ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РЫБИНСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. А. СОЛОВЬЕВА»
(РГАТУ имени П.А. Соловьева)

ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА ПОДГОТОВКИ АСПИРАНТОВ

направление подготовки 09.06.01 Информатика и вычислительная техника

профиль подготовки 05.13.06 Автоматизация и управление технологическими процессами и производствами (в промышленности)

КОНСПЕКТ ЛЕКЦИЙ

по дисциплине

Информационное обеспечение процессов автоматизации (СУБД)

Разработал: д.т.н. Юдин А. В.

Рыбинск, 2014 г.

Изучаются специализированные информационные технологии в сфере автоматизации процессов и производств. Также в ходе изучения дисциплины аспиранты получают знания о информационном обеспечение процессов автоматизации, используемых в настоящее структурах данных и языках описания и манипулирования данными.

Содержание

Лекция 1. Понятие данных, системы данных. Объекты данных. Атрибуты объектов.

Хранение информации – одна из важнейших функций компьютера. Одним из распространенных средств такого хранения являются базы данных.

База данных (БД) – совокупность взаимосвязанных, хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляется централизованно с помощью специального программного инструментария – системы управления базами данных.

Система управления базами данных (СУБД) – это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации.

Концептуальная модель БД описывает сущности, их свойства и связи между ними; не зависит от конкретной СУБД.

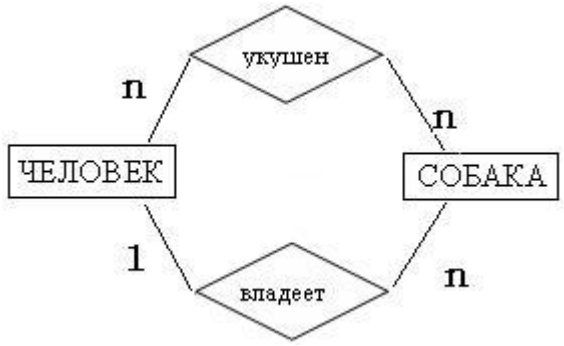
Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна. В диаграммах сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности – это имя типа, а не некоторого конкретного экземпляра этого типа. Примеры сущностей: ФАКУЛЬТЕТ, ГРУППА, СТУДЕНТ. Каждый экземпляр сущности (объект) должен быть отличим от любого другого экземпляра той же сущности.

Пример экземпляров сущности ФАКУЛЬТЕТ: ПС, ФМ, АТ и т.п., сущности СТУДЕНТ: Иванов А.П., Петрова Н.Н. и т.п.

Связь (relationship) – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Связь может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Возможны связи на основе отношений:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

Связь «содержит»: ГРУППА содержит много СТУДЕНТОВ. Каждый СТУДЕНТ входит только в одну ГРУППУ.



Связь «укушен»: СОБАКА может укусить много ЧЕЛОВЕК, ЧЕЛОВЕК может быть укушен многими СОБАКАМИ.

Связь «владеет»: ЧЕЛОВЕК может владеть многими СОБАКАМИ. У СОБАКИ может быть только один хозяин.

Связь "один к одному" встречается редко. Например, у нас есть таблица с информацией о всех сотрудниках и таблица с информацией о всех торговых агентах, которые являются сотрудниками нашего предприятия. Записи в таких таблицах могут быть связаны отношением "один к одному".

Свойства сущностей

Сущности имеют свойства, которые называются *атрибутами* (attribute).

Например, атрибуты:

- сущности ФАКУЛЬТЕТ:
  - название;
  - год создания;
- сущности ГРУППА:
  - номер;
- сущности СТУДЕНТ:
  - фамилия;
  - имя;
  - отчество;
  - номер студенческого билета;
  - номер паспорта;
  - год рождения;
  - месяц рождения;
  - день рождения.

Любой атрибут принимает значения из некоторого множества допустимых значений, называемого **доменом атрибута.**

Например:

- домен атрибута «год создания»: целые положительные числа;
- домен атрибута «имя»: строка, не содержащая пробелов;
- домен атрибута «год рождения»: целые положительные числа;
- домен атрибута «месяц рождения»: январь, февраль, март … декабрь;
- домен атрибута «день рождения»: целые числа от 1 до 31.

Ключ сущности

Ключ сущности (entity key), **первичный ключ** – это атрибут (или множество атрибутов) уникальным образом идентифицирующих экземпляр сущности (объект).

Например: ключ сущности СТУДЕНТ – номер студенческого билета, ключ ФАКУЛЬТЕТА – название. Если ключ состоит из одного атрибута, его называют **простым ключом**. Если ключ сущности состоит из нескольких атрибутов, его называют **составным ключом.**

Например, для сущности ДОМ с атрибутами «улица», «этажность», «год постройки», «номер дома», первичным ключом будет «улица»+ «номер дома».

Основные функции СУБД

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;

Транзакция – в информатике – совокупность операций над данными, которая, с точки зрения обработки данных, либо выполняется полностью, либо совсем не выполняется. **Транзакция** – в информационных системах – последовательность логически связанных действий, переводящих информационную систему из одного состояния в другое. Транзакция либо должна завершиться полностью, либо система должна быть возвращена в исходное состояние.

- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

Виды моделей данных

Организация данных рассматривается с позиций той или иной модели данных. Модель данных является ядром любой базы данных. С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними.

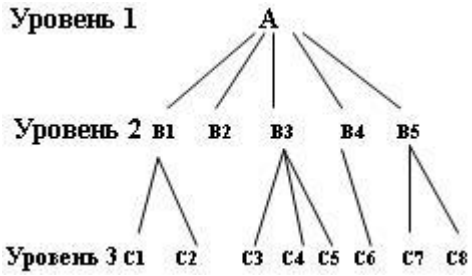
Модель данных – совокупность структур данных, ограничений целостности и операций манипулирования данными. Модели используются для представления данных в информационных системах.

Различают три типа моделей данных, которые имеют множества допустимых информационных конструкций:

- *иерархическая;*
- *сетевая;*
- *реляционная.*

Иерархическая модель данных

Иерархическая структура представляет совокупность элементов, связанных между собой по определенным правилам. Объекты, связанные иерархическими отношениями, образуют ориентированный граф (перевернутое дерево), вид которого представлен на рисунке:



Основные понятия иерархической структуры

Это – узел, уровень и связь.

Узел – это совокупность атрибутов данных, описывающих некоторый объект. На схеме иерархического дерева узлы представляются вершинами графа. Каждый узел на более низком уровне связан только с одним узлом, находящимся на более высоком уровне.

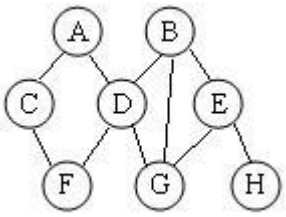
Иерархическое дерево имеет только одну вершину (корень дерева), не подчиненную никакой другой вершине и находящуюся на самом верхнем (первом) **уровне**. Зависимые (подчиненные) узлы находятся на втором, третьем и т.д. уровнях. К каждой записи базы данных существует только один (иерархический) путь от корневой записи. Например, как видно из рисунке, для записи С4 путь проходит через записи В3 к А.

Пример иерархической структуры:

Сетевая модель данных

В сетевой структуре при тех же основных понятиях (уровень, узел, связь) каждый элемент может быть связан с любым другим элементом.

На рисунке изображена сетевая структура базы данных в виде графа.



Пример сетевой структуры:

Примером сложной сетевой структуры может служить структура базы данных, содержащей сведения о студентах, участвующих в научно-исследовательских работах (НИРС). Возможно участие одного студента в нескольких НИРС, а также участие нескольких студентов в разработке одной НИРС. Графическое изображение описанной в примере сетевой структуры состоит только из двух типов записей.

Реляционная модель данных

Понятие *реляционный* (англ. *relation* – отношение) связано с разработками известного американского специалиста в области систем баз данных Е.Кодда.

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая *реляционная таблица* представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы – один элемент данных;
- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;
- каждый столбец имеет уникальное имя (заголовки столбцов являются названиями полей в записях);
- одинаковые строки в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным.

Лекция 2. Понятие записи данных. Файлы данных. Базы данных.

Любой из нас, начиная с раннего детства, многократно сталкивался с «базами данных». Это — всевозможные справочники (например, телефонный), энциклопедии и т. п. Записная книжка — это тоже «база данных», которая есть у каждого из нас.

Базы данных представляют собой информационные модели, содержащие данные об объектах и их свойствах. Базы данных хранят информацию о группах объектов с одинаковым набором свойств.

Например, база данных «Записная книжка» хранит информацию о людях, каждый из которых имеет фамилию, имя, телефон и так далее. Библиотечный каталог хранит информацию о книгах, каждая из которых имеет название, автора, год издания и так далее.

Информация в базах данных хранится в упорядоченном виде. Так, в записной книжке все записи упорядочены по - алфавиту, а в библиотечном каталоге - либо по алфавиту – алфавитный каталог), либо по области знания (предметный каталог).

Существует несколько различных структур информационных моделей и соответственно различных типов баз данных: табличная, сетевая, иерархическая (см. модели).

Иерархические базы данных

Иерархические базы данных графически могут быть представлены как перевернутое дерево, состоящее из объектов различных уровней. Верхний уровень (***корень дерева***) занимает один объект, второй — объекты второго уровня и так далее.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении ***предка*** (объект, более близкий к корню) к ***потомку*** (объект более низкого уровня), при этом объект-предок может не иметь потомков или иметь их несколько, тогда как объект-потомок обязательно имеет только одного предка. Объекты, имеющие общего предка, называются ***близнецами.***

Например: иерархической базой данных является *Каталог папок Windows,* с которым можно работать, запустив Проводник. Верхний уровень занимает папка *Рабочий стол.* На втором уровне находятся папки *Мой компьютер, Мои документы, Сетевое окружение* и *Корзина,* которые являются потомками папки*Рабочий стол,* а между собой является близнецами. В свою очередь, папка *Мой компьютер* является предком по отношению к папкам третьего уровня - папкам дисков *(Диск 3,5(A:), (C:), (D:), (E:), (F:))* и системным папкам *(Принтеры, Панель управления* и др.)

Сетевые базы данных

Сетевая база данных является обобщением иерархической за счет допущения объектов, имеющих более одного предка. Вообще, на связи между объектами в сетевых моделях не накладывается никаких ограничений.

Сетевой базой данных фактически является *Всемирная паутина* глобальной компьютерной сети Интернет. Гиперссылки связывают между собой сотни миллионов документов в единую распределенную сетевую базу данных.

Табличные базы данных

Табличная база данных содержит перечень объектов одного типа, то есть объектов, имеющих одинаковый набор свойств. Такую базу данных удобно представлять в виде двумерной таблицы: в каждой ее строке последовательно размещаются значения свойств одного из объектов; каждое значение свойства — в своем столбце, озаглавленном именем свойства.

Рассмотрим, например, базу данных: **Телефонный справочник**

№	Фамилия	Адрес	Телефон
1	Иванов В.В.	Серова, 5 12	4325345
2	Петров И.И.	Седова, 3-21	3454365
3	Сидоров С.С.	Мира, 33-17	3454354

Столбцы такой таблицы называют полями; каждое поле характеризуется своим именем (именем соответствующего свойства) и типом данных, представляющих значения данного свойства.

Строки таблицы являются записями об объекте; эти записи разбиты на поля столбцами таблицы, поэтому каждая запись представляет собой набор значений, содержащихся в полях.

Каждая таблица должна содержать, по крайней мере, одно ключевое поле, содержимое которого уникально для каждой записи в этой таблице. Ключевое поле позволяет однозначно идентифицировать каждую запись в таблице.

В качестве ключевого моля чаще всего используют поле, содержащее тип данных ***счетчик.*** Однако иногда удобнее в качестве ключевого поля таблицы использовать другие поля: код товара, инвентарный номер и т. п.

Телефонный справочник

Имена полей
Запись
Запись
Запись

№	Фамилия	Адрес	Телефон
1	Иванов В.В.	Серова, 5 12	4325345
2	Петров И.И.	Седова, 3-21	3454365
3	Сидоров С.С.	Мира, 33-17	3454354

Ключевое поле ***Поле*** ***Поле*** ***Поле***

<u>Тип поля определяется</u> типом данных, которые оно содержит. Поля могут содержать данные следующих основных типов:

- *счетчик* — целые числа, которые задаются автоматически при вводе записей. Эти числа не могут быть изменены пользователем;
- *текстовый* — тексты, содержащие до 255 символов;
- *числовой* — числа;
- *дата/время* — дата или время;
- *денежный* — числа в денежном формате;
- *логический* — значения *Истина* (Да) или *Ложь* (Нет);
- *поле объекта OLE* - изображение или рисунок
- *гиперссылка* — ссылки на информационный ресурс в Интернете (например, Web-сайт).

Поле каждого типа имеет свой набор свойств. **<u>Наиболее важными свойствами полей</u>** являются:

- *размер поля* - определяет максимальную длину текстового или числового поля;
- *формат поля* - устанавливает формат данных;
- *обязательное поле* - указывает на то, что данное поле обязательно надо заполнить

Лекция 3. Модели данных. Реляционная модель данных.

основе реляционных систем лежит *реляционная модель данных*. Принципы реляционной модели были заложены в 1969–1970 гг. американским ученым Е. Ф. Коддом (E. F. Codd), в то время работавшим в корпорации IBM. Будучи математиком по образованию, он привнес в область управления базами данных строгие математические принципы и точность, которых не хватало ранним системам. Хотя реляционный подход утвердился не сразу, можно отметить, что почти все созданные с конца 70-х гг. продукты баз данных основаны именно на реляционном подходе. Подавляющее большинство научных исследований в области баз данных в течение последних 35 лет также проводилось именно в этом направлении.

Рассматривая и постепенно уточняя основные понятия реляционной модели, будем иметь в виду три компоненты модели данных:

- структуры данных,
- операции, которые можно выполнять над данными, и
- ограничения, связанные с обеспечением целостности данных.

Основной структурой данных в реляционной модели являются *таблицы*, называемые в реляционной теории *отношениями*. Собственно от термина *отношение* (по-английски relation) и произошло само название модели – *реляционная*. На рис. 6.1 приведен пример такой таблицы-отношения и пояснение основных терминов реляционной модели – *кортеж*,*кардинальное число*, *атрибут*, *степень*, *домен*, *первичный ключ*.

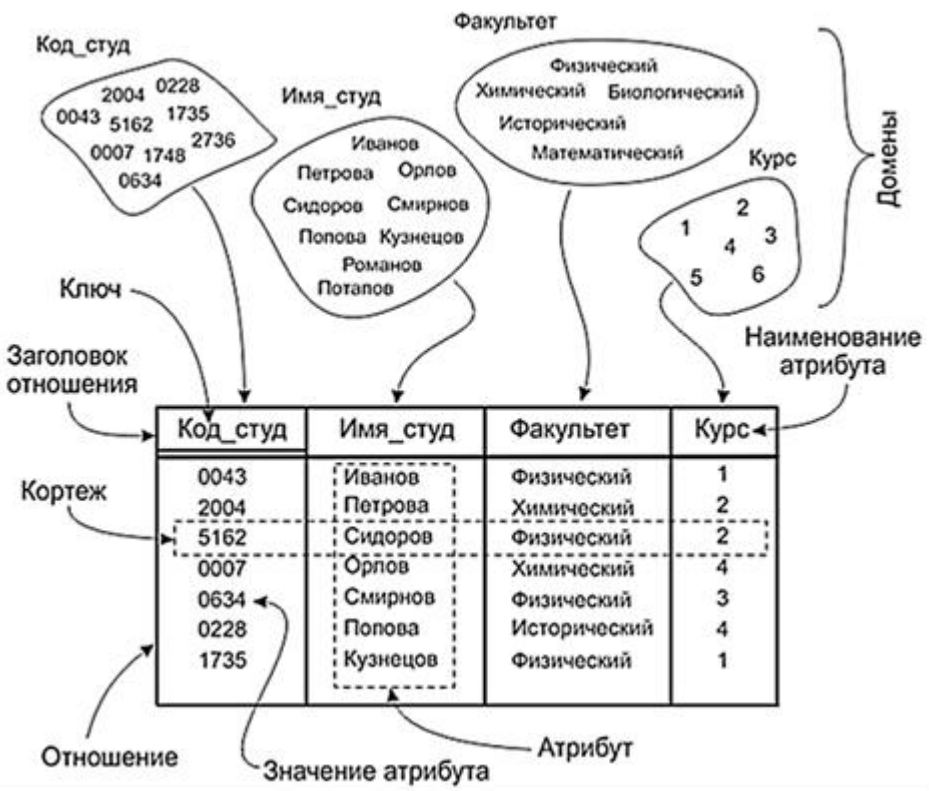


Рис. 6.1. Отношение и его компоненты

Коротко, пока не очень строго, основные понятия реляционной модели можно определить следующим образом.

- *Отношение* это таблица, подобная приведенной на рис. 6.1, состоящая из строк и столбцов и имеющая вверху строку, называемую *заголовок отношения*.
- Строки таблицы-отношения называются *кортежами* (tuple), а столбцы *атрибутами* (attribute).
- Количество кортежей в отношении называется *кардинальным числом* отношения, а количество атрибутов называется *степенью* отношения.
- Каждый атрибут в отношении имеет *наименование*, которое указывается в заголовочной части отношения.
- *Ключ* отношения – это атрибут или набор атрибутов отношения такие, что в любой момент времени в отношении не существует строк, для которых значение или комбинация значений ключевых атрибутов являются одинаковыми. Ключ, таким образом, является уникальным идентификатором кортежей отношения (на рис. 6.1 ключевой атрибут выделен жирным шрифтом).
- *Домен* отношения – это множество значений, из которого могут браться значения конкретного атрибута. То есть конкретный набор значений атрибута в любой момент времени должен быть подмножеством множества значений домена, на котором определен этот атрибут. Значения атрибута, которые отсутствуют в множестве, задаваемом доменом, являются недопустимыми.

Понятие домена является важным для реляционной модели. Домен фактически задает ограничения, которым должны удовлетворять значения соответствующего атрибута.

Как уже отмечалось, приведенные выше определения не являются строгими. Такие термины как *таблица*, *строка*, *столбец*, строго говоря, не являются полностью эквивалентными используемым в реляционной модели *математическим* понятиям *отношение*, *кортеж*, *атрибут* соответственно.
Однако на практике их часто используют именно как синонимы, что, в общем, допустимо, если при этом понимать, какой действительный смысл вкладывается в эти термины.

Лекция 4. Сетевая модель данных. Иерархическая модель данных.

На разработку этого стандарта большое влияние оказал американский ученый Ч.Бахман. Основные принципы сетевой модели данных были разработны в середине 60-х годов, эталонный вариант сетевой модели данных описан в отчетах рабочей группы по языкам баз данных (COnference on DAta SYstem Languages) CODASYL (1971 г.).

Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых

отношений. Связь между между записью-владельцем и записью-членом также имеет вид **1:N**.

Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом *более чем одного* группового отношения. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: экземпляр записи не может быть членом двух экземпляров групповых отношений одного типа (т.е. сотрудник из примера в п.3.1, например, не может работать в двух отделах).

Иерархическая структура из п.3.1. преобразовывается в сетевую следующим образом (см. рис. 3.2):

- древья (a) и (b), показанные на рис. 3.1, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
- для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, см. рис. 3.2.(Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту.)

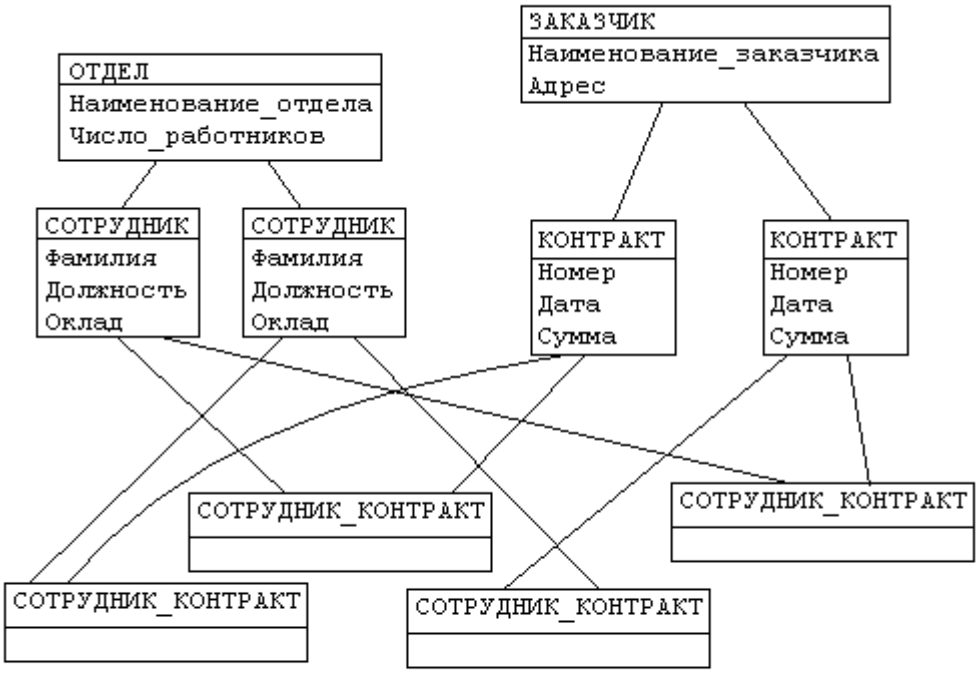


Рис. 3.2

Каждый экземпляр группового отношения характеризуется следующими признаками:

- **способ упорядочения подчиненных записей**:
  - произвольный,
  - хронологический /очередь/,
  - обратный хронологический /стек/,
  - сортированный.

  Если запись объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

- **режим включения подчиненных записей**:
  - автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
  - ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем).

- **режим исключения** Принято выделять три класса членства подчиненных записей в групповых отношениях:
  1. *Фиксированное.* Подчиненная запись жестко связана с записью владельцем и ее можно исключить из группового отношения только удалив. При удалении записи-владельца все подчиненные записи автоматически тоже удаляются. В рассмотренном выше примере фиксированное членство предполагает групповое отношение "ЗАКЛЮЧАЕТ" между записями "КОНТРАКТ" и "ЗАКАЗЧИК", поскольку контракт не может существовать без заказчика.
  2. *Обязательное.* Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи "СОТРУДНИК" и "ОТДЕЛ". Если отдел расформировывается, все его сорудники должны быть либо переведены в другие отделы, либо уволены.
  3. *Необязательное.* Можно исключить запись из группового отношения, но сохранить ее в базе данных не прикрепляя к другому владельцу. При удалении записи-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа. Примером такого группового отношения может служить "ВЫПОЛНЯЕТ" между "СОТРУДНИКИ" и "КОНТРАКТ", поскольку в организации могут существовать работники, чья деятельность не связана с выполненинем каких-либо договорных обязательств перед заказчиками.

Лекция 5. Стандарты на обмен данными между подсистемами АСУ.

астоящий стандарт распространяется на интерфейс, регламентирующий общие правила организации взаимодействия локальных подсистем в составе автоматизированных систем управления рассредоточенными объектами, использующими магистральную структуру связи (в дальнейшем - интерфейс).

В части физической реализации стандарт распространяется на интерфейсы агрегатных средств, использующих для передачи сообщений электрические сигналы.

1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

1.1. Интерфейс предназначен для организации связи и обмена информацией между локальными подсистемами в составе автоматизированных системы управления технологическими процессами, машинами и оборудованием в различных отраслях промышленности и непромышленной сфере.

1.2. Интерфейс обеспечивает взаимодействие рассредоточенных локальных подсистем, использующих спорадическую передачу информации в составе систем, функционирующих в реальном масштабе времени.

1.3. Посредством интерфейса могут сопрягаться локальные подсистемы, функционирующие в автономном режиме и реализующие частично или полностью следующие функциональные задачи:

- сбор, первичная обработка и хранение информации;
- непосредственное цифровое и супервизорное регулирование;
- программное-логическое управление;
- сопряжение с оперативно-технологическим персоналом;
- сопряжение с управляющими вычислительными комплексами верхнего яруса в иерархических системах.

2. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ

2.1. Интерфейс реализует бит-последовательный способ обмена данными по двухпроводной линии связи.

2.2. Максимальная длина линии связи (включая длину отводов) - 3 км.

2.3. Рекомендуемое количество сопрягаемых локальных подсистем - не более 60.

2.4. Номинальная скорость передачи данных должна составлять 30, 100 или 500 кбит/с.

2.5. Для представления сигналов должна применяться двухфазная модуляция с фазоразностным кодированием.

2.6. Для кодовой защиты передаваемых сообщений должен применяться циклический код с производящим полиномом $X^{16}+X^{12}+X^5+1$.

2.7. В целях устранения случайных ошибок должна быть предусмотрена возможность повторной передачи сообщений между теми же локальными подсистемами.

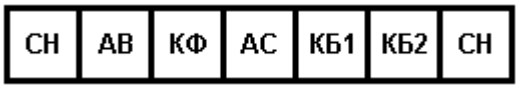
2.8. Передача сообщений между локальными подсистемами должна осуществляться посредством ограниченного набора функциональных байтов, последовательность которых устанавливается форматом сообщения. Интерфейс устанавливает два типа форматов сообщений (черт. 1).

Формат 1 имеет фиксированную длину и предназначен для передачи только интерфейсных сообщений.

Формат 2 включает переменную по длине информационную часть, предназначенную для передачи данных.

Типы форматов сообщений

Формат 1



Формат 2

Черт. 1

2.9. Форматы сообщений должны включать следующие функциональные байты:

- синхронизирующий СН;
- адрес вызываемой локальной подсистемы АВ;
- код выполняемой функции КФ;
- собственный адрес локальной подсистемы АС;
- длина информационной части (количество байтов данных) ДС, данных $ДН_1$-$ДН_n$;
- байты контрольных кодов КБ1 и КБ2.

2.9.1. Синхронизирующий байт СН служит для обозначения начала и конца сообщения. Синхронизирующему байту присвоен код 01111110.

2.9.2. Байт адреса подсистемы АВ определяет локальную подсистему, которой направляется сообщение.

2.9.3. Байт выполняемой функции КФ определяет операцию, которая выполняется в данном цикле связи. Назначение разрядов внутри байта КФ приведено на черт 2.

Структура байта КФ

Лекция 6. Проектирование баз данных. Жизненный цикл базы данных.

Только небольшие организации могут обобществить данные в одной полностью интегрированной базе данных. Чаще всего администратор баз данных (даже если это группа лиц) практически не в состоянии охватить и осмыслить все информационные требования сотрудников организации (т.е. будущих пользователей системы). Поэтому информационные системы больших организаций содержат несколько десятков БД, нередко распределенных между несколькими взаимосвязанными ЭВМ различных подразделений. (Так в больших городах создается не одна, а несколько овощных баз, расположенных в разных районах.)

Отдельные БД могут объединять все данные, необходимые для решения одной или нескольких прикладных задач, или данные, относящиеся к какой-либо предметной области (например, финансам, студентам, преподавателям, кулинарии и т.п.). Первые обычно называют *прикладными БД*, а вторые *–предметными БД* (соотносящимся с предметами организации, а не с ее информационными приложениями). (Первые можно сравнить с базами материально-технического снабжения или отдыха, а вторые – с овощными и обувными базами.)

Предметные БД позволяют обеспечить поддержку любых текущих и будущих приложений, поскольку набор их элементов данных включает в себя наборы элементов данных прикладных БД. Вследствие этого предметные БД создают основу для обработки неформализованных, изменяющихся и неизвестных запросов и приложений (приложений, для которых невозможно заранее определить требования к данным). Такая гибкость и приспосабливаемость позволяет создавать на основе предметных БД достаточно стабильные информационные системы, т.е. системы, в которых большинство изменений можно осуществить без вынужденного переписывания старых приложений.

Основывая же проектирование БД на текущих и предвидимых приложениях, можно существенно ускорить создание высокоэффективной информационной системы, т.е. системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Поэтому прикладное проектирование до сих пор привлекает некоторых разработчиков. Однако по мере роста числа приложений таких информационных систем быстро увеличивается число прикладных БД, резко возрастает уровень дублирования данных и повышается стоимость их ведения.

Таким образом, каждый из рассмотренных подходов к проектированию воздействует на результаты проектирования в разных направлениях. Желание достичь и гибкости, и эффективности привело к формированию методологии проектирования, использующей как предметный, так и прикладной подходы. В общем случае предметный подход используется для построения первоначальной информационной структуры, а прикладной – для ее совершенствования с целью повышения эффективности обработки данных.

При проектировании информационной системы необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей (сотрудников организации) [2, 3, 4, 6, 8, 9, 10]. Сбор данных начинается с изучения сущностей организации и процессов, использующих эти сущности (подробнее в приложении Б). Сущности группируются по "сходству" (частоте их использования для выполнения тех или иных действий) и по количеству ассоциативных связей между ними (самолет – пассажир, преподаватель – дисциплина, студент – сессия и т.д.). Сущности или группы сущностей, обладающие наибольшим сходством и (или) с наибольшей частотой ассоциативных связей объединяются в предметные БД. (Нередко сущности объединяются в предметные

БД без использования формальных методик – по "здравому смыслу".) Для проектирования и ведения каждой предметной БД (нескольких БД) назначается АБД, который далее занимается детальным проектированием базы.

Далее будут рассматриваться вопросы, связанные с проектированием отдельных реляционных предметных БД.

Основная цель проектирования БД – это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Так называемый, "чистый" проект БД ("Каждый факт в одном месте") можно создать, используя методологию нормализации отношений. И хотя нормализация должна использоваться на завершающей проверочной стадии проектирования БД, мы начнем обсуждение вопросов проектирования с рассмотрения причин, которые заставили Кодда создать основы теории нормализации.

Лекция 7. Синтез логических структур локальных и распределенных баз данных.

Решение задачи синтеза оптимальной логической структуры РБД, обеспечивающей минимум времени выполнения множества запросов пользователей по критерию (7.4.17), включает два взаимосвязанных шага. [1]

Решение задачи синтеза оптимальной логической структуры БД для режима ее эксплуатации при наличии нескольких возможных точек входа в структуру по каждому запросу (задача (5.1.12), (5.1.5) - (5.1.11)) предлагается осуществлять в два этапа. [2]

Решение задачи синтеза оптимальной логической структуры СБД при наличии нескольких точек входа (целевая функция (5.1.20)) осуществляется аналогично решению задачи (5.1.12) и проводится также в 2 этапа. [3]

Результатом *решения задач синтеза оптимальных логических структур БД* является определение числа и состава логических записей, выбор структуры связей между записями, определение записей, выбираемых в качестве точек входа в структуру. [4]

При *решении задачи синтеза оптимальной логической структуры РВД*, минимизирующей общее время реализации множества запросов пользователей, алгоритм перечисления ра и операцию сокращения Ф целесообразно рассматривать в композиции Ф а. [5]

Точный алгоритм *решения задачи синтеза оптимальной логической структуры ЛБД* по критерию минимума связности синтезируемых записей (задача (5.1.19)) аналогичен рассмотренным выше. Данный алгоритм аналогичен рассмотренному ниже алгоритму для решения задачи декомпозиции логической структуры СБД на кластеры. [6]

Рассмотрим точный алгоритм *решения задачи синтеза оптимальной логической структуры РБД*, минимизирующей суммарное время выполнения транзакций. [7]

Рассмотрим приближенные алгоритмы *решения задач синтеза оптимальных логических структур РБД* и БмД репозитария. [8]

Аналогичные приближенные алгоритмы предложены для *решения задач синтеза оптимальных логических структур РБД* и структуры БмД репозитария по другим критериям эффективности. Так, например, алгоритм решения задачи синтеза по критерию минимума общего времени выполнения множества транзакций состоит из следующих этапов. [9]

Разработанные в данном параграфе формализованные описания характеристик канонической структуры РБД, множества запросов, транзакций, пользователей РБД, репозитария, узлов и топологии ВС, а также методы расчета основных временных, стоимостных и объемных характеристик функционирования РБД используются для постановки и *решения задач синтеза оптимальных логических структур РБД* и репозитария. [10]

При удовлетворении осуществляется вывод на печать решения задачи. В противном случае следует переход на блок, в котором осуществляются присвоения Iq - / o - 1, J - Jo- Рассмотрим алгоритм *решения задачи синтеза оптимальной логической структуры СБД* для запросов реального масштаба времени. Алгоритм состоит из следующих шагов. [11]

БД, и общего времени счета разработаны приближенные алгоритмы, базирующиеся на учете специфики поставленных задач, анализе структуры ограничений и графовой интерпретации вариантов решения задач. Рассмотрим алгоритмы *решения задачи синтеза оптимальных логических структур ЛБД* и СБД для режима обработки заданного множества запросов пользователей. [12]

В данном параграфе последовательно рассматриваются методы и алгоритмы решения задач синтеза оптимальных логических и физических структур локальных, сетевых и распределенных БД. БД относятся к классу задач дискретного целочисленного программирования с булевыми переменными. Для решения поставленных задач синтеза разработаны эффективные точные и приближенные алгоритмы. Доказан ряд утверждений, позволяющих получить аналитические выражения для точной нижней границы множества*решений задач синтеза оптимальных логических структур БД*. Получены аналитические выражения для оценок вершин деревьев множеств решений задач синтеза. [13]

Синтез логической структуры РБД рассматривается в работе как поиск оптимального варианта отображения канонической структуры РБД в логическую, обеспечивающего оптимальное значение заданного критерия эффективности функционирования корпоративных АИУС и удовлетворяющего основным системным, сетевым и структурным ограничениям. Логическая структура РВД и структура размещения БмД репозитария обеспечивают сохранение семантических свойств и характеристик информационных элементов и взаимосвязей, зафиксированных в канонической структуре РБД, с учетом ограничений, накладываемых параметрами СУРБД и СУБД локальных БД, аппаратными средствами передачи данных, топологией ВС и требованиями различных режимов функционирования корпоративных АИУС. Основными критериями эффективности синтеза логических структур РБД являются: минимум общего времени последовательной и параллельной обработки множества запросов пользователей, в т.ч. при наличии многопроцессорных серверов в отдельных узлах ВС; минимум общего времени последовательного выполнения множества транзакций; минимум стоимости функционирования корпоративной АИУС. Ограничениями задач синтеза являются ограничения на число групп данных в составе логических записей, на длину формируемых логических записей, на количество синтезируемых логических записей и

ЛБмД, размещаемых в узлах ВС, на требуемый уровень информационной безопасности системы и др. В результате *решения задач синтеза оптимальных логических структур РБД* определяются: оптимальные характеристики логической структуры РБД (состав и структуры логических записей и взаимосвязей, структура размещения логических записей по серверам баз данных); структура размещения ЛБмД репозитария по серверам узлов ВС; оптимальные структуры реализации запросов и транзакций. [14]

Лекция 8. **Язык управления базами данных для реляционных баз данных SQL**

Из истории SQL:

В начале 70-х годов в компании IBM была разработана экспериментальная СУБД System R на основе языка SEQUEL (Structured English Qeury Language - структурированный английский язык запросов), который можно считать непосредственным предшественником SQL. Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. В 1981 году IBM объявила о своем первом, основанном на SQL программном продукте, SQL/DS. Чуть позже к ней присоединились Oracle и другие производители. Первый стандарт языка SQL был принят Американским национальным институтом стандартизации (ANSI) в 1987 (так называемый SQL level /уровень/ 1) и несколько уточнен в 1989 году (SQL level 2). Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 нового расширенного стандарта (ANSI SQL-92 или просто SQL-2). В настоящее время ведется работа по подготовке третьего стандарта SQL, который должен включать элементы объекто-ориентрованного доступа к данным.

Необходимо сказать, что хотя SQL и задумывался как средство работы конечного пользователя, в конце концов он стал настолько сложным, что превратился в инструмент программиста. Вопросы создания приложений обработки данных с использованием SQL рассматриваются в конце данной главы.

В SQL определены два подмножества языка:

- **SQL-DDL** (Data Definition Language) - язык определения структур и ограничений целостности баз данных. Сюда относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д.
- **SQL-DML** (Data Manipulation Language) - язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями

Здесь не дается строгое описание всех возможностей SQL-92. Во-первых, ни одна СУБД не поддерживает их в полной мере, а во-вторых, производители СУБД часто предлагают собственные расширения SQL, несовместимые друг с другом. Поэтому мы рассматриваем некое подмножество языка, которое дает общее представление о его специфике и возможностях. В то же время, этого подмножества достаточно, чтобы начать самостоятельную работу с любой СУБД. Более формальный (и более полный) обзор стандартов SQL сделан в статье С. Д. Кузнецова "Стандарты языка реляционных баз данных SQL: краткий обзор",журнал СУБД N 2, 1996 г. Ознакомится с русским переводом стандарта SQL можно на сервере Центра информационных технологий, сравнительное описание различных версий языка (для СУБД Sybase SQL Server, Sybase SQL Anywhere, Microsoft SQL Server, Informix, Oracle Server) приводится в книге Дж.Боуман, С.Эмерсон, М.Дарновски "Практическое руководство по SQL", Киев, Диалектика, 1997.

Следует также отметить, что в отличие от "теретической" терминологии, используемой при описании реляционной модели (*отношение, атрибут, кортеж*), в литературе при описании SQL часто используется терминология "практическая" (соответственно - *таблица, столбец, строка*). Здесь мы следуем этой традиции.

Все примеры построены применительно к базе данных **publications**, содержащей сведения о публикациях (как печатных, так и электронных), относящихся к теме данного курса. Структуру этой базы данных можно посмотреть здесь, ее проектирование описано в разделе 5.4, доступ к ней для практических занятий можно получить через Internet посредством СУБД Leap (реляционная алгебра) или СУБД PostgreSQL. (язык SQL).

Типы данных SQL.

- Символьные типы данных - содержат буквы, цифры и специальные символы.
  - **CHAR** или **CHAR(n)** -символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответсвует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
  - **VARCHAR(n)** - символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.
- Целые типы данных - поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).
  - **INTEGER** или **INT**- целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения)* Интервал значений от - 2147483647 до + 2147483648
  - **SMALLINT** - короткое целое (2 байта), интервал значений от - 32767 до +32768
- Вещественные типы данных - описывают числа с дробной частью.
  - **FLOAT** и **SMALLFLOAT** - числа с плавающей точкой (для хранения отводится обычно 8 и 4 байта соответсвенно).
  - **DECIMAL(p)** - тип данных аналогичный **FLOAT** с числом значащих цифр **p**.
  - **DECIMAL(p,n)** - аналогично предыдущему, **p** - общее количество десятичных цифр, **n** - количество цифр после десятичной запятой.
- Денежные типы данных - описывают, естественно, денежные величины. Если в ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n).
  - **MONEY(p,n)** - все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.
- Дата и время - используются для хранения даты, времени и их комбинаций. Большинство СУБД умеет определять интервал между двумя датами, а также уменьшать или увеличивать дату на определенное количество времени.
  - **DATE** - тип данных для хранения даты.
  - **TIME** - тип данных для хранения времени.
  - **INTERVAL** - тип данных для хранения верменного интервала.

    - **DATETIME** - тип данных для хранения моментов времени (год + месяц + день + часы + минуты + секунды + доли секунд).
- Двоичные типы данных - позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:
    - **BINARY**
    - **BYTE**
    - **BLOB**
- Последовательные типы данных - используются для представления возрастающих числовых последовательностей.
    - **SERIAL** - тип данных на основе **INTEGER**, позволяющий сформировать уникальное значение (например, для первичного ключа). При добавлении записи СУБД автоматически присваивает полю данного типа значение, получаемое из возрастающей последовательности целых чисел.

В заключение следует сказать, что для всех типов данных имеется общее значение **NULL** - "не определено". Это значение имеет каждый элемент столбца до тех пор, пока в него не будут введены данные. При создании таблицы можно явно указать СУБД могут ли элементы того или иного столбца иметь значения **NULL** (это не допустимо, например, для столбца, являющего первичным ключом).

Лекция 9. Язык манипуляции данными (DML)

Язык SQL имеет две составляющие: язык обращения с данными Data Manipulation Language (DML) и язык определения данных Data Definition Language (DDL). DML состоит из операторов, используемых для создания и получения данных. DDL состоит из операторов, используемых для создания объектов в базе данных и для установки свойств и значений атрибутов самой базы данных.

DML и DDL

Чем же отличаются эти две группы операторов? В то время, как операторы DML достаточно однотипны для различных реализаций SQL (что дает возможность каждому поставщику программной продукции вводить свои расширения), DDL имеет существенные различия для разных продуктов. Каждый поставщик системы управления базой данных на физическом уровне различным образом реализует реляционную модель и каждый поставщик DDL неизбежно отражает эти различия. Большинство поставщиков предоставляют графические инструменты для определения данных и многие, включая и Microsoft, не ограничивают вас использованием только SQL DDL. Например, Microsoft предоставляет поддержку двух стандартов определения данных: ADO и DAO.

Мы уже успели рассмотреть основные операторы DML: SELECT, INSERT, UPDATE и DELETE. Базовыми же операторами SQL DDL являются CREATE, ALTER и DROP, каждый из которых имеет несколько вариаций для создания объектов различных типов.

Язык определения данных (Data Definition Language)

Не многие программисты создают базу данных программным путём, большинство из нас для этого используют некую визуальную среду наподобие MS Access для построения файла MDB. Но иногда нам всё таки приходится создавать и удалять базу данных, а так же объекты базы данных программным путём. Для этого используется наиболее распространённая на сегодняшний день технология Structured Query Language Data Definition Language (SQL DDL). Выраджения языка определения данных (DDL) — это SQL выражения, которые поддерживают определения или объявления объектов базы данных (например, CREATE TABLE, DROP TABLE, CREATE INDEX либо подобные им).

Давайте взглянем на простейший пример выражения CREATE TABLE:

CREATE TABLE PhoneBook(
Name TEXT(50)
Tel TEXT(50));

Данное DDL выражение (для MS Access) в время выполнения создаст новую таблицу с названием PhoneBook. Таблица PhoneBook будет иметь два поля: Name и Tel. Оба поля имеют строковый тип (TEXT) и размер поля 50 символов.

Язык манипулирования данными

Язык манипулирования данными — командный язык, обеспечивающий выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

К базовым средствам манипулирования данными языка SQL относятся «поисковые» варианты операторов UPDATE и DELETE. Эти варианты называются поисковыми, потому что при задании соответствующей операции задается логическое условие, налагаемое на строки адресуемой оператором таблицы, которые должны быть подвергнуты модификации или удалению. Кроме того, в такую категорию языковых средств входит оператор INSERT, позволяющий добавлять строки в существующие таблицы.

Лекция 10. Язык определения доступа к данным (DCL)

SQL Server 2000 поддерживает две модели проверки права пользователя устанавливать соединения с сервером:

Windows only — интегрированная модель, при которой регистрация пользователя производится на основе его текущего доменного идентификатора или идентификатора пользователя компьютера — в случае, если вход был осуществлен не в домен, а на локальный компьютер. Соответственно, при таком доступе могут использоваться только trusted-соединения.

SQL Server and Windows — смешанная модель, когда допускается использование входа как на основе идентификатора пользователя Windows, так и на основе логина и пароля, передаваемого непосредственно SQL-серверу.

Выбор конкретной модели производится на этапе установки и затем может быть изменен, например, через SQL Enterprise Manager.

После того как соединение с сервером установлено, каждому пользователю, независимо от принятой модели, ставится в соответствие учетная запись (Login). Не следует путать Login и имя учетной записи, которое вводится вместе с паролем при соединении к серверу. То, что вводится вместе с паролем, — это символьное имя учетной записи (Login name). Более универсальным идентификатором учетной записи является SID (Securty Identifier). SID — это бинарное значение размером до 85 байт. В свою очередь, Login name — это символьное имя, которое ему сопоставлено.

На одном сервере не может быть двух одинаковых имен учетной записи. Однако, если на разных серверах существуют учетные записи с одинаковыми именами, то это не значит, что им будут сопоставлены одинаковые идентификаторы SID. В связи с этим часто возникают проблемы при переносе базы данных с одного сервера на другой. Лишь стандартные учетные записи имеют фиксированные (то есть всегда одинаковые) идентификаторы SID. Для всех остальных учетных записей SID генерируются при создании.

Хорошей практикой при задании прав подключения является использование Windows Authentification и объедениние пользователей в группы. Это позволяет экономить время при перемещении пользователя из группы в группу и тонко настраивать набор привилегий каждого конкретного члена группы. Кроме того, данный принцип позволяет использовать весь набор преимуществ Windows-аутентификации, таких как требование к сложности пароля, его изменение по прошествии определенного периода, время, в которое пользователь может входить в систему, и т. П.

Вопросы для самопроверки

Какая из моделей проверки права пользователя устанавливать соединения с сервером принята в нашем институте?

Какую модель проверки прав пользователей вы применили на своем домашнем MS SQL Server 2000?

Какие проблемы могут возникнуть при переносе базы данных с вашего домашнего компьютера на институтский сервер и почему?

Серверные роли

Для сервера существует несколько серверных ролей (табл. 1). Набор серверных ролей фиксирован: нельзя ни добавить новые роли, ни удалить существующие. Каждая из этих ролей имеет свой набор прав для работы с сервером. В роли могут быть включены любые учетные записи. Члены любой роли также могут добавлять в свою роль другие учетные записи.

Таблица 1. Список серверных ролей для MS SQL Server 2000

Краткое имя роли	Полное имя роли	Права
sysadmin	System Administrators	Полные, ничем не ограниченные права на сервер
setupadmin	Setup Administrators	Конфигурирование хранимых процедур, запускаемых при запуске сервера; управление связанными серверами
serveradmin	Server Administrators	Изменение настройки сервера, остановка сервера, управление полнотекстовым поиском, управление низкоуровневыми настройками таблиц
securityadmin	Security Administrators	Управление учетными записями (добавление, удаление, изменение пароля и базы данных по умолчанию и т. П.), чтение журнала ошибок сервера
diskadmin	Disck Administrators	Используется только для совместимости с предыдущими версиями сервера
dbcreator	Database Creators	Создание, удаление, изменение, восстановление базы данных
bulkadmin	Bulk Insert Administrators	Выполнение операций массовой вставки в таблицы
processadmin	Process Administrators	Уничтожение процессов, то есть соединений других пользователей с сервером

1.2. Специальные учетные записи

В MS SQL Server существуют две специальные учетные записи. Эти записи создаются при установке: Builtin\Administrators и sa. Первая из них — администраторы домена (или компьютера), на котором установлен SQL Server. Вторая учетная запись использует SQL Server-аутентификацию. Обе эти учетные записи включены после инсталляции сервера в серверную роль sysadmin.

Если члены роли Builtin\Administrators не должны иметь административных прав на сервер базы данных, то рекомендуется исключить ее из sysadmin. Однако стоит отметить, что невозможно сделать настоящую защиту от администратора компьютера, на котором установлен сервер. Пользователя sa рекомендуется удалять из соображений безопасности (предоставив предварительно администраторские права какому-либо другому

пользователю). Как минимум, требуется, чтобы его пароль был непустым, иначе безопасность вашего сервера окажется под угрозой.

Перечень всех серверных ролей можно посмотреть также с помощью графического интерфейса Enterprise Manager. Для этого надо открыть папку Security, установить курсор на «золотой ключик» с названием Server Roles, и тогда в правом окне появится список серверных ролей с указанием их прав — правда, на английском языке (рис. 1).

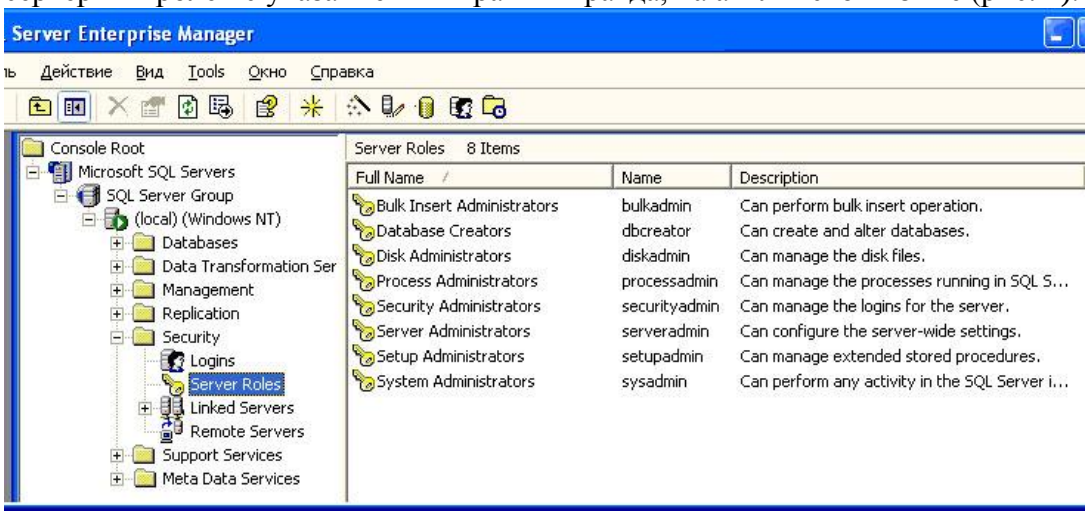


Рис. 1. Стандартные серверные роли

Как посмотреть свою серверную роль? Для этого надо выбрать свой логин в списке логинов папки Security, щелчком правой кнопки вызвать контекстное меню, выбрать в нем пункт Свойства и перейти на закладку Server roles (рис. 2).

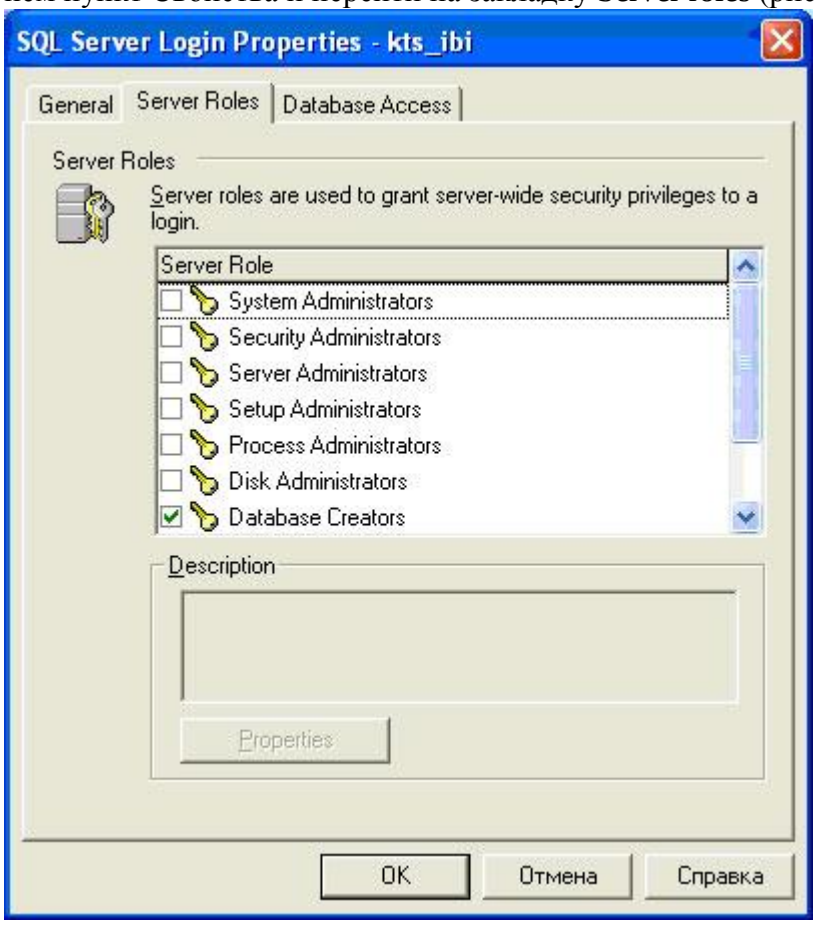


Рис. 2. Серверные роли конкретного пользователя

1.3. Права на доступ к базе данных

Факт установления соединения с сервером еще не дает пользователю права осуществлять манипуляции с объектами сервера.
Для каждой базы данных SQL Server хранит независимый набор пользователей и групп, в которые они входят. По умолчанию в каждой базе существует пользователь dbo — владелец базы данных, и группа public, к которой принадлежат все имена пользователей этой базы. Каждое имя пользователя может дополнительно принадлежать еще и к другим группам. Выделяются фиксированные роли базы данных, обладающие стандартным набором прав. Также могут быть созданы дополнительные группы, и им назначены права. Набор фиксированных ролей баз данных приведен в табл. 2.
Таблица 2. Стандартные роли пользователей баз данных

Имя роли	Права
Public	Роль, в которую входят все пользователи базы данных. Используется для задания прав по умолчанию
Db_securityadmin	Управление правами доступа других пользователей к объектам базы данных
Db_owner	Полные, ничем не ограниченные права на базу данных
Db_denydatawriter	Запрещение чтения любых данных, даже в том случае, если был явно предоставлен доступ к каким-либо данным
Db_denydatareader	Запрещение записи любых данных, даже в том случае, если был явно предоставлен доступ к каким-либо данным
Db_datawriter	Разрешение записи данных в любую таблицу и представление
DB_Datareader	Разрешение чтения данных из любой таблицы и представления
Db_ddladmin	Создание и изменение объектов базы данных
Db_backupoperator	Выполнение резервного копирования базы данных
Db_accessadmin	Создание, изменение, удаление пользователей базы данных

Каждой роли и/или каждому пользователю БД могут быть назначены права на любой из объектов БД (чтение, запись, изменение, удалении данных, выполнение ссылочной целостности и выполнение процедур). Если имеется несколько пользователей с одинаковыми правами, то их следует объединять в группы и назначать права этой группе.
Чтобы конкретный пользователь имел доступ к базе данных, его учетная запись должна быть ассоциирована с каким-либо пользователем в этой базе данных. Одна учетная запись может соответствовать в разных БД разным пользователям. В одной БД одна учетная запись может соответствовать только одному пользователю. Один пользователь БД может соответствовать не более чем одной учетной записи.
Если ваша учетная запись не сопоставлена никакому пользователю БД, вы, тем не менее, можете получить доступ к объектам БД, если в ней заведен специальный пользователь — guest. Под этим именем доступ к объектам БД получают те, чья учетная запись не сопоставлена конкретному пользователю.
Надо отметить, что учетные записи, входящие в серверную роль sysadmin, всегда сопоставляются с пользователем dbo, даже в том случае, если базу создавал какой-либо другой пользователь сервера. Кроме того, пользователям, применяющим Windows-authentification, могут быть предоставлены права на объекты БД напрямую, без предварительного сопоставления с пользователем БД. Однако делать этого не рекомендуется.

Посмотреть текущее состояние ролей вашей базы данных можно, выбрав в объектах базы данных Roles и щелкнув на любой роли в правом окне. Вы увидите список пользователей, которые принадлежат к данной роли в вашей базе данных (рис. 3).

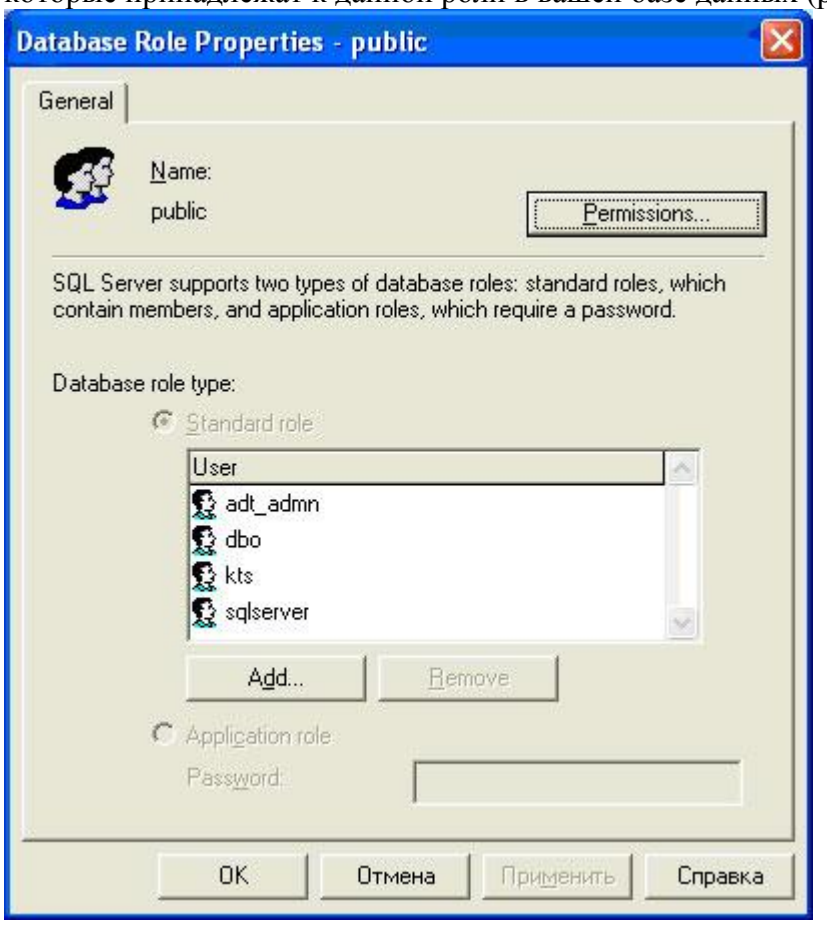


Рис. 3. Список пользователей, имеющих конкретную роль в базе данных

Одновременно в данном интерфейсе можно добавить к роли нового пользователя. Однако необходимо помнить, что этот пользователь уже должен иметь право на подключение к серверу базы данных, то есть быть легальным пользователем сервера MS SQL Server 2000, на котором установлена ваша база данных, и быть пользователем самой базы данных. Сделать любого пользователя домена пользователем сервера баз данных вы не можете, потому что не обладаете соответствующими правами. А какова должна быть роль пользователя, чтобы он мог разрешить другому пользователю доступ к серверу?

1.4. Права на доступ к объектам базы данных

Типы разрешений

Следующий уровень безопасности — это разрешения на уровне объектов базы данных. Существует несколько видов разрешений на объекты базы данных.

Во-первых, это разрешения на операции Select, Update, Insert и Delete (операции выборки, изменения, вставки и удаления). Эти разрешения применимы только для таблиц и представлений.

Для хранимых процедур и определенных пользователями функций существует разрешение на выполнение (Execute). Если в отношении некоторой процедуры или функции для пользователя предоставлено такое разрешение, то он может ее выполнять, иначе — нет.

Наконец, существует разрешение на декларативную целостность (DRI). Такие разрешения допускаются для таблиц, представлений и пользовательских функций. Если пользователь имеет разрешение на DRI для некоторой таблицы, это означает, что он может создавать в других таблицах связи, ссылающиеся на ту таблицу, для которой пользователь имеет это разрешение.

Кроме того, забегая немного вперед, скажем, что при создании представления можно указать, что это представление привязано к схеме данных. Это позволяет отслеживать попытки изменения объектов, на основе которых строится представление или функция. Например, если представление строится на основе таблицы Students и используется привязка этого представления к схеме, то будет невозможно удалить таблицу Students, пока такая привязка существует. Также будет невозможно изменить ту часть таблицы (столбцы), которая присутствует в описании представления. Для того чтобы осуществить привязку представления или функции к схеме, необходимо, чтобы на все ссылающиеся объекты для пользователя, создающего или изменяющего данное представление или функцию, имелись разрешения типа DRI.

Разрешения типа DRI для представлений и пользовательских функций имеют такой же смысл, как и для таблиц, за исключением того, что они не дают возможности устанавливать связи, поскольку ни с представлениями, ни с функциями нельзя установить связь посредством внешнего ключа. Таким образом, для представлений и пользовательских функций разрешения типа DRI позволяют привязывать к схеме объекты, которые основываются на данных представлениях или функциях.

Предоставление и запрещение доступа

В MS SQL Server существует три состояния возможности доступа: разрешение (GRANT), запрет (DENY) и неявное отклонение доступа (REVOKE).

Оператор GRANT имеет следующий синтаксис:

GRANT {<список действий> | ALL PRIVILEGES}
ON <имя_объекта> TO {<имя_пользователя> | PUBLIC}
[WITH GRANT OPTION]

Список действий определяет набор действий из общедопустимого перечня действий над объектом данного типа.

Опция ALL PRIVILEGES указывает, что разрешены все действия из допустимых для объектов данного типа.

<Имя_объекта> задает имя конкретного объекта: таблицы, представления, хранимой процедуры, триггера.

<Имя_пользователя> или PUBLIC определяет, кому предоствляются данные привилегии.

Опция WITH GRANT OPTION является необязательной. Она определяет режим, при котором передаются не только права на указанные действия, но и право передавать (делегировать) эти права другим пользователям. Передавать права в этом случае пользователь может только в рамках разрешенных ему действий.

Для отмены ранее назначенных привилегий в стандарте SQL определен оператор REVOKE. Оператор отмены привилегий имеет следующий синтаксис:

REVOKE {<список операций> | ALL PRIVILEGES}
ON <имя_объекта>
FROM {<список пользователей> | PUBLIC} {CASCADE | RESTRICT}

Опции CASCADE или RESTRICT определяют, каким образом должна производиться отмена привилегий. Опция CASCADE отменяет привилегии не только того пользователя, который непосредственно упоминался в операторе GRANT при предоствлении ему привилегий, но и всем пользователям, которым были присвоены привилегии данным пользователем с использованием данной ему опции WITH GRANT OPTION.

С разрешением и запретом все интуитивно понятно: происходит либо предоставление некоторых прав пользователю, либо выставление запретов. Что касается неявного отклонения доступа, можно считать, что это удаление ранее предоставленного разрешения или запрета для данного пользователя. Если бы не было возможности неявного отклонения доступа, то для некоторого конкретного пользователя можно было бы либо дать разрешение, либо выставить запрет на некоторый конкретный объект базы данных. С помощью же неявного отклонения доступа можно убрать как явное

разрешение, так и явное отклонение доступа, то есть, другими словами, установить неопределенность.

Почему такое состояние называется неявным отклонением доступа, а не его неявным предоставлением? Дело в том, что здесь применяется политика, в соответствии с которой запрещается все, что явно не разрешено. То есть если у пользователя не будет ни запрета, ни разрешения на некоторую операцию, то он не сможет ее выполнить.

Тогда возникает вопрос: в чем разница между явным и неявным запретом? Дело в том, что для одного и того же объекта на одну и ту же операцию пользователь может иметь сразу несколько разрешений (или запретов). То есть непосредственно пользователю, конечно, может быть выдано только одно разрешение. Однако, вспомним, что пользователь может входить в множество различных ролей базы данных. Например, для базы данных института можно рассмотреть роли Преподаватели, Студенты, Сотрудники деканата, Сотрудники профсоюза, Лаборанты, Системные администраторы. Один пользователь может входить сразу в несколько ролей. Для некоторого объекта на одну операцию (например, операцию удаления) могут быть выставлены и запреты, и разрешения. Кроме того, они могут быть выставлены еще дополнительно непосредственно для каждого пользователя.

Таким образом, каждый пользователь базы данных может иметь разрешения, предоставленные непосредственно ему или посредством вхождения в роли. Причем может получиться, что эти разрешения и запреты будут противоречить друг другу. Например, операция удаления будет запрещена для роли Студенты, но разрешена для роли Лаборанты. Если учесть, что некоторый пользователь может входить в обе роли, как будет определяться возможность или невозможность выполнить ту или иную операцию в таких случаях? Наиболее весомым считается запрет, из тех соображений, что главное — не допустить (по ошибке) несанкционированного доступа к данным. Если же кто-либо из пользователей недополучит каких-либо прав, то он обратится к администратору и тот предоставит ему необходимые права. То есть считается, что лучше недодать прав, чем дать их слишком много. В этом и состоит разница между REVOKEи DENY: если пользователю одновременно даны разрешение (GRANT) и запрет (DENY), то результатом будет запрет, а если же дополнительно к разрешению дано неявное отклонение доступа, то результатом будет разрешение.

Интерпретация двойных разрешений приведена в табл. 3. Здесь видно, что будет происходить при наличии разных разрешений.

Таблица 3. Интерпретация двойных назначений разрешений

Разрешение 1	Разрешение 2	Результат	Итоговое разрешение
Grant	Deny	Deny	Нет
Grant	Revoke	Grant	Есть
Grant	Grant	Grant	Есть
Deny	Deny	Deny	Нет
Deny	Revoke	Deny	Нет
Deny	Grant	Deny	Нет
Revoke	Deny	Deny	Нет
Revoke	Revoke	Revoke	Нет
Revoke	Grant	Grant	Есть

Иерархия прав доступа

Выдать разрешения на объект базы данных может либо владелец объекта, либо владелец базы данных, либо любой другой пользователь, которому уже предоставлены эти права с дополнительной опцией, позволяющей выдавать это разрешение кому-либо еще.
Владельцем объекта базы данных является пользователь, который его создал. В принципе, можно передавать владение объектом базы данных другому пользователю, но это используется нечасто.
Назначение прав пользователям базы данных, также как и большинство функций администрирования, может быть выполнено с использованием графического интерфейса Enterprise Manager. Для этого необходимо установить курсор на конкретного пользователя — или на пользовательскую роль в вашей базе данных, если надо предоставить одинаковые права всем членам данной роли, — и нажать кнопку Permissions.
В открывшемся окне надо установить «галочки» в соответствующие графы разрешений. Каждая графа отвечает за разрешение определенных действий над определенным объектом (рис. 4).

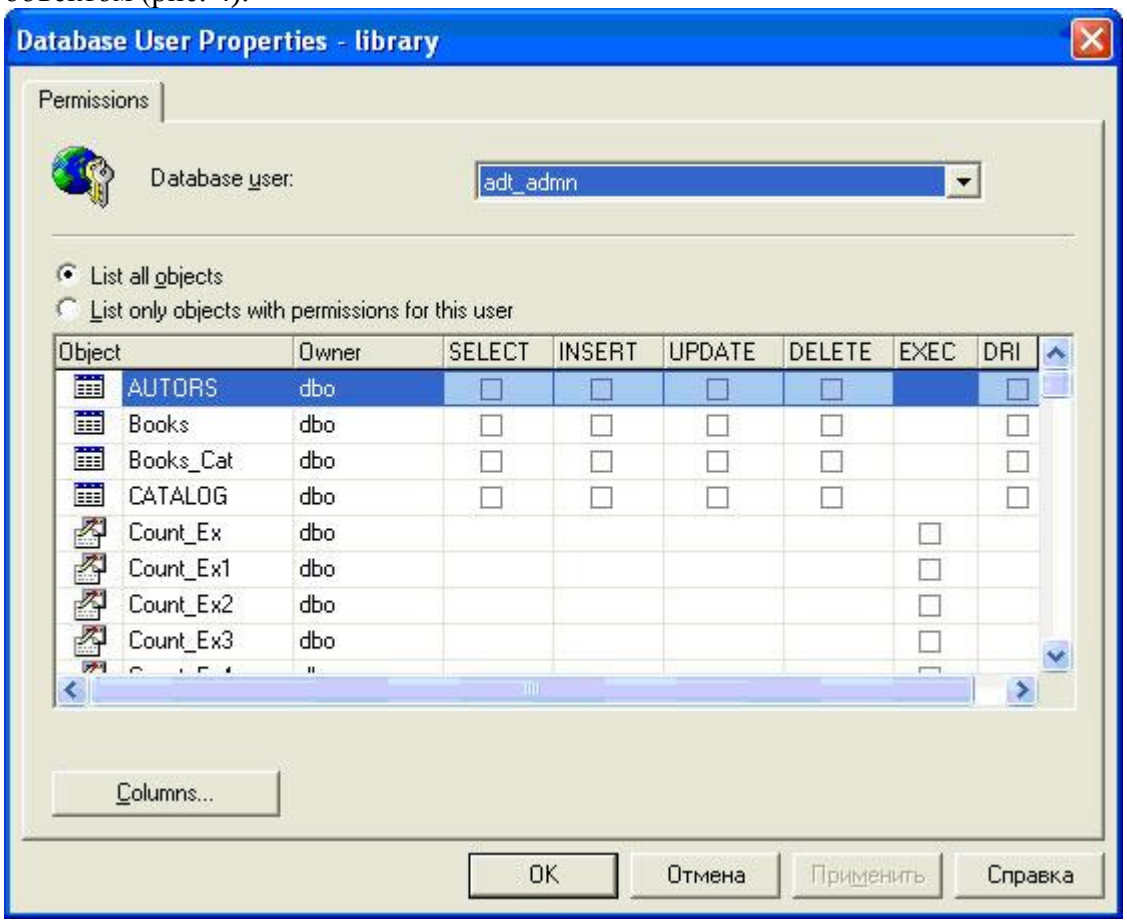


Рис. 4. Интерфейс предоставления прав пользователям
Перечень доступных операций зависит от типа объекта. Если требуется еще более тонко дифференцировать права доступа, то можно воспользоваться кнопкой Columns и установить права на работу с отдельными атрибутами.

Лекция 11. Язык управления транзакциями (TCL)

ТРАНЗАКЦИЯ – это ряд операций манипулирования данными SQL,которые выполняют логическую единицу работы. Например, две операции UPDATE кредитуют один банковский счет и дебитируют другой. В один момент времени ORACLE либо делает

постоянными, либо отменяет все изменения в базе данных, осуществленные транзакцией. Если ваша программа сбивается в середине транзакции, ORACLE обнаруживает ошибку и выполняет отмену (откат) транзакции. Следовательно, база данных автоматически возвращается в свое прошлое состояние.

Для управления транзакциями используются команды COMMIT, ROLLBACK, SAVEPOINT и SET TRANSACTION. COMMIT делает постоянными все изменения в базе данных, сделанные в течение текущей транзакции («подтверждает» транзакцию). До тех пор, пока вы не подтвердите свои изменения, другие пользователи не могут их увидеть. ROLLBACK заканчивает текущую транзакцию и отменяет все изменения, сделанные с момента ее начала. SAVEPOINT отмечает текущую точку в обработке транзакции. Совместно с ROLLBACK, команда SAVEPOINT позволяет отменить часть транзакции. SET TRANSACTION устанавливает режим транзакции
«только чтение».

Обработка транзакций

Первое предложение SQL в вашей программе начинает транзакцию. Когда одна транзакция заканчивается, очередное предложение SQL автоматически начинает следующую транзакцию. Таким образом, каждое предложение SQL является частью некоторой транзакции.

Использование COMMIT

Предложение COMMIT завершает текущую транзакцию и делает постоянными все изменения, осуществленные в течение этой транзакции. До этого момента другие пользователи не могут видеть измененных данных; они видят данные в том состоянии, каким оно было к моменту начала транзакции.

Рассмотрим простую транзакцию, которая осуществляет перевод денег с одного банковского счета на другой. Эта транзакция требует двух операций обновления (UPDATE), потому что она должна дебитовать один счет и кредитовать другой. После кредитования второго счета вы выдаете команду COMMIT, делая изменения постоянными. Лишь после этого новое состояние счетов становится видимым другим пользователям.

```
BEGIN
 …
 UPDATE accts SET bal = my_bal – debit
WHERE acctno = 7715;
 …
 UPDATE accts SET bal = my_bal + credit
WHERE acctno = 7720;
 COMMIT WORK;
END;
```

Необязательное ключевое слово WORK не имеет никакого эффекта, помимо улучшения читабельности.

Предложение COMMIT освобождает все блокировки таблиц и строк. Оно также стирает все точки сохранения (обсуждаемые ниже), отмеченные после последней операции COMMIT или ROLLBACK.

Использование ROLLBACK

Предложение ROLLBACK противоположно COMMIT. Оно заканчивает текущую транзакцию и отменяет все изменения, осуществленные за время этой транзакции. Предложение ROLLBACK полезно по двум причинам.:

если вы сделали ошибку, например, удалили не ту строку из базы данных, вы можете использовать ROLLBACK для восстановления первоначальных данных. Вариант ROLLBACK TO позволяет вам отменить изменения до промежуточной точки в текущей транзакции, так что вы не обязаны стирать все ваши изменения.

Предложение ROLLBACK полезно, когда вы начали транзакцию, которую не в состоянии завершить, например, при возникновении исключения или ошибки в предложении SQL. В таких случаях ROLLBACK позволяет вам вернуться к стартовой точке, так что вы можете предпринять корректирующие действия и попытаться снова повторить транзакцию.

Рассмотрим следующий пример, в котором вы вставляете информацию о сотруднике в три различных таблицы базы данных. Все три таблицы имеют столбец, содержащий номер сотрудника и ограничиваемый уникальным индексом. Если предложение INSERT пытается вставить повторяющийся номер сотрудника, возбуждается предопределенное исключение DUP_VAL_ON_INDEX. В этом случае вам необходимо отменить все изменения. Поэтому вы выдаете ROLLBACK в обработчике исключений.

```
DECLARE
 emp_id INTEGER;
 …
BEGIN
 SELECT empno, … INTO emp_id, … FROM new_emp WHERE …
 …
 INSERT INTO emp VALUES (emp_id, …);
 INSERT INTO tax VALUES (emp_id, …);
 INSERT INTO pay VALUES (emp_id, …);
 …
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
ROLLBACK;
 …
END;
```

Откаты на уровне предложений

Прежде чем исполнять предложение SQL, ORACLE выдает неявную точку сохранения. Затем, если это предложение сбивается, ORACLE автоматически выполняет его откат. Например, если предложение INSERT пытается вставить повторяющееся значение в уникальный индекс, оно откатывается. При этом теряется лишь работа, начатая сбившимся предложением SQL; вся работа, проделанная в текущей транзакции до этого момента, не затрагивается.

Прежде чем исполнять предложение SQL, ORACLE должен выполнить его РАЗБОР (parse), т.е. исследовать его, чтобы убедиться, что оно синтаксически корректно и ссылается на действительные объекты базы данных. Ошибки, обнаруженные во время разбора предложения (в отличие ошибок во время выполнения) не приводят к откату.

Использование SAVEPOINT

SAVEPOINT отмечает и именует текущую точку (точку сохранения) в процессе транзакции. Такая точка, используемая в предложении ROLLBACK TO, позволяет отменить часть транзакции. В следующем примере вы отмечаете точку сохранения перед тем, как выполнять вставку строки. Если предложение INSERT попытается вставить повторяющееся значение в столбец empno, возникнет предопределенное исключение DUP_VAL_ON_INDEX. В этом случае вы откатитесь к точке сохранения, отменив лишь вставку.

```
DECLARE
 emp_id emp.empno%TYPE;
BEGIN
 …
 UPDATE emp SET … WHERE empno = emp_id;
 DELETE FROM emp WHERE …
 …
 SAVEPOINT do_insert;
 INSERT INTO emp VALUES (emp_id, …);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
ROLLBACK TO do_insert;
END;
```

При выполнении ROLLBACK TO все точки сохранения, отмеченные после указанной, стираются, а все изменения, сделанные после этой точки, отменяются. Однако сама точка сохранения, к которой вы возвращаетесь, не удаляется. Например, если вы последовательно отметите точки сохранения A, B C и D, а затем выполните ROLLBACK TO к точке B, то будут стерты лишь C и D.

ROLLBACK без аргументов, как и COMMIT, стирает все точки сохранения.

Если вы отмечаете точку сохранения в рекурсивной подпрограмме, то на каждом уровне рекурсивного спуска предложение SAVEPOINT будет создавать новые экземпляры точек сохранения. Однако ROLLBACK TO вернет вас лишь к самой последней из точек сохранения с данным именем.

Имена точек сохранения – это необъявляемые идентификаторы. Их можно повторно использовать внутри транзакции. При этом точка сохранения сдвигается со своей старой позиции в текущую точку в транзакции. Таким образом, откат к точке сохранения воздействует лишь на текущую часть вашей транзакции. Рассмотрим следующий пример:

```
…
BEGIN
 …
 SAVEPOINT my_point;
 UPDATE emp SET … WHERE empno = emp_id;
 …
 SAVEPOINT my_point; -- перемещает my_point в текущую точку
 INSERT INTO emp BALUES (emp_id, …);
 …
EXCEPTION
 WHEN OTHERS THEN
ROLLBACK TO my_point;
END;
```

По умолчанию число активных точек сохранения на сессию не может быть больше 5. АКТИВНАЯ ТОЧКА СОХРАНЕНИЯ – это точка, отмеченная после последней операции COMMIT или ROLLBACK. Вы или ваш АБД можете поднять этот лимит (вплоть до 255), увеличив значение параметра инициализации ORACLE с именем SAVEPOINTS.

Неявные точки сохранения

Перед выполнением каждого предложения INSERT, UPDATE и DELETE ORACLE создает неявную точку сохранения (недоступную вам). Если предложение сбивается, то выполняется откат к этой неявной точке. Обычно отменяется лишь сбившееся предложение SQL, а не вся транзакция.

Завершение транзакций

Хорошей практикой программирования является явное подтверждение или явный откат каждой транзакции. Если вы пренебрегаете явными операциями COMMIT или ROLLBACK, то окончательное состояние транзакции определяет сама СУБД.

Например, если ваш блок PL/SQL не содержит предложения COMMIT или ROLLBACK, то окончательное состояние вашей транзакции зависит от того, что вы делаете после выполнения этого блока:

Если вы выполняете операции определения данных, операции управления данными или операцию COMMIT, либо если вы выдаете команду EXIT, DISCONNECT или QUIT,
то ORACLE неявно подтверждает вашу транзакцию.

Если вы выдаете операцию ROLLBACK или аварийно снимаете сессию ,
то ORACLE выполняет откат транзакции.

Использование SET TRANSACTION

Умалчиваемым режимом для всех транзакций является согласованность данных по чтению НА УРОВНЕ ОПЕРАЦИИ. Т.е. запрос видит лишь то состояние данных, которое было перед началом его выполнения, плюс все изменения, которые внесены предыдущими операциями в текущей транзакции. Если во время запроса другие пользователи (транзакции) вносят изменения в эти же таблицы базы данных, то эти изменения будут видны лишь последующим, но не текущему, запросу.

Однако вы можете, выдав предложение SET TRANSACTION, установить режим согласованности данных по чтению НА УРОВНЕ ТРАНЗАКЦИИ. Это гарантирует, что запрос видит лишь то состояние данных, которое было подтверждено перед началом всей транзакции; однако при этом транзакция не должна вносить изменений в базу данных. Предложение SET TRANSACTION READ ONLY не принимает дополнительных параметров и имеет вид:

SET TRANSACTION READ ONLY;

Предложение SET TRANSACTION должно быть первым предложением SQL в транзакции и может появиться лишь один раз на транзакцию. Как уже сказано, в таком режиме транзакции все запросы, выдаваемые в
ней, видят то состояние данных, которое было подтверждено перед началом всей транзакции. Режим READ ONLY не влияет на других пользователей или другие транзакции.

В транзакции READ ONLY допускаются лишь предложения SELECT, COMMIT и ROLLBACK. Другие предложения, например, INSERT или DELETE, приводят к возбуждению исключения.

В течение транзакции READ ONLY все ее запросы обращаются к одному и тому же снимку базы данных, что обеспечивает многотабличное, многозапросное, согласованное по чтению представление данных для транзакции. Другие пользователи могут продолжать опрашивать или обновлять данные в обычном режиме.

Транзакция READ ONLY завершается выдачей COMMIT или ROLLBACK. В примере вы, как управляющий складом, используете транзакцию READ ONLY, чтобы собрать

цифры по продажам за день, прошедшую неделю и прошедший месяц. На эти цифры не могут повлиять другие пользователи, обновляющие базу данных во время транзакции.

```
DECLARE
 daily_sales REAL;
 weekly_sales REAL;
 monthly_sales REAL;
BEGIN
 SET TRANSACTION READ ONLY;

 SELECT SUM(amt) INTO daily_sales FROM sales
WHERE dte = SYSDATE;
 SELECT SUM(amt) INTO weekly_sales FROM sales
WHERE dte > SYSDATE – 7;
 SELECT SUM(amt) INTO monthly_sales FROM sales
WHERE dte > SYSDATE – 30;
 COMMIT; -- это просто сигнал об окончании транзакции, так как никаких изменений она не делает
 …
END;
```

Переопределение умалчиваемой блокировки

По умолчанию ORACLE автоматически блокирует для вас структуры данных. Однако вы можете запросить специфические блокировки по строкам или таблицам, если вам почему-либо выгодно изменить умалчиваемый режим блокировки. Явная блокировка позволяет вам разрешать или запрещать совместный доступ к таблице на время транзакции.

Использование FOR UPDATE

При объявлении курсора операции UPDATE или DELETE, вы должны использовать фразу FOR UPDATE, чтобы затребовать для этого курсора монопольные блокировки строк. Фраза FOR UPDATE, когда она присутствует, должна появляться в конце объявления курсора, как показывает следующий пример:

```
DECLARE
 CURSOR c1 IS SELECT empno, sal FROM emp
WHERE job = 'SALESMAN' AND comm > sal FOR UPDATE;
 …
BEGIN
 OPEN c1;
 LOOP
FETCH c1 INTO …
…
UPDATE emp SET sal = new_sal;
 END LOOP;
COMMIT;
CLOSE c1;
 …
```

Фраза FOR UPDATE указывает, что строки, выбираемые запросом, будут обновляться или удаляться, и блокирует все строки в активном множестве курсора. Это полезно, когда вы хотите, чтобы обновление базировалось на существующих значениях строк. В этом случае вам нужна гарантия, что строка не будет изменена другим пользователем, прежде

чем вы обновите ее. Все строки в активном множестве блокируются в момент открытия курсора, и разблокируются при выполнении COMMIT.

Извлечения между COMMIT'ами
Фраза FOR UPDATE запрашивает монопольную блокировку строк. Все строки в активном множестве блокируются
во время открытия курсора, а не во время их извлечения. Все строки освобождаются при завершении транзакции (COMMIT или ROLLBACK). Поэтому вы не можете извлекать строки из курсора, объявленного FOR UPDATE, после COMMIT. Если вы попытаетесь сделать это, возникнет исключение. Рассмотрим следующий цикл FOR, который собьется после десятой вставки:

```
DECLARE
 CURSOR c1 IS SELECT ename FROM emp FOR UPDATE OF sal;
 ctr NUMBER := 0;
BEGIN
 FOR emp_rec IN c1 LOOP – неявные операции FETCH
…
ctr := ctr + 1;
INSERT INTO temp VALUES (ctr, 'еще работает');
IF ctr >= 10 THEN
COMMIT; -- освобождает блокировки
END IF;
 END LOOP;
END;
```

Если вы собираетесь выполнять COMMIT, не закончив всех извлечений, то не используйте фразу FOR UPDATE.
Если фразу FOR UPDATE OF не использовать, то извлекаемые строки НЕ БЛОКИРОВАНЫ. Поэтому вы можете
получить несогласованные результаты, если другой пользователь изменит строку после того, как вы прочитали ее, но до того, как вы модифицировали ее.

Использование LOCK TABLE

Предложение LOCK TABLE позволяет вам заблокировать одну или несколько таблиц в указанном режиме, так что вы можете регулировать одновременный доступ к таблицам, поддерживая их целостность. Например, предложение, приведенное ниже, блокирует таблицу emp в режиме row share. Такой режим разрешает одновременный доступ к таблице, но запрещает другим пользователям блокировать всю таблицу для монопольного использования. Блокировка таблицы освобождается, когда ваша транзакция выдает COMMIT или ROLLBACK.

```
LOCK TABLE emp IN ROW SHARE MODE NOWAIT;
```

Режим блокировки определяет, какие другие блокировки могут быть применены к таблице. Например, несколько пользователей могут одновременно затребовать блокировки row share для одной и той же таблицы, но лишь один пользователь за раз может затребовать МОНОПОЛЬНУЮ (exclusive) блокировку. Пока один пользователь имеет монопольную блокировку таблицы, другие пользователи не могут изменять (INSERT, UPDATE или DELETE) строк этой таблице.

Необязательное ключевое слово NOWAIT указывает, что, если запрос LOCK TABLE не может быть удовлетворен (возможно, потому, что таблица уже заблокирована другим пользователем), то LOCK TABLE вернет управление пользователю, вместо того, чтобы ждать удовлетворения запроса. Если вы опустите ключевое слово NOWAIT, то ORACLE будет ждать освобождения таблицы; это ожидание не имеет устанавливаемого предела.

Лекция 12. Циклы или рекурсии, конкатенация полей из разных строк таблицы

WITH предоставляет способ написания вспомогательных операторов для использования в более больших запросах. Эти операторы, которые часто называются как Общие Табличные Выражения или CTE, могут быть задуманы как определяющие временные таблицы, которые существуют только для данного запроса. Каждый вспомогательный оператор в предложении WITH может затем быть подвергнут SELECT, INSERT, UPDATE или DELETE; а само предложение WITH прикрепляется к первичному оператору, которые также может быть одним из SELECT, INSERT, UPDATE или DELETE.

7.8.1. SELECT в WITH

Базовое значение SELECT в WITH должно разбивать сложные запросы на более простые части. Например:

```
WITH regional_sales AS (
      SELECT region, SUM(amount) AS total_sales
      FROM orders
      GROUP BY region
   ), top_regions AS (
      SELECT region
      FROM regional_sales
      WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
   )
SELECT region,
     product,
     SUM(quantity) AS product_units,
     SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

запрос показывает общие продажи каждого продукта только в регионах с высоким объёмом продаж. Преложение WITH определяет два вспомогательных оператора с именами regional_sales и top_regions, где вывод regional_sales используется в top_regions, а вывод top_regions используется в первичном запросе SELECT. Этот пример может быть написан без WITH, но тогда потребуется два уровня вложенных под-SELECT'ов. Гораздо легче следовать вышеописанному методу.

Необязательный модификатор RECURSIVE изменяет WITH с явного синтаксического комфорта в возможность, выполняющую такие вещи, которые невозможны в стандарте SQL. Используя RECURSIVE, запрос WITH может ссылаться на свой собственный вывод. Простой пример такого запроса состоит в суммировании целых чисел от 1 до 100:

```
WITH RECURSIVE t(n) AS (
   VALUES (1)
 UNION ALL
   SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Общая форма рекурсивного запроса WITH всегда *нерекурсивный термин*, затем UNION (или UNION ALL), затем *рекурсивный термин*, где только рекурсивный термин может содержать ссылку на свой собственный вывод запроса. Такой запрос выполняется так:

Рекурсивное выполнение запроса

1. Выполняется не-рекурсивный термин. Для UNION (но не для UNION ALL), отбрасываются дублирующиеся строки. Включаются все оставшиеся строки из результата рекурсивного запроса и также размещаются во временную *рабочую таблицу*.

2. Пока рабочая таблица не окажется пустой, повторяются следующие шаги:

    a. Выполняется рекурсивный термин, подстановка текущего содержимого рабочей таблицы для рекурсивной ссылки на саму себя. Для UNION (но не для UNION ALL), отбрасываются дублирующиеся строки и строки, которые дублируют любые строки в предыдущих результатах. Включаются все оставшиеся строки из результата рекурсивного запроса и также размещаются во временную *промежуточную таблицу*.

    b. Замещается содержимое рабочей таблицы на содержимое промежуточной таблицы, затем промежуточная таблица очищается.

Note: Строго говоря, данный процесс является нерекурсивной итерацией, но RECURSIVE является терминологическим выбором комитета по стандартам SQL.

В данном выше примере, рабочая таблица на каждом шаге содержит только одну строку, в которую на успешных шагах попадают значения от 1 до 100. На 100-м шаге, вывода нет из-за преложения WHERE и таким образом, запрос завершается.

Рекурсивные запросы обычно используются при работе с иерархическими данными или данными, которые имеют древовидную структуру. Полезный пример такого запроса состоит в нахождении всех прямых и непрямых составных частей продукта, представленных как таблица, которая показывает только состав продукта из частей или одной части из других:

```
WITH RECURSIVE included_parts(sub_part, part, quantity) AS (
   SELECT sub_part, part, quantity FROM parts WHERE part = 'our_product'
 UNION ALL
   SELECT p.sub_part, p.part, p.quantity
   FROM included_parts pr, parts p
   WHERE p.part = pr.sub_part
 )
SELECT sub_part, SUM(quantity) as total_quantity
```

```
FROM included_parts
GROUP BY sub_part
```

При работе с рекурсивными запросами, важно убедиться, что рекурсивная часть запроса не будет возвращать строки в определённый момент времени, в противном случае запрос станет бесконечным циклом. Иногда, используя UNION вместо UNION ALL этого можно добиться отбрасывая строки, которые дублируют выведенные ранее. Однако, часто цикл не выдаёт строк, которые дублируются полностью: это может быть необходимо, чтобы проверить только одно или несколько полей, чтобы увидеть, когда та же самая точка была достигнута ранее. Стандартный метод управления такими ситуациями состоит в подсчёте массива уже обработаных значений. Например, рассмотрим следующий запрос, который просматривает таблицу graph, используя поле link:

```
WITH RECURSIVE search_graph(id, link, data, depth) AS (
      SELECT g.id, g.link, g.data, 1
      FROM graph g
    UNION ALL
      SELECT g.id, g.link, g.data, sg.depth + 1
      FROM graph g, search_graph sg
      WHERE g.id = sg.link
)
SELECT * FROM search_graph;
```

Данный запрос зациклится, если отношения link содержат циклы. Поскольку мы требует вывода "depth", то простое изменение UNION ALL на UNION должно исключить зацикливание. Вместо этого, нам нужно определять достигли ли мы или нет той же строки снова, когда мы приходим к следующуму определённому пути поля link. Мы добавляем две колонки path и cycle в зацикливающийся запрос:

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS (
      SELECT g.id, g.link, g.data, 1,
        ARRAY[g.id],
        false
      FROM graph g
    UNION ALL
      SELECT g.id, g.link, g.data, sg.depth + 1,
        path || g.id,
        g.id = ANY(path)
      FROM graph g, search_graph sg
      WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

Для предотвращения зацикливания, значение массива часто полезно для представления самого себя как "path", чтобы понять, что была достигнула любая отдельная строка.

В большинстве случаев, когда для определения зацикливания нужна проверка более чем одного поля, используйте массив строк. Например, если вам необходимо сравнить поля f1 и f2:

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS (
      SELECT g.id, g.link, g.data, 1,
        ARRAY[ROW(g.f1, g.f2)],
        false
```

```
      FROM graph g
    UNION ALL
      SELECT g.id, g.link, g.data, sg.depth + 1,
        path || ROW(g.f1, g.f2),
        ROW(g.f1, g.f2) = ANY(path)
      FROM graph g, search_graph sg
      WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

Tip: Опускайте синтаксис ROW() в общем случае, где для определения зацикливания необходимо проверять только одно поле. Это позволит использовать простой массив, вместо массива с составным типом, что даёт преимущество в производительности.

Tip: Алгоритм выполнения рекурсивного запроса выводит результаты в порядке нахождения первых подходящих значений. Вы можете просматривать эти результаты в порядке вложенности, указав во внешнем запросе ORDER BY для колнки "path".

Полезная уловка для тестирования запросов, когда вы не знаете точно, может ли случиться зацикливание, состоит в добавлении LIMIT к родительскому запросу. Например, данный запрос без LIMIT зацикливался бы всегда:

```
WITH RECURSIVE t(n) AS (
   SELECT 1
 UNION ALL
   SELECT n+1 FROM t
)
SELECT n FROM t LIMIT 100;
```

Это работает, потому что текущая реализация PostgreSQL производит дальнейшее выполнение только в зависимости от того, как много строк запроса WITH фактически было получено родительским запросом. Использование данной уловки в продуктивных решениях не рекомендуентся, потому что другие СУБД могут работать по-другому. Также, такое обычно не работает, если вы делаете внешнюю сортировку результатов рекурсивного запроса или соединяете их с некоторой другой таблицей, потому что в таких случаях внешний запрос обычно в любом случае будет пытаться получить вывод от всех запросов WITH.

Полезное свойство запросов WITH заключается в том, что они выполняются только один раз при запуске родительского запроса, даже если они они вызываются из родительского запроса или вложеных WITH запросов более одного раза. Таким образом, затратные по ресусам вычисления, которые необходимы во многих случаях, могут быть размещены внутри WITH запроса, чтобы избежать избыточной обработки. Другое возможное применение состоит в предотвращении нежелательного многократного выполнения функций, которое может вызвать посторонние эффекты. Однако, другая сторона монеты в том, что оптимизатор мало поможет в применении ограничений из родительского запроса к WITH запросу, как в случае обычного подзапроса. WITH запрос будет обычно выполнен как описано, без подавления строк, которые впоследствие могут быть отброшены родительским запросом. (Но, как говорилось выше, выполнение можно остановить раньше, если данный запрос будет ограничен указанием количества строк.)