при помощи которой администратор управляет всеми службами SQL-сервера;
– Uninstall SQL Server – мастер, который должен выполнить удаление SQL-сервера в случае, если в этом возникнет необходимость.

## 8.2. Использование программы Enterprise Manager администрирования MS SQL сервера

При наличии соответствующих прав Enterprise Manager позволяет выполнять создание базы данных, таблиц, представлений пользователя, процедур, триггеров и других объектов базы данных, управлять подключениями

и правами пользователей, получать информацию о текущей работе сервера – словом выполнять все работы по администрированию сервера, используя привычный для пользователей Windows графический интерфейс (см. рис. 15)

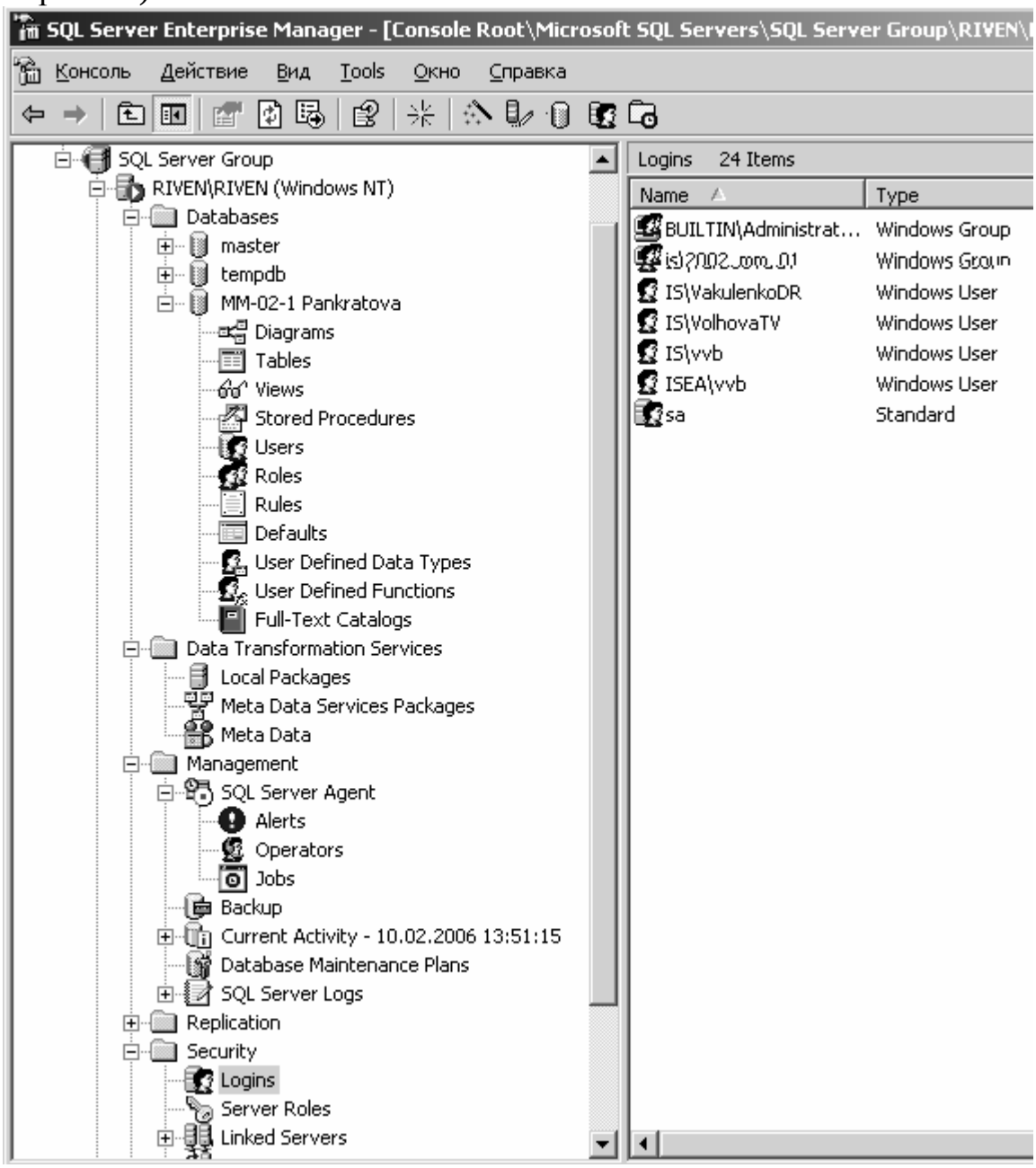



Рис. 15. Окно программы Enterprise Manager

Для выполнения большинства функций достаточно на левой панели указать группу объектов или объект и правой кнопкой мыши вызвать контекстное меню. Группа объектов Databases содержит список всех баз данных сервера. Data Transformation Service предоставляет средства экспорта и импорта данных. Management отображает компоненты управления сер-

вером: задания и сопутствующие объекты, определенные для службы SQL Server Agent состояние резервных копий (Backup), текущие подключения к серверу (Current Activity), планы резервного копирования (Database Maintenance Plans) и журналы событий (SQL Server Logs). Security позволяет просматривать и управлять подключением к серверу пользователей и других серверов.

### 8.3. Система безопасности

Подключение к SQL-серверу выполняется по регистрационному имени (login) пользователя в сети. Это сетевое имя должно быть зарегистрировано на SQL-сервере. При этом возможна аутентификация независимая от Windows NT или интегрированная с Windows NT. Второй вариант удобней в администрировании, так как сетевое имя и пароль запрашиваются и проверяются контроллером домена при регистрации пользователя, и при успешной аутентификации пользователь получает определенные права на SQL сервере.

Права на уровне сервера присваиваются пользователю сети, включением его в одну из стандартных ролей сервера (см. таблицу 2). Кроме этого пользователь сети может получить определенные права в каждой базе данных сервера.

**Таблица 2**

**Стандартные роли на уровне сервера**

| Встроенная роль сервера | Назначение |
|---|---|
| Sysadmin | Может выполнять любые действия в SQL сервере. |
| Serveradmin | Выполняет конфигурирование и выключение сервера |
| Setupadmin | Управляет связанными серверами и процедурами, автоматически запускающимися при запуске SQL сервера |
| Securityadmin | Управляет учетными записями и правами на создание базы данных, также может читать журнал ошибок. |
| Processadmin | Управляет процессами, запущенными в SQL сервере. |
| Dbcreator | Может создавать и модифицировать базы данных. |
| Diskadmin | Управляет файлами SQL сервера |

Каждая база данных имеет свой список пользователей БД – user-ов (не путать с пользователями сети – login-ами). Пользователь сети (login) может быть отображен в определенного пользователя БД (user-а). Один пользователь БД создается в момент создания БД и не может быть удален – это владелец БД (dbo). Он обладает всеми полномочиями в БД и раздает права

всем остальным пользователям. Системный администратор SQL сервера (Sysadmin) отображается во владельца (dbo) в каждой базе сервера.

Для удобства администрирования в БД вводятся роли (группы пользователей). Права и запреты на операции с объектами БД обычно раздаются ролям, хотя могут назначаться и непосредственно пользователям. В дополнении к персональным правам и запретам пользователь получает права и запреты ролей, в которые он включен. В случае противоречия прав и запретов последние имеют приоритет. Если пользователь персонально или благодаря включению в некоторую роль получает право выполнять команду, а другая его роль содержит запрет на выполнение этой команды, то в результате пользователю будет запрещено выполнять эту команду.

Предусмотрены стандартные роли базы данных (см. в таблицу 3).

**Таблица 3**

**Стандартные роли пользователя базы данных**

| Встроенная роль базы данных | Назначение |
|---|---|
| db_owner | Имеет все права в базе данных |
| db_accessadmin | Может добавлять или удалять пользователей |
| db_securityadmin | Управляет всеми разрешениями, объектами, ролями и членами ролей |
| db_ddladmin | Может выполнять любые команды DDL, кроме GRANT, DENY и REVOKE |
| db_backupoperator | Может выполнять команды DBCC, CHECKPOINT и BACKUP |
| db_datareader | Может просматривать любые данные в любой таблице БД |
| db_datawriter | Может модифицировать любые данные в любой таблице БД |
| db_denydatareader | Запрещается просматривать данные в любой таблице |
| db_denydatawriter | Запрещается модифицировать данные в любой таблице |

Для создания или управления подключениями пользователей к серверу используется список Security\Logins. Как правило, пользователь не получает полномочий на уровне сервера (не включается в стандартные роли на уровне сервера). Для него определяют базы данных, с которыми он будет работать при выполнении служебных обязанностей, и отображают в определенного пользователя (user-a). На рис. 16 login пользователя ISEA\BelkovaEG отображается в одноименного пользователя базы Academia, включенного в группы просмотра данных факультета (Faculty_browser) и кафедры (Kaf_browser).

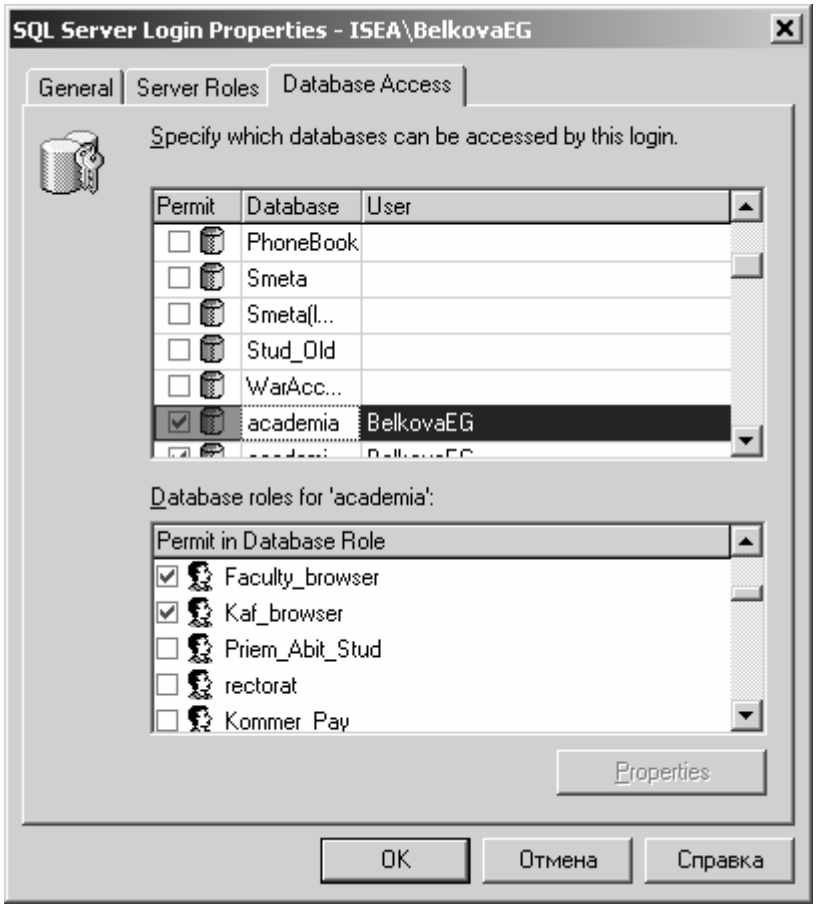



Рис. 16. Параметры (свойства) подключения (login-а) к серверу

Права и запреты определяются для объектов БД (таблицы, представления, процедуры, табличные функции) и соответствующих операций SELECT, INSERT, UPDATE, DELETE, EXECUTE. Полномочия предоставляются командой Manage permissions в контекстном меню объекта, пользователя или группы (см. рис. 17). Для операций SELECT и UPDATE можно определить права и запреты на уровне колонок (кнопка Columns). В дополнении к персональным правам и запретам пользователь получает права и запреты ролей, в которые он включен. Конфликт права и запрета на выполнение какой-либо операции решается в пользу запрета.

Кроме перечисленных, предусмотрены права на специальные операции, такие как создание объектов БД. Эти права прописываются в свойствах базы данных.

Полномочия имеет право предоставлять владелец (создатель) объекта (таблицы, представления, табличной функции).

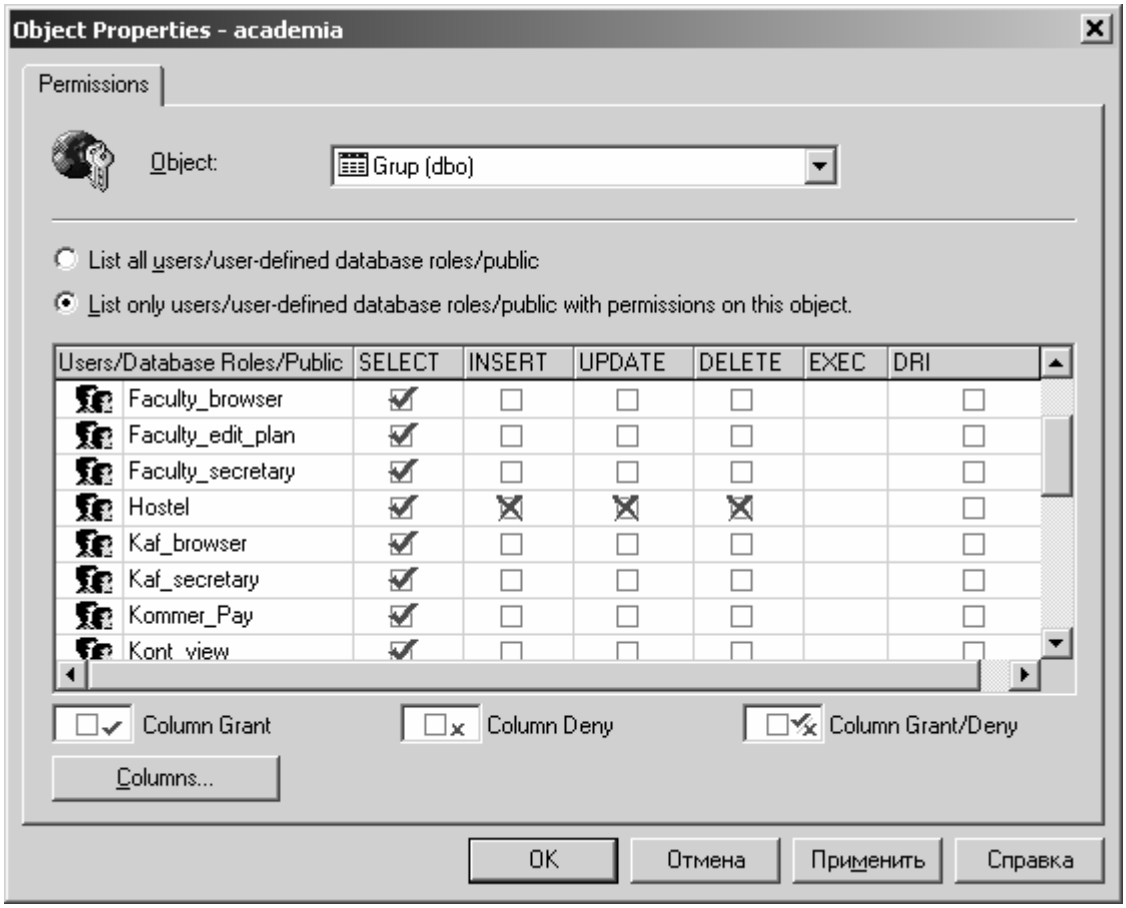



Рис. 17. Форма определения прав выполнения операций с объектами БД

## 8.4. Резервное копирование и восстановление БД

Резервное копирование баз данных является необходимым компонентом системы защиты данных. Обычно резервное копирование выполняется периодически и предусматривает копирование всей БД и/или журнала транзакций во время минимальной загруженности сервера. Например, еженедельное копирование БД и ежедневное копирование журнала. В случае жесткого сбоя, повлекшего потерю содержания основного носителя БД, данные восстанавливаются на момент последнего резервного копирования.

Копирование журнала транзакций позволяет восстанавливать изменения БД по журналу и копии БД не только на момент копирования, но также на любой момент времени (переключатель «point in time restore» на рис. 19). Кроме этого, копия журнала транзакций более компактна, и ее создание требует меньше ресурсов.

Копирование можно запустить командой Backup в контекстном меню базы данных в программе Enterprise Manager (см. рис. 18). Обычно резервное копирование выполняют автоматически при помощи специального задания для MS SQL Agent. Создается такое задание при помощи мастера ре-

зервного копирования (Backup Wizard) или специальной формы (включается переключателем Schedule и соответствующей кнопкой – см. рис. 18). Основным параметром на автоматическое резервное копирование является время запуска копирования. Напомним, что такие задания выполняются службой SQLServerAgent, которая для этого должна быть запущена.

Резервное копирование кроме создания копии БД приводит к отчистке журнала транзакций.

Кроме копирования всей БД и журнала транзакций MS SQL сервер предусматривает разностное (Differential) копирование – копирование страниц, измененных с момента последнего копирования – и копирование выбранных групп файлов.

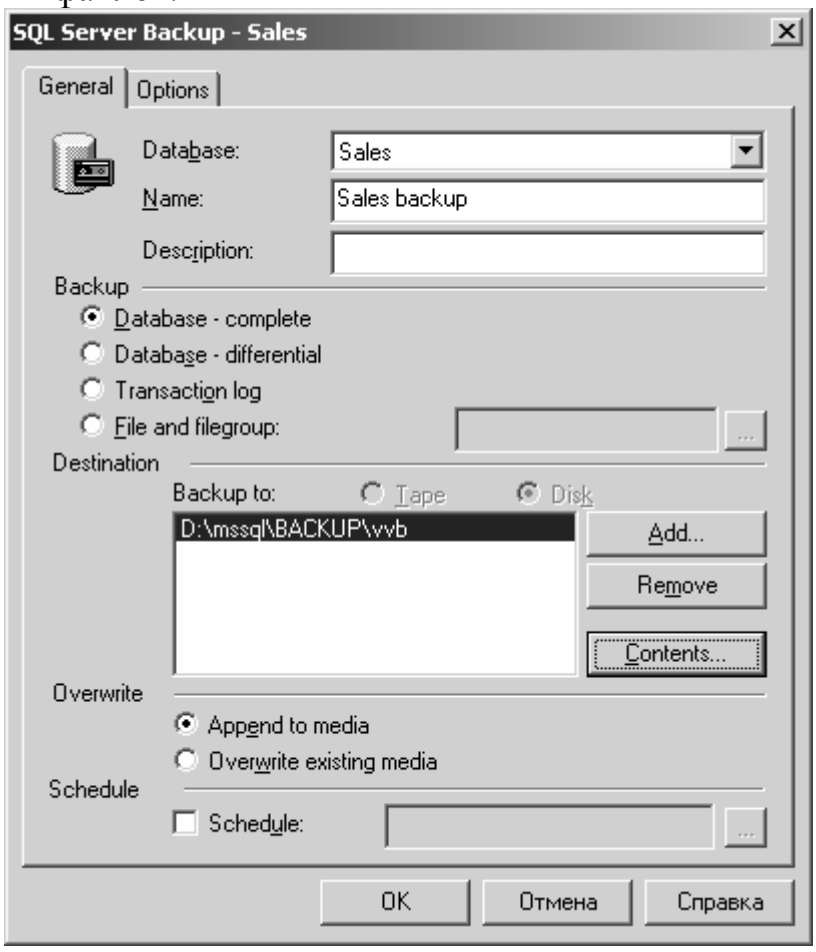



Рис. 18. Определение параметров команды копирования в Enterprise Manager

Восстановление базы данных может быть выполнено командой Restore. Параметры этой команды представлены на рис. 19.

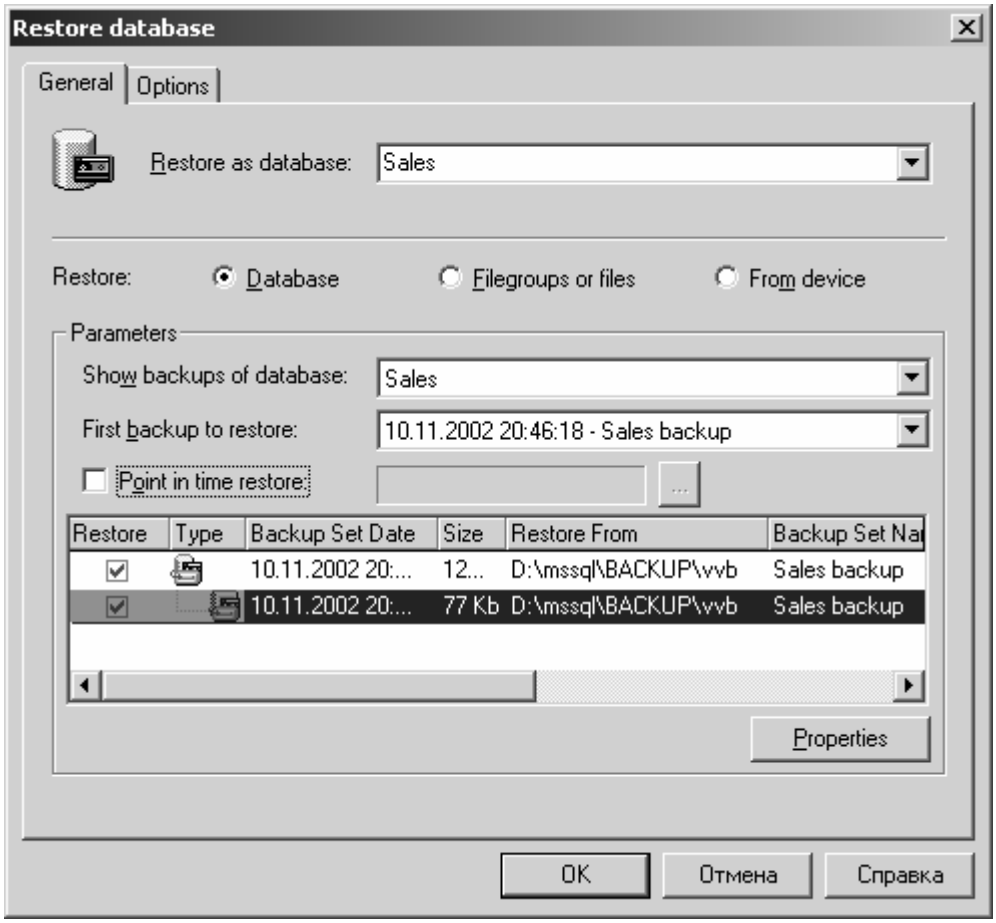



**Рис. 19.** Определение параметров команды восстановления в Enterprise Manager

## 8.5. Прочие возможности контроля объектов базы данных

Многие возможности администрирования можно использовать в программных объектах БД в виде специальных команд, системных процедур и функций. Например, команда Grant определяет права пользователя. Системные процедуры начинаются с префикса SP_ и предназначены для выполнения действий по настройке системы и получения свойств и параметров объектов базы данных. Многие процедуры являются командным вариантом выполнения функций администрирования. Например, команда sp_adduser добавляет нового пользователя БД.

Ниже перечислены системные функции по определению пользователей сети и базы данных:

– *UserName()* – имя пользователя БД,
– *IS_MEMBER ('<имя роли>')* – проверяет принадлежность пользователя указанной роли,

– *SUSER_SID ( [ '<логин>' ] )* возвращает уникальный sid (security identification number) пользователя сети,
– *SUSER_SNAME ( [ <sid пользователя> ] )* возвращает логин пользователя.

Эти функции можно применять в процедурах, функциях, триггерах для определения кто выполняет соответствующие действия, и на этой основе выполнять более сложные правила предоставления полномочий. Например, ограничивать по времени выполнение некоторых операций определенным ролям и пользователям.

Для получения статистики выполнения запросов можно использовать программу Query Analyzer. Это бывает полезно для оптимизации выполнения запросов. Следующие команды включают или отключают вывод специальной информации:

*SET SHOWPLAN {ON | OFF}* – плана выполнения запросов,
*SET STATISTICS IO {ON | OFF}* – статистики ввода-вывода,
*SET STATISTICS PROFILE {ON | OFF}* – подробной статистики о выполнении запроса,
*SET STATISTICS TIME {ON | OFF}* – сведений о времени выполнения плана.

Кроме этого, можно в меню Query включить или отключить вывод:
– плана выполнения запросов – Show Execution Plan,
– статистики о выполнении запроса на сервере – Show Server Trace,
– статистики о получении данных на клиентом – Show Client Statistics.

Статистика позволяет выделить наиболее ресурсоемкие части запросов.

Наблюдение взаимодействия клиента и сервера можно выполнить при помощи программы SQL Profiler. Для этого надо входить в группу администраторов сервера. Программа SQL Profiler обеспечивает выполнение следующих работ:
– Контроль работы SQL сервера.
– Отладка SQL команд и процедур.
– Аудит работы SQL сервера.

Обычно настройка трассировкив SQL Profiler заключается в следующем:
– Запускаетя SQL Profiler.
– Стартуется новая трассировка. При необходимости указывается запись трассировки в файл.
– На вкладке Event выбирают события, отображаемые в трассировке.
– На вкладке Data Columns выбирают поля, описывающие события.
– На вкладке Filters установают фильтрацию событий (например, по имени базы данных, login-у пользователя и т.п.).

**Вопросы**

1) Перечислите и укажите назначение сетевых служб MS SQL сервера.
2) Перечислите и укажите назначение системных баз данных MS SQL сервера.
3) Перечислите и укажите назначение основных утилит MS SQL сервера.
4) Опишите систему безопасности MS SQL сервера. Как происходит аутентификация?
5) Кто и каким образом получает полномочия на уровне сервера? Какие это полномочия?
6) Кто и каким образом получает права на уровне базы данных? Какие это права?
7) Как соотносятся понятия «пользователь сети», «группа сети», «пользователь БД», «роль БД»?
8) Опишите встроенные роли и пользователей БД.
9) Какие права, для каких объектов могут получать пользователи?
10) Как определяются права и запреты пользователя, если он является членом нескольких ролей и обладает персональными правами и запретами?
11) Перечислите виды резервного копирования, укажите их назначения и условия их применения.
12) Перечислите преимущества резервного копирования журнала транзакций.
13) Как можно получить сведения об особенностях и статистике выполнения запросов?
14) Как получить сведения о взаимодействии пользователей с SQL сервером?

# 9. Системы оперативной аналитической обработки данных (OLAP)

## 9.1. Основные понятия

Принять любое управленческое решение невозможно, не обладая необходимой для этого информацией, обычно представленной спектром показателей, описывающих состояние и динамику бизнес-процесса. Показатели должны вычисляться «влет» по произвольным аспектам деятельности. Для этого был создан специальный инструмент – хранилища данных (Data warehouses) – инструмент для сбора, отсеивания и статистического анализа данных с целью предоставления результирующей информации пользователям в виде удобных аналитических отчетов.

Приведем определение, сформулированное «отцом-основателем» хранилищ данных Биллом Инмоном: «**Хранилище данных** – это предметно-ориентированное, привязанное ко времени и неизменяемое собрание данных для поддержки процесса принятия управляющих решений».

Данные в хранилище попадают из оперативных систем (OLTP-систем), которые предназначены для автоматизации бизнес-процессов. Кроме того, хранилище может пополняться за счет внешних источников, например различных статистических отчетов.

Почему неудобно использовать для этих целей SQL-серверы или другие СУБД? Причин несколько:

1. Необходим специальный инструментарий и эффективная реализация вычислений значений показателей для произвольного набора признаков и выявления зависимостей.
2. Анализировать данные оперативных систем напрямую невозможно или очень затруднительно: разрозненность данных, хранение их в форматах различных СУБД и в разных серверах корпоративной сети, сложность и запутанность структур хранения данных, избыточность детальной информации для анализа и принятия решений.
3. Сложные аналитические запросы к оперативной информации тормозят текущую работу компании, надолго блокируя таблицы и захватывая ресурсы сервера.

К хранилищам данных предъявляют следующие требования (тест FASMI – Fast Analysis of Shared Multidimensional Information):

1. Fast (Быстрый) – анализ должен производиться одинаково быстро по всем аспектам информации. Приемлемое время отклика – 5 секунд или менее.
2. Analysis (Анализ) – должна быть возможность осуществлять основные типы числового и статистического анализа.

3. Shared (Разделяемой) – много пользователей должны иметь доступ к данным, при этом необходимо контролировать доступ к конфиденциальной информации.
4. Multidimensional (Многомерной) – показатели должны вычисляться для произвольного набора классификационных признаков.
5. Information (Информации) – приложение должно иметь возможность обращаться к любой нужной информации, независимо от ее объема и места хранения.

В OLAP пользователь получает естественную, интуитивно понятную модель данных, представленную в виде многомерных ***кубов*** (Cubes). ***Измерениями*** (Dimensions) многомерной системы координат куба служат основные атрибуты (признаки) анализируемого бизнес-процесса. Таким образом, ***измерение*** – набор значений одного типа для идентификации некоторого свойства бизнес-процесса. Значения, "откладываемые" вдоль измерений, называются ***метками*** (members). Например, для анализа продаж измерениями могут быть товар, регион, тип покупателя. Метками измерения «Товар» будут все наименования товаров из конкретной предметной области.

Измерение может иметь иерархическую структуру – задавать некоторую классификацию значений. Измерение «Товар» может включать классификацию, например по категориям товаров. В этом случае измерение описывается не одним, а несколькими полями по числу уровней иерархии в классификации. Исключением – является время (Time dimension) для которого вводится набор из общеизвестных уровней (день, неделя, месяц, квартал, год).

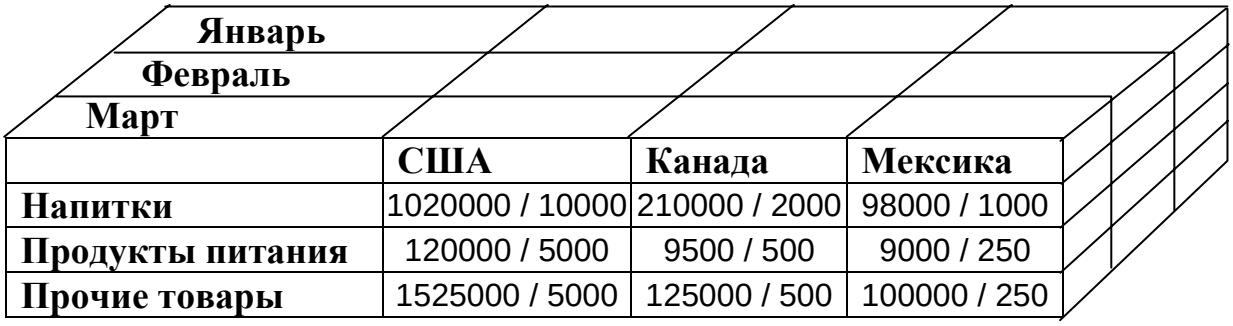
Значения всех измерений (координат) задают ***ячейку куба*** (cell) с которой связан набор ***показателей*** или ***мер*** (Measures), количественно характеризующих процесс. Это могут быть объемы продаж в штуках или в денежном выражении, остатки на складе, издержки и т. п.

В качестве показателей в трехмерном кубе, изображенном на рис. 20, использованы стоимости (числитель) и количество (знаменатель) продаж, а в качестве измерений – время продажи, товар и место продажи. Измерения представлены системами классификации меток: товары группируются по категориям, место продажи представлено магазинами и странами, а данные о времени совершения операций используются в общепринятой классификации: день, месяц, квартал, год.

Для многомерных данных вводят специальные операции манипулирования в кубах.

**Формирование "Среза".** Подмножество гиперкуба, получившееся в результате фиксации значения одного или более измерений, называется ***срезом*** (Slice).

Двумерное представление куба можно получить, “разрезав” его поперек одной или нескольких осей, выбрав соответствующие значения измерений. Если фиксируются значения всех измерений, кроме двух, – получается обычная двумерная таблица. В этом случае, в горизонтальной оси таблицы (заголовки столбцов) представлено одно измерение, в вертикальной (заголовки строк) – другое, а в ячейках таблицы – значения показателей. При этом набор показателей фактически рассматривается как одно из измерений: один показатель – одна двумерная таблица, второй – другая с теми же координатами (метками-заголовками строк и столбцов), но значениями другого показателя.




| | США | Канада | Мексика |
|---|---|---|---|
| **Напитки** | 1020000 / 10000 | 210000 / 2000 | 98000 / 1000 |
| **Продукты питания** | 120000 / 5000 | 9500 / 500 | 9000 / 250 |
| **Прочие товары** | 1525000 / 5000 | 125000 / 500 | 100000 / 250 |

Рис. 20 . Пример куба

На рис. 21 изображен двумерный срез куба для одного показателя – «стоимость» и двух “неразрезанных” измерений – «место продажи» и «время продажи». Зафиксировано значение категории «Напитки» для измерения «Товары».

| | США | Канада | Мексика |
|---|---|---|---|
| **Январь** | 1020000 | 210000 | 98000 |
| **Февраль** | 1200000 | 195000 | 94000 |
| **Март** | 1525000 | 225000 | 101000 |

Рис. 21. Двумерный срез куба для показателя «стоимость»

Метки используются как для “разрезания” куба, так и для ограничения (фильтрации) выбираемых данных – когда в измерении, остающемся “неразрезанным”, нас интересуют не все значения, а их подмножество, например три города из нескольких десятков. Значения меток отображаются в двумерном представлении куба как заголовки строк и столбцов.

**Операция агрегирования (Drill Up)** – переход от детализированных данных к агрегированным. Агрегирование выполняется при «проекции» (свертке) куба на меньшее число измерений. Агрегирование – проекция

куба на два измерения «Месяц» и «Страна» (свертка по измерению «Товар») приведет к получению куба аналогичного представленному на рис. 21. Разница будет в значениях показателей, которые для проекции будут вычисляться по всем товарам, а не только по категории «Напитки».

Агрегирование выполняется также при фиксации не детального уровня иерархии измерения. Пример на рис. 20 кроме среза содержит агрегирование: значение показателя суммируется для всех товаров, входящих в категорию «Напитки».

**Операция детализации (Drill Down)** – переход от агрегированных к более детализированным данным. Детализация появляется при добавлении ранее свернутого измерения – при этом уровень агрегирования понижается: каждая ячейка куба заменяется набором ячеек по количеству меток в добавляемом измерении, а каждое значение показателя рассчитывается для новой комбинации координат измерений. Детализация также может быть вызвана переходом на более детальный уровень измерения.

Работа пользователя с кубом обычно включает все перечисленные операции. Первоначально менеджера интересуют агрегированные значения показателей, скажем сумма всех продаж за последний период. Затем можно выполнить детализацию этого показателя по категориям товаров или товарам для определения имеющих максимальную сумму продаж. Возможно выполнить сечение по предыдущему периоду для анализа тенденций в изменения суммы продаж в целом и по отдельным категориям. Невозможно заранее предугадать направление анализа, тем не менее перечисленные операции позволяют реализовать любое из них.

## 9.2. Архитектура OLAP средств фирмы Microsoft

Многомерность в OLAP-приложениях может быть разделена на три уровня:

1. Многомерное представление данных – средства конечного пользователя, обеспечивающие многомерную визуализацию и манипулирование данными.
2. Многомерная обработка – средство (язык) формулирования многомерных запросов (традиционный реляционный язык SQL здесь оказывается непригодным) и процессор, умеющий обработать и выполнить такой запрос.
3. Многомерное хранение – средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов.

Конкретные OLAP-продукты, как правило, представляют собой либо средство многомерного представления данных – OLAP-клиент (например, сводные таблицы (Pivot Tables) в Excel фирмы Microsoft или ProClarity

фирмы Knosys), либо многомерную серверную СУБД – OLAP-сервер (например, Oracle Express Сервер или Microsoft OLAP Services).

Слой многомерной обработки обычно бывает встроен в OLAP-клиент и/или в OLAP-сервер, но может быть выделен в чистом виде, как, например, компонент Pivot Table Service фирмы Microsoft.

Основным компонентом аналитических служб является Analysis Server — сервис операционной системы Windows NT/2000. Этот сервер предназначен для создания OLAP-кубов на основе реляционных хранилищ данных, а также для предоставления доступа к ним из клиентских приложений.

Теоретически OLAP-куб, созданный с помощью аналитических служб Microsoft, может содержать все данные из таблицы фактов плюс агрегатные значения для тех групп записей из этой таблицы, которые соответствуют уровням иерархии измерений. При необходимости можно производить динамическое обновление куба, если в таблицу фактов были добавлены новые записи.

Аналитические службы сохраняют агрегатные данные только для простейших агрегатных функций (сумм, числа записей, максимальных и минимальных значений). Однако, в случае необходимости можно создавать так называемые вычисляемые члены (calculated members) для получения других типов агрегатных значений (средних, средневзвешенных, смещенных и несмещенных дисперсий и т.д.). При этом, помимо применения встроенных средств создания агрегатных данных, Analysis Server позволяет использовать для вычисления агрегатных данных функции VBA или Excel, а также создавать собственные средства.

Так, для создания нескольких кубов, имеющих одинаковые измерения, можно сгруппировать их в одну многомерную базу данных, а сами эти измерения поместить в библиотеку (library), сделав их коллективными. Можно также создавать измерения, принадлежащие только одному кубу (private dimensions).

Наконец, аналитические службы Microsoft позволяют создавать так называемые виртуальные кубы (virtual cubes), которые в определенной степени являются аналогами представлений (view) реляционных СУБД. Виртуальные кубы не содержат данных, но позволяют представить в виде единого куба данные из нескольких кубов, имеющих хотя бы одно общее коллективное измерение.

Decision Support Objects (DSO) — это набор библиотек, содержащих COM-объекты, позволяющие создавать и модифицировать многомерные базы данных и содержащиеся в них объекты (кубы, коллективные измерения и т.д.). Отметим, что Analysis Manager — приложение, использующее SQL DSO, — входит в состав аналитических служб.

Эти библиотеки можно использовать для разработки собственных приложений, в которых осуществляется создание или модификация многомерных баз данных, в том числе и для реализации действий, не предусмотренных в клиентских утилитах, входящих в состав аналитических служб.

Отметим, что SQL DSO можно использовать только для доступа к аналитическим службам Microsoft. Ни к каким другим OLAP-серверам с помощью этих библиотек обратиться нельзя.

Приложения, предназначенные для чтения OLAP-данных, при взаимодействии с аналитическими службами обязательно используют PivotTable Service — библиотеки, загружаемые в адресное пространство клиентского приложения. Эти библиотеки автоматически устанавливаются вместе с аналитическими службами (независимо от того, какая именно их часть установлена — клиентская или серверная), а также вместе с Microsoft Office 2000. В состав Microsoft SQL Server 2000 входит также инсталляционное приложение для установки PivotTable Service на компьютер, на котором не установлены ни аналитические службы, ни Microsoft Office.

PivotTable Service можно использовать в любой 32-разрядной версии Windows для просмотра серверных OLAP-кубов, а также для создания, модификации и чтения локальных OLAP-кубов, созданных в клиентском приложении, реализуя таким образом клиентскую OLAP-функциональность.

Для взаимодействия с PivotTable Service клиентское приложение может использовать OLE DB for OLAP — расширение универсального механизма доступа к данным OLE DB, позволяющее обращаться к многомерным данным, а также ADO MD — библиотеки, представляющие собой надстройку над OLE DB for OLAP и являющиеся COM-серверами для доступа к многомерным данным, удобными для применения в клиентских приложениях.

Отметим, что спецификация OLE DB for OLAP является открытой. Это означает, что можно создавать и другие OLAP-серверы, поддерживающие OLE DB for OLAP (либо разрабатывать OLE DB-провайдеры к уже имеющимся OLAP-средствам), а также создавать клиентские приложения, обращающиеся к любым таким источникам данных с помощью PivotTable Service, OLE DB for OLAP и ADO MD.

Analysis Manager представляет собой утилиту, входящую в состав аналитических служб и предназначенную для администраторов баз данных OLAP. Analysis Manager использует библиотеки SQL DSO для создания и модификации объектов многомерной базы данных и OLE DB для доступа к исходным реляционным хранилищам данных.

Из других клиентских приложений, не входящих в состав аналитических служб, но часто используемых для просмотра OLAP-кубов, следует назвать приложения Microsoft Office, в частности Microsoft Excel. С помо-

щью Excel можно обращаться к серверным OLAP-кубам, получая их двух- и трехмерные сечения или проекции на листах рабочих книг Excel в виде сводных таблиц, а также создавать локальные OLAP-кубы в виде файлов на основе реляционных данных, доступных с помощью OLE DB.

Кроме того, в состав Microsoft Office Web Components входит элемент управления ActiveX PivotTable List, позволяющий реализовать сходную функциональность как в обычном Windows-приложении, так и на HTML-странице, предназначенной для применения внутри корпоративной сети.

## 9.3. Создание многомерных баз данных в MS Analysis Server

Программа Analysis Manager позволяет выполнять все функции по созданию и администрированию многомерных баз данных. Для создания хранилища следует зарегистрировать OLAP-сервер в Analysis Manager, выбрав пункт Register Server из контекстного меню элемента Analysis Servers.

После этого можно создать многомерную базу данных, выбрав пункт New Database. Для многомерной базы определяется имя базы данных и ее описание.

Прежде чем создавать OLAP-кубы, необходимо описать источники исходных данных для них. Для описания источника данных, размещенного на MS SQL сервере нужно выбрать из контекстного меню элемента Data Sources пункт New Data Source и заполнить поля стандартной диалоговой панели «Поставщик данных». Если данные для хранилища будут извлекаться из БД MS SQL сервера, то в качестве провайдера данных следует выбрать OLE DB Provider for SQL Server, на вкладке «Подключение» выбрать сервер и базу данных, содержащую данные для построения куба.

Многомерный куб представлен в хранилище таблицей фактов (Fact Table), которая содержит значения показателей для комбинации меток измерений, обычно представленных кодами. Данные для каждого измерения берутся из соответствующего справочника (Dimension Table), в котором коду соответствует значение измерения. При этом возможны следующие конфигурации:

1. «Звезда» – таблица фактов непосредственно связана со справочником.
2. «Снежинка» – таблица фактов связана со справочником через другие таблицы (по такой схеме связан справочник «Поставщики» с таблицей фактов «Income» на рис. 23).

### Создание измерений

Измерения в MS Analysis Manager создаются мастером Dimension wizard (командой New Dimension | Wizard):

3. Выбор схемы измерения: «звезда» – Star Schema или «снежинка» – Snowflake.
4. Выбор таблиц – источника данных для создаваемого измерения .
5. Выбор типа измерения: «Time dimension» или «Standard dimension».
6. Выбор уровней иерархии.
7. Определение дополнительных свойств измерения. В изменяющихся измерениях (changing dimension) можно перемещать члены измерений между уровнями без перерасчета данных измерения.
8. Ввод имени измерения.

После выполнения перечисленных действий открывается окно редактирования измерения (см. рис. 22).

Основными свойствами измерения являются ключ (поле для указания метки) и значение (поле, содержащее метки). В качестве свойств (property) измерения можно указывать дополнительные поля. Для примера с рис. 22 такими свойствами являются розничная цена устройства (поле «Рознруб») и количество на складе (поле «Количество»).

Возможно создание несбалансированных иерархий – типа «родитель-потомок» (parent-child). Такие иерархии нередко основаны на таблицах, где первичный ключ является одновременно и внешним ключом (например, для каждого работника указывается код его начальника).

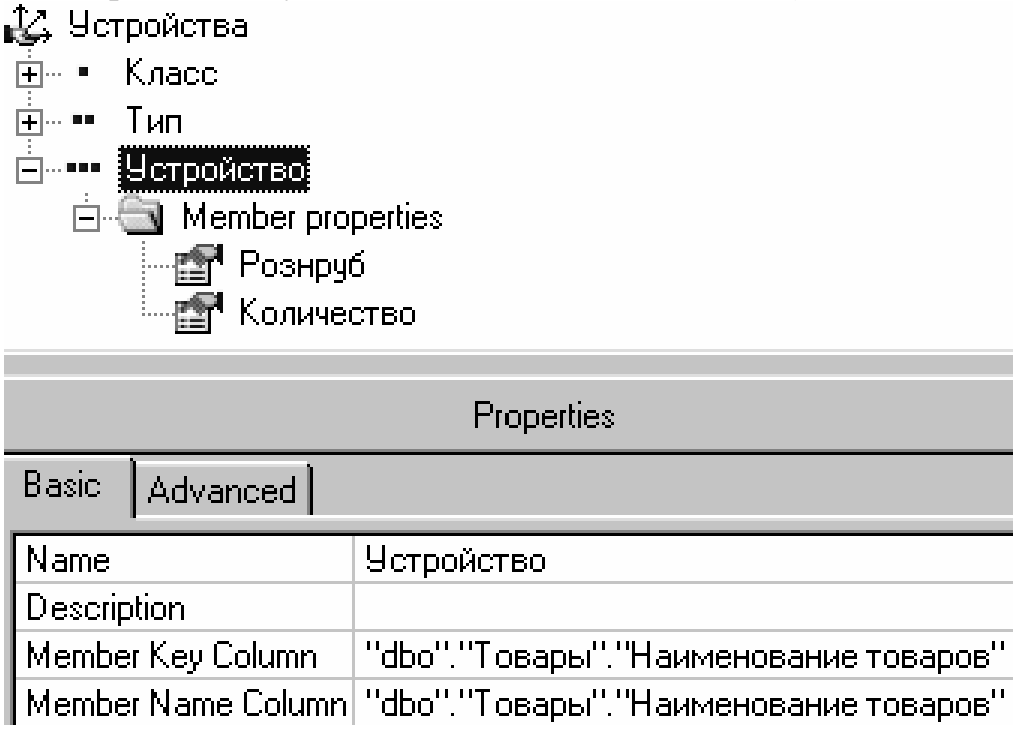



**Рис. 22. Форма редактирования измерения**

**Определение OLAP-кубов**

Куб заполняется значениями показателей на основании таблицы фактов, связанной с таблицами измерений. Мастер создания куба (Cube wizard) включает следующие шаги:

1. Выбор таблицы фактов.
2. Выбор полей – показателей.
3. Выбор измерений.
4. Ввод имени куба.

После выполнения перечисленных шагов открывается окно редактора куба, фрагмент которого приведен на рис. 23.

Для показателя определяется функция агрегирования, которая выполняется при выполнении агрегирования (Drill Up). Функция агрегирования может быть суммой, минимумом, максимумом, средним или другой функцией, вычисляющей значение по набору чисел. Дополнительно могут быть определены вычисляемые показатели в виде формул для вычисления значений (см. рис. 24). Вычисляемые показатели не хранятся в кубе, а вычисляются по мере необходимости.

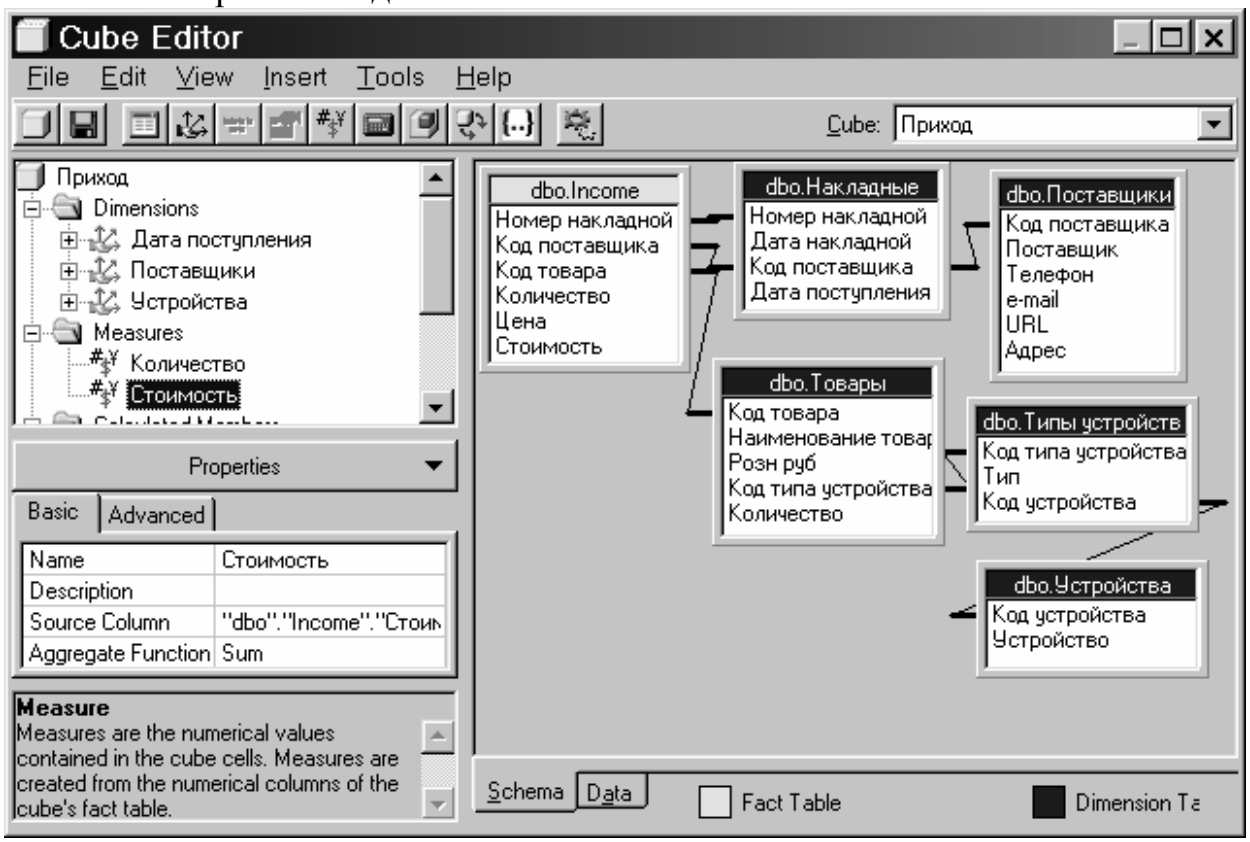



Рис. 23. Окно редактора куба

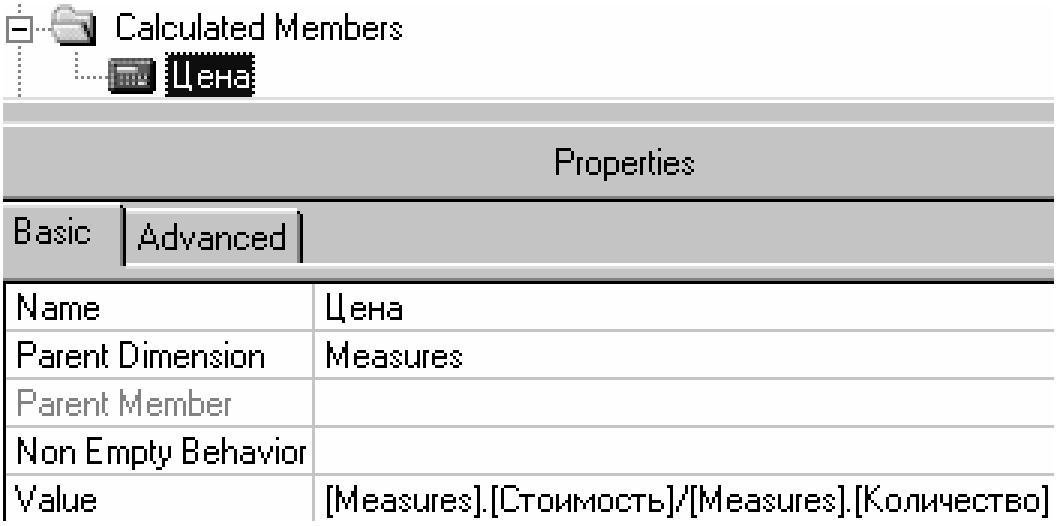



Рис. 24. Пример определения вычисляемого показателя

**Обработка и оптимизация базы данных OLAP**

Чтобы использовать базу данных OLAP, сначала необходимо обработать содержащиеся внутри нее кубы. Обработка куба— это его заполнение реальными данными из источника данных и вычисление обобщений (агрегатов). Но, прежде чем заполнить куб, нужно определить для него методы оптимизации. Analysis Server предоставляет два основных механизма оптимизации кубов внутри базы данных OLAP — обобщения (агрегаты) и разделы.

Обобщения (aggregations) — это итоговые данные, которые вычисляются службами анализа данных и сохраняются в базе данных вместе с данными. Это могут быть, например, итоговые данные о продажах за определенный период. Вычисления могут выполняться либо во время обработки куба, либо когда клиентское приложение запросит эти данные. Если создать обобщения, то вычисления будут выполняться во время обработки куба и итоговые данные будут сохранены в базе данных OLAP.

Когда клиентское приложение запросит данные из куба, сначала будут просмотрены все сохраненные обобщения, чтобы выяснить, существуют ли необходимые данные. Обнаружив запрошенные данные, сервер просто вернет их клиенту, не делая никаких дополнительных вычислений. Если же запрошенных данных нет ни в одном из сохраненных обобщении, то сервер должен вычислить данные «на лету». Поэтому, используя обобщения, можно существенно повысить производительность.

Однако обобщения не лишены недостатков. Они занимают значительное место на сервере и могут потребовать много времени для обработки. Поэтому, создавая для куба обобщения, нужно найти компромисс между производительностью и памятью.

Разделы (partitions) представляют собой физическое место хранения исходных и итоговых данных куба. При создании куба службы Analysis Server автоматически создают один раздел. После создания куба можно вернуться к этом процессу и создать новые разделы. Разделы используются для физического сегментирования данных из куба.

Преимущество разделов заключается в том, что у каждого из них может быть свой режим хранения и уникальный набор обобщений. Таким образом, один логический куб можно разделить на несколько источников физических данных, выбрав для каждого из них режим хранения и набор обобщений.

При первой обработке куба или при изменении параметров хранения (команда Design Storage) появится первое диалоговое окно мастера выбора режима хранения:

1. Режим хранения MOLAP предусматривает хранение исходных данных и обобщений в кубе и дает значительные преимущества в производительности по сравнению с ROLAP и HOLAP.
2. Метод хранения ROLAP использует существующую реляционную базу данных и оставляет исходные данные в их "родном" месте хранения. В реляционной базе данных создаются дополнительные служебные таблицы, которые хранят агрегированные значения. Данный метод проигрывает в производительности MOLAP, но позволяет экономить память за счет однократного хранения детальных значений показателей.
3. HOLAP – комбинирует возможности MOLAP и ROLAP – оставляет подробные данные в их "родном" источнике и в то же время дает возможность повысить производительность, поскольку итоговые данные помещаются в MOLAP.

Следующее диалоговое окно Set aggregation Options позволяет задать параметры создания обобщения:

1. Estimated Storage Reaches – службы анализа данных будут создавать обобщения до тех пор, пока заданный объем дискового пространства не будет использован полностью.
2. Performance Gain Reaches – службы анализа данных будут создавать обобщения до тех пор, пока производительность не увеличится на заданную величину в процентах (рекомендуется с указанием параметра 80%).
3. Until Clicks Stop – службы анализа данных будут создавать обобщения до тех пор, пока пользователь не щелкнет на кнопке Stop.

В следующем окне – Storage Design Wizard – можно выбрать один из двух параметров:

1. Process now – Analysis Server выполнит все вычисления и сохранит данные с помощью выбранного для куба режима хранения данных.
2. Save, but don't process now – сохранение плана создания обобщений, не проводя реальных вычислений и не сохраняя данные немедленно.

Analysis Server отобразит окно состояния, в котором будет виден ход выполнения процесса.

В Analysis Server предусмотрен инструмент, который называется *оптимизацией, ориентированной на использование* – выделите имя куба и щелкните на нем правой кнопкой мыши. Из контекстного меню выберите команду Usage-Based Optimization.

Еще один инструмент — это мастер анализа использования (Usage Analysis Wizard). Он позволяет администратору базы данных напечатать несколько графиков, которые покажут, как кубы базы данных OLAP используются клиентскими приложениями. Выделите имя куба и щелкните на нем правой кнопкой мыши. Из контекстного меню выберите команду Usage Analysis.

**Управление многомерными данными**

Кубы нужно обновлять по мере изменения исходных данных. В Analysis Server предусмотрено три основных механизма обновления многомерных данных внутри кубов: обработка куба, слияние разделов и запись данных клиентом.

Чтобы заполнить куб реальными данными, его нужно обработать. Аналогично, чтобы обновить данные в кубе, его также нужно обработать. Для обработки данных куба используется команда Process. На экране появится диалоговое окно Process a Cube. В этом окне предусмотрено три параметра обработки куба.

1. Incremental update (Инкрементальное обновление). К кубу добавляются только измененные данные. Все остальные данные куба останутся неизмененными, только обобщения будут пересчитаны с учетом добавленных данных.
2. Refresh data (Обновить данные). Куб заполняется заново.
3. Process (Обработать). Структура куба изменилась – будет выполняться построение структуры куба и заполнение его данными.

Службы анализа данных позволяют сделать куб доступным для записи. В действительности новые данные не записываются в сам куб или лежащий в его основе источник данных. Вместо этого в Analysis Server предусмотрена отдельная таблица для записей, в которой сохраняются все данные, записанные в куб. Чтобы сделать куб доступным для записи выберите команду Write-enable (Разрешить запись). В этом диалоговом окне введите

имя источника данных и таблицы, в которой будут сохраняться записанные данные. Причем пользователю будет казаться, что данные являются частью куба.

**Обеспечение безопасности данных OLAP**

Сервер Analysis Server отвечает за предоставление данных пользователям и, следовательно, нуждается в надежном механизме обеспечения безопасности данных. Для аутентификации пользователя Analysis Server применяет интегрированную систему безопасности Windows NT/2000.

После установки Analysis Server создается локальная группа OLAP Administrators. По умолчанию учетная запись пользователя, от имени которой устанавливалась Analysis Server, становится членом этой группы. Первичным механизмом управления доступом к данным OLAP является роль. Роли определяются на уровне базы данных. Для создания роли базы данных откройте окно Analysis Manager для папки Database Roles выберите команду Manage Roles. В окне Database Roles Manager можно увидеть все роли, созданные, для указанной базы данных. Добавить (изменить) роль можно кнопкой New (Edit).

На вкладке Membership можно редактировать список пользователей роли. Для добавления в роль групп или пользователей Windows NT/2000 щелкните на кнопке Add (Добавить). Появится стандартное диалоговое окно Windows NT/2000, в котором вам необходимо выбрать группы домена и пользователей, которые будут принадлежать новой роли.

Во вкладке Cubes укажите кубы, к которым роль должна получить доступ. На уровне куба можно только предоставить или отменить право доступа ко всему кубу. Однако, можно реализовать более детализированные уровни безопасности: на уровне измерений и ячеек куба.

Для установки параметров безопасности на уровне измерения активизируйте вкладку Dimensions. В этом диалоговом окне можно установить параметры безопасности:

1. Unrestricted (Неограниченный). Все пользователи, являющиеся членами роли, будут иметь неограниченный доступ для чтения всех уровней и всех членов измерения.
2. Fully Restricted (Полностью ограниченный). Установка этого правила безопасности означает, что все пользователи, являющиеся членами роли, не смогут просматривать никаких данных, относящихся к измерению.
3. Custom (Настраиваемый) – параметров безопасности измерения устанавливаются щелчком в поле Custom Settings.

Чтобы установить параметры безопасности на уровне ячейки, разверните куб, выделите папку Cube Roles и выберите команду Manage Roles. Появится диалоговое окно, в котором показаны все роли, которым предоставлен доступ к данному кубу. Для установки параметров на уровне ячейки щелкните на кнопке Cells. На уровне ячейки можно предоставить роли три уровня прав доступа:

1. Read (Чтение). Позволяет члену роли считывать значения данных ячейки в зависимости от установки критерия, определенного в PermissionsExpression.
2. Contingent Read (Условное чтение). Относится только к вычисляемым членам. С его помощью устанавливается доступ на чтение ячейки, возвращающей вычисляемые члены.
3. Read/Write (Чтение/запись). Позволяет записывать в ячейку данные. Его можно устанавливать только для куба, в который разрешена запись.

Чтобы ограничить доступ к отдельным ячейкам, необходимо определить для выбранного уровня прав доступа настраиваемое правило. Это можно сделать с помощью диалогового окна Edit a Cube Role. Из списка Cell Security Policy выберите значение Advanced. Для выбранного уровня прав доступа в столбце Rule установите значение Custom. Настраиваемое правило устанавливается с помощью определения выражения на языке MDX (см. п. 9.5).

## 9.4. Клиенты OLAP-данных

### Манипуляция OLAP-данными в Microsoft Excel

Для построения сводной таблицы по данным хранилища в Microsoft Excel можно воспользоваться мастером сводных таблиц (команда «Данные», «Сводная таблица»):

1. Выбор данных: для данных из хранилища выбрать вариант «во внешнем источнике данных».
2. Для получения данных следует нажать кнопку «Получить данные», выбрать вкладку «Кубы OLAP», определить источник многомерных данных.
3. Далее выполняются обычные действия по определению макета сводной таблицы.

Microsoft Excel позволяет создавать локальные OLAP-кубы, представляющие собой подмножества данных серверных OLAP-кубов. Локальные кубы хранятся в файлах с расширением *.cub.

### Публикация сводных таблиц на Web-страницах

Самый простой способ воспользоваться компонентом PivotTable List — сохранить сводную таблицу Microsoft Excel как Web-страницу. Для этого выберем в Microsoft Excel пункт меню «Файл», «Сохранить как вэб-страницу», в появившейся диалоговой панели выберем переключатель «Добавить интерактивность», нажмем кнопку «Опубликовать», в диалоговой панели выберем из выпадающего списка «Элементы «Лист1» и добавим «Работу со сводными таблицами».

Далее следует изменить заголовок, который появится на будущей Web-странице, и сохранить ее. Если открыть эту страницу в Microsoft Internet Explorer версии 4.01 или выше, мы увидим, что она содержит PivotTable List — элемент управления, предназначенный для просмотра OLAP-данных и сводных таблиц на Web-страницах.

Сразу же заметим, что этот элемент управления можно применять только в локальных сетях на компьютерах, для которых приобретена лицензия на Microsoft Office; другие способы его применения, например на Web-страницах, доступных в Интернете, запрещены лицензионным соглашением.

Пользователь, манипулирующий PivotTable List в браузере или в Windows-приложении, может, как и в сводной таблице Excel, перемещать данные в область строк, столбцов и страниц (в Microsoft Office Web Components приняты термины Row Area, Column Area и Filter Area) с диалоговой панели, напоминающей панель «Список полей сводной таблицы» из Excel.

Пользователь может также выполнять операцию детализации (drill-down), щелкая мышью на значках «+». Компонент PivotTable List позволяет сортировать и фильтровать данные. Во-первых, фильтрация данных может быть осуществлена с помощью отображения только выбранных членов измерений, которые могут быть отмечены в выпадающем списке, сходном с соответствующим списком Excel. Во-вторых, с помощью диалоговой панели «Команды и параметры» можно выбрать способы фильтрации и группировки данных.

Помимо этого пользователь может изменять атрибуты отображения данных — цвет и шрифт текста, цвет фона, выравнивание текста, отображение и т.д. Для этого достаточно поместить курсор на один из элементов данных, атрибуты которых нужно изменить (например, на наименование члена измерения, на ячейку с суммарными данными или с итоговыми значениями), и выбрать новые атрибуты отображения данных этого типа в той же диалоговой панели «Команды и параметры».

Помимо этого компонент PivotTable List позволяет на основе агрегатных данных вычислять доли или проценты общей суммы или суммы, соответствующей родительскому члену измерения (например, процент от годовой прибыли, полученный в данном квартале), — соответствующие опции можно найти в контекстных меню элементов данных.

Пользователю также доступен специально предназначенный для него файл справки (на русском языке, если используются Web-компоненты из комплекта поставки русской версии Microsoft Office XP). Однако пользователь не может изменить источник данных и отобразить на Web-странице другой OLAP-куб, поскольку право сделать это есть только у разработчика Web-страницы.

Отметим, что подобную Web-страницу можно создать и с помощью Microsoft FrontPage.

## 9.5. Язык запросов к многомерным данным MDX

Язык запросов к многомерным данным MDX (MultiDimensional eXpressions) был впервые введен в рамках спецификации OLE DB for OLAP для работы с многомерными кубами. Будучи открытым стандартом, MDX является основным инструментом программирования для Microsoft SQL Server 2000 Analysis Server.

Для выполнения запросов на языке MDX можно использовать утилиту MDX Sample Application, входящую в состав Microsoft SQL Server 2000 Analysis Server. При запуске этой утилиты появляется диалоговая панель Connect, в которой следует указать имя сервера и тип провайдера для связи с этим сервером (например, MSOLAP).

Верхняя панель утилиты MDX Sample Application (см. рис. 25) предназначена для задания MDX-запросов. Можно вводить MDX-команды непосредственно в панели запросов или конструировать запрос, перетаскивая измерения и меры куба в панель запросов. Помимо этого можно использовать примеры функций из панели Syntax Examples. Выполнить запрос можно одним из трех способов: выбрав команду Run в меню Query; нажав клавиши F5; нажав кнопку Run Query на панели инструментов. Результат выполнения запроса отображается в нижней части экрана

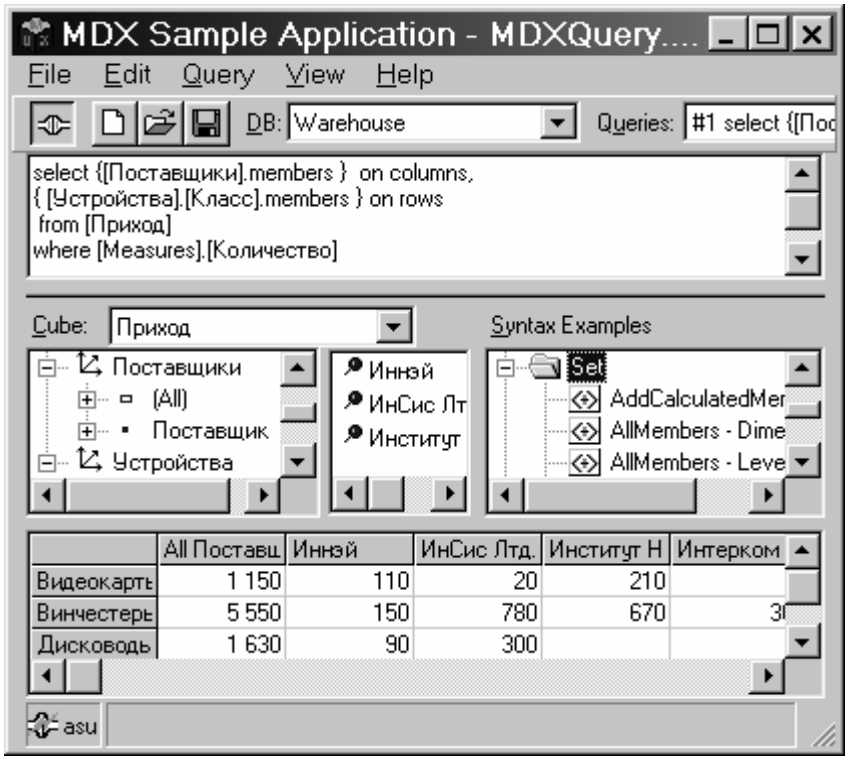



Рис. 25. Окно утилиты MDX Sample Application

MDX использует следующие понятия (для примеров ниже использованы данные куба с рис. 26):

– ***Компонент*** или ***метка*** (Member) — значение измерения на каждом уровне иерархии. Определяется перечислением всех узлов, расположенных на пути к вершине. Примеры:
*[Устройства].[All Устройства]*
*[Устройства].[All Устройства]. [Принтеры]*
*[Устройства].[All Устройства]. [Принтеры].[Лазерные]*
*[Устройства].[All Устройства]. [Принтеры].[Лазерные]. [Принтер лазерный А3 HP LJ-5100]*

– ***Кортеж*** (Tuple) — коллекция ячеек куба (срез), соответствующих некоторой комбинации значений измерений. Определяется как набор меток в круглых скобках. Размерность кортежа – множество измерений меток, вошедших в кортеж. Например, кортеж
*([Measures].[Стоимость],[Поставщики].[All Поставщики]. [ИнСис Лтд.], [Дата поступления].[All Дата поступления]. [2000].[Quarter 4])*
определяет измерения «Показатели», «Поставщики», «Дата поступления».

Рис. 26. Пример куба

– ***Множество*** (Set) — набор кортежей. Определяется как перечисление значений (кортежей) в фигурных скобках. Размерности всех кортежей множества должны совпадать.

Запрос на языке MDX представляет собой команду, которая выглядит следующим образом:

*SELECT [<ось> [, <ось>...]]*
*FROM [<куб>]*
*[WHERE [<срез>]]*

где: *<ось>* —описание измерения куба-результата в виде
*<множество> ON <имя оси>*,
*<имя оси>* может быть следующим:

– *COLUMNS* – колонки,
– *ROWS* – строки,
– *PAGES* – страницы,
– *SECTIONS* – разделы,
– *CHAPTERS* – главы,
– *AXIS(<номер>)* – ось с указанным номером (оси нумеруются с нуля).

Срез куба задается кортежем. Измерение не может употребляться одновременно для задания двух осей или оси и среза. Для указания множества меток иерархии (или ее уровня) применяется метод members. Кортеж заключается в круглые скобки, в то время как множество — в фигурные. Порядок перечисления измерений и мер в кортеже не имеет значения.

Результатом запроса всегда являются значения одного или нескольких показателей, соответствующих значениям измерений (см. рис. 27). Измерения располагаются по одной или нескольким осям, определенным в запросе. Например, выборка с рис. 27 создается следующим запросом

```
select { [Дата поступления].[Year] .members }  on columns,
{  [Поставщики].members } on rows
 from [Приход]
where [Measures].[Стоимость]
```

| | 2000 | 2001 | 2002 | 2003 | 2004 |
|---|---|---|---|---|---|
| All Поставщики | 34 335 151,09 | 102 938 249,85 | 84 287 083,70 | 91 299 288,86 | 22 678 875,76 |
| Иннэй | | | | 17 629 444,33 | 6 810 703,55 |
| ИнСис Лтд. | 10 339 820,90 | 7 027 899,11 | 16 879 648,67 | 7 468 279,40 | |
| Институт Новых Технологий Образован | | 6 153 567,90 | 14 018 973,94 | 16 219 473,67 | |
| Интерком (Intercom) | | | 7 027 212,30 | | |
| ИнтерКоннект | | 6 210 211,94 | | | |
| Интерлинк Системс (Interlink Systems) | 11 799 940,87 | 3 745 585,36 | | 8 656 252,40 | |
| Инфарс | | 5 714 527,87 | 6 818 875,83 | | |
| Инфоком | | 35 637 114,01 | 5 097 507,33 | | |
| Информат-Сервис | | | 8 092 414,88 | | |
| Инэкс Компьютерные Системы (INECS) | | 11 835 538,84 | | 7 921 152,13 | 4 749 221,50 |
| Исследовательская Лаборатория | | | 10 753 409,03 | 7 108 926,84 | |
| Кварта Технологии (Quarta Technologies | | | 7 800 157,43 | 22 655 934,22 | |
| Конструктор НТЦ | 8 852 275,62 | 21 632 976,37 | 7 798 884,31 | 3 639 825,87 | 7 269 470,67 |
| Корона (Korona) | 3 343 113,71 | 4 980 828,44 | | | 3 849 480,05 |

Рис. 27. Результат запроса MDX

В MDX Sample Application используются лишь оси столбцов и строк

Для определения осей можно использовать следующие функции:

*Crossjoin(<Множество>, <Множество>)* позволяет комбинировать несколько измерений в одно измерение куба-результата

Запрос

*SELECT [Measures].AllMembers on columns,*
*CROSSJOIN([Устройства].[Класс].members , [Поставщики].[Поставщик].members) on rows from [Приход]*

формирует двухуровневую структуру (классы устройств, поставщики) заголовков столбцов.

*Filter((<Множество>, <Условия>)* позволяет выполнять фильтрацию меток измерения (оси).

*SELECT [Measures].AllMembers on columns,*
*Filter ({[Поставщики].[Поставщик] .members} ,*
*[Measures].[Стоимость]>10000000) on rows*
*from [Приход]*

*Order(<Множество>, <Выражение> [, ASC | DESC | BASC | BDESC])* сортирует метки измерения с сохранением иерархии *ASC | DESC* или без нее *BASC | BDESC*.

В примере

*SELECT [Measures].AllMembers on columns,*
*ORDER([Устройства] .members, [Устройства].CurrentMember.Name, basc) on rows from [Приход]*

используется сортировка по наименованиям меток без учета иерархии измерения. Ниже приведен запрос с сортировкой поставщиков по убыванию стоимости

*SELECT [Measures].AllMembers on columns,*
*ORDER( [Поставщики].[Поставщик].members,*
*[Measures].[Стоимость], desc) on rows FROM [Приход]*

*TopCount (<Множество>, n, <Выражение>)* выделяет из множества первые *n* компонент с наибольшими значениями выражения, например

*SELECT [Measures].AllMembers on columns,*
*TopCount ([Поставщики].[Поставщик].members ,3, [Measures]. [Стоимость]) on rows FROM [Приход]*

Аналогичными функциями являются TopPercent, BottomCount, BottomPercent. Запрос

*TopPercent([Поставщики].[Поставщик].members ,60, [Measures].[Стоимость])*
выбирает поставщиков, суммарная стоимость продаж которых не меньше 60%

Множество может быть задано интервалом
*<1-ая метка> : <последняя метка>*, например

*SELECT [Measures].AllMembers on columns,*
*[Дата поступления].[All Дата поступления].[2003].[Quarter 3].[August]: [Дата поступления].[All Дата поступления].[2003]. [Quarte 4].[December] on rows FROM [Приход]*

Для указания иерархии измерения используются следующие методы и функции:

*.children, .FirstChild, . LastChild* – «потомки» метки иерархии (перечисленные методы указываются для определенной метки, например *[Устройства].[All Устройства].[Компьютеры].children*);

*.Parent* – «родительская» метка

*.Siblings , .FirstSibling, .LastSibling* – «соседи» по уровню;

*Ascendants(<метка>)* – функция возвращает родительскую метку;

*Descendants(<метка> [, «Level»[, «Desc_flags»]])* – функция, возвращающая метки расположенные ниже по уровню иерархии метки – аргумента, например функция *Descendants( [Устройства].[All Устройства].[Компьютеры]. ["Драйв" Молния (Duron 1400-1800)] , [Устройства].[Устройство] )* вернет все устройства указанного типа.

Определение *NON EMPTY <ось>* исключает строки, у которых все клетки пустые (например, не было продаж).

Во всех выражениях языка MDX значения указываются при помощи кортежей. В примере

*select { [Measures] .Allmembers} on columns,*
*FILTER( [Устройства].[Устройство].Members ,*
*([Measures].[Количество],[Дата поступления].*
*[All Дата поступления].[2002]) >*
*([Measures].[Количество],[Дата поступления].[All Дата*
*поступления].[2003])) on rows from [Приход]*
*WHERE ( [Дата поступления].[All Дата поступления].[2003] )*

кортеж используется внутри функции фильтрации, чтобы для каждой метки измерения «Устройство» вычислить показатель «Количество» за определенный год.

Свойства измерений можно использовать в качестве меток

*<Измерение> [DIMENSION] PROPERTIES <поле> [,<поле>...]*

Запрос

*select { [Measures].AllMembers} on columns,*
*[Устройства].[Устройство].members DIMENSION PROPERTIES [Устройства].[Устройство].Name,*
*[Устройства].[Устройство].Количество on rows from [Приход]*

для измерения «Устройство» предусматривает вывод полей «Название устройства» и «Количество».

В запросах можно определять вычисления при помощи следующей конструкции

*WITH <формула> [, <формула>] <Запрос>*

Наиболее часто применяется формулы вычисления уровней измерений

*MEMBER <имя уровня> AS '<выражение>'*
*[,SOLVE_ORDER = <номер итерации>]*
*[,<свойство>=<значение>...]*

< *имя уровня* > – полностью квалифицированное имя с указанием измерения и уровня иерархии, к которому будет отнесено вычисляемое значение.

< *выражение* > – выражение, вычисляющее значение,

< *номер итерации* > – номер, определяющий порядок вычисления, если вычисляемые значения используют другие вычисляемые значения.

В качестве свойств ячейки можно указывать шрифты и другие особенности форматирования, например, FORMAT_STRING = '# ##0' означает вывод чисел с отбрасыванием дробной части и разделением групп разрядов.

Следующий запрос для каждого периода времени вычисляется суммарная стоимость продаж в сопоставлении с предыдущим периодом.

```
WITH MEMBER [Measures].[Стоимость товаров за прошлый период]
  AS '([Measures].[Стоимость],  [Дата поступления].PrevMember)'
  MEMBER [Measures].[Увеличение стоимости товаров]
  AS '[Measures].[Стоимость]- [Measures].[Стоимость товаров за прошлый период]'
select {  [Дата поступления].[Year] .Members} ON ROWS,
{[Measures].[Стоимость] ,
[Measures].[Стоимость товаров за прошлый период],
[Measures].[Увеличение стоимости товаров] } ON COLUMNS
from [Приход]
```

Можно вычислять не только новые уровни измерений, но и множества

```
WITH SET <set_name> AS '<set>'
```

например:

```
WITH
 SET [Отсортированные устройства] AS
   'ORDER( {[Устройства].[Устройство] .members} ,
              [Measures].[Стоимость], desc)'
 MEMBER [Measures].[All] AS
   '  Sum( [Устройства].[Наименование Товаров].Members ,
        [Measures].[Стоимость]) '
 MEMBER [Measures].[Процент] AS
   '  [Measures].[Стоимость] / [Measures].[All] ',
     FORMAT_STRING = '# ###.0 %'
select { [Measures].[Стоимость],
 [Measures].[Количество],[Measures].[Цена],  [Measures].[All],
 [Measures].[Процент]}  on columns,
[Отсортированные устройства]  on rows
 from [Приход]
```

MDX-запросы можно использовать для извлечения данных из кубов аналогично SQL-запросам. Такое использование обладает большими аналитическими возможностями и высоким быстродействием.

**Вопросы**

1. Назначение хранилищ данных. Отличие хранилищ данных от баз данных.
2. Причины построения и использования хранилищ данных.
3. Требования, предъявляемые к хранилищам данных.
4. Опишите структуру данных хранилища.

5. Дайте определение понятий «куб», «измерение», «показатель», «ячейка куба», «метка измерения».
6. Перечислите и опишите основные операции в кубе.
7. Опишите технологию создания куба с помощью MS Analysis Manager.
8. Что содержит таблица фактов?
9. Для чего используются обобщения (агрегаты)?
10. Перечислите и опишите режимы хранения данных в кубе.
11. Как обновляется содержание куба?
12. Опишите систему безопасности MS Analysis Server.
13. Какие средства можно использовать для доступа к данным хранилища?
14. Для чего предназначен язык MDX? Каковы его основные возможности?
15. Как выглядит запрос на языке MDX? Каковы его основные компоненты?
16. Описание кортежей, множеств, меток в языке MDX.
17. Опишите применение функций сортировки, фильтрации, комбинирования измерений в языке MDX.
18. Опишите применение функций работы с иерархическими структурами в языке MDX.
19. Укажите возможности программирования вычислений в языке MDX.

## Использованная литература


1. Martin, J. Information Engineering: a trilogy.- Englewood Cliff: Prentice-Hall, 1989-1990.- 3 vol.
2. Отраслевой стандарт “Информационные технологии в высшей школе. Термины и определения. ОСТ ВШ 01.002-95”.- М., 1995 г., 24 с.
3. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник.- М.: Финансы и статистика, 2004.- 512с.:ил.
4. Дейт К.Дж. Введение в системы баз данных: Изд. 6-е.-М.: Вильямс, 2000.- 846с.
5. Оутей М., Конте П., Эдир О. SQL Server 2000. SQL Server 2000.- СПб.; Киев: Питер: Издат. группа BHV, 2002.-990с.
6. Голосов А.О. Аномалии в реляционных базах данных.- Системы управления базами данных, № 03, 1996.- http://www.osp.ru/dbms/1996/03/23.htm

Учебное издание

Братищенко Владимир Владимирович

БАЗЫ ДАННЫХ

Учебное пособие

Издается в авторской редакции

Издается в авторской редакции

ИД № 06318 от 26.11.01.
Подписано в печать 07.06.06. Формат 60x90 1/16. Бумага офсетная. Печать трафаретная. Усл. печ. л. 6,1. Уч.-изд. л. 5,4. Тираж 100 экз. Заказ.

Издательство Байкальского государственного университета
экономики и права.
664003, Иркутск, ул. Ленина, 11.
Отпечатано в ИПО БГУЭП