I. Введение

Wednesday, September 5, 2018 11:39

Преподаватель - *Фомичева Татьяна Генриховна*

Введение

Дисциплина посвящена изучению методов проектирования **баз данных (БД)** и реализации прикладного **программного обеспечения (ПО)** на базе **систем управления базами данных (СУБД)**

База данных - это именнованная совокупность данных, хранящихся во внешней памяти и обладающая таким свойствами, как:

- Интегрированность, направленная на решение общих задач
- Структурированность, отражающая часть реального мира
- Взаимосвязанность
- Независимость описания данных от программ их обработки

Система управления базами данных (СУБД) представляет собой программную систему, которая решает следующие задачи:

- Обеспечивает пользователей языковыми средствами описания данных и манипулирования ими.
  Часто СУБД имеют встроенный язык программирования - так, Access использует Visual Basic. Часто используются и внешние языки - C++, C#, Java.
- Поддерживает логические модели данных
- Обеспечивает поддержку манипулирования данными на логическом уровне, т.е. выполнение таких операций, как **выбор, вставка, обновление, удаление**, с одновременным отображением этих операций на физическом уровне
- Обеспечивает защиту, поддерживает целостность и непротиворечивость данных.
  - Защита от аппаратных сбоев. В системе должны быть предусмотрены возможности сохранить существующие данные или вернуться к предыдущему варианту
  - Защита от несанкционированного доступа
  - Защита от неквалифицированных действий пользователя

В настоящий момент на рынке существуют около 200 различных СУБД, более 100 из них относятся к реляционным.

Банк данных - это совокупность БД, СУБД и аппаратных и организационных средств, поддерживающщих их функционирование. Часто используется термин **автоматизированная информационная система (АИС)**

Модели данных, которые поддерживает СУБД, делят на:

- Сетевые. В основе модели - произвольный граф
- Иерархические. В основе модели - дерево
- Реляционные
- Объектно-ориентированные
- Многомерные

Каждая СУБД поддерживает какую-нибудь одну, но иногда и одновременно несколько моделей.

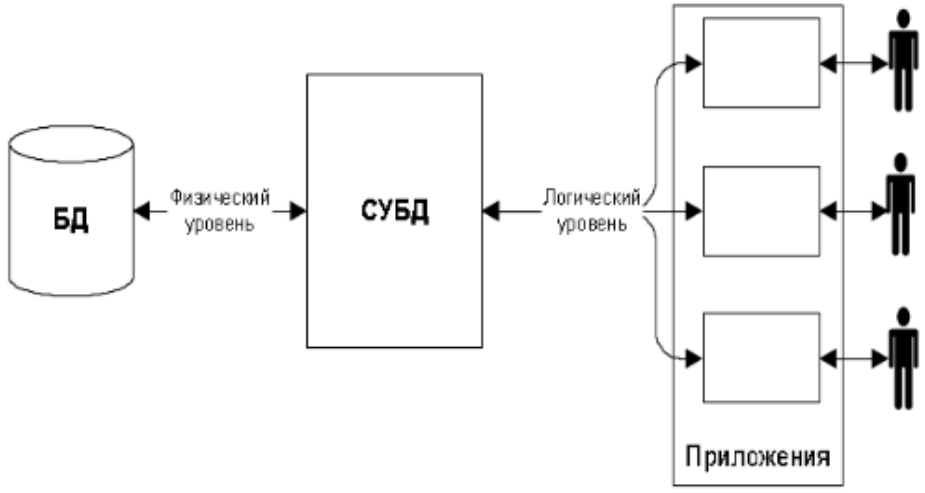


Схема взаимодействия СУБД с БД при извлечение записи, хранимой в БД

1. Запрос записи
2. Запрос страницы
3. Дисковая операция ввода/вывода
4. Чтение данных с диска
5. Возвращение страницы
6. Возвращение записи

Часто используемые структуры хранения:

- **Индексы** (в частности Б-деревья)
- **Хеширования** (в т.ч. расширяемое)
- **Цепочка указателей** (родительско-дочерние структуры)

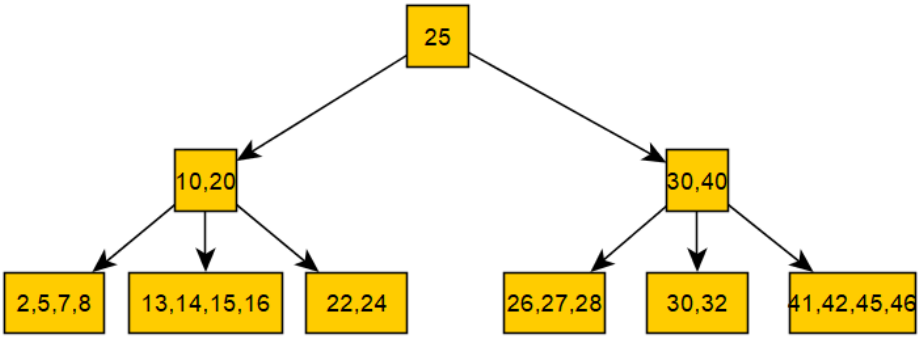
Идеальной структуры хранения, оптимальной для любых задач, не

существует.

Б-дерево

- Каждая страница содержит не более $2n$ элементов (ключей)
- Каждая страница содержит не менее n элементов, кроме корневой
- Каждая страница либо лист, либо имеет $m + 1$ потомков
- Все листья находятся на одном уровне

Пример



II. Реляционные СУБД

Wednesday, September 5, 2018 12:40

Модель предметной области

Совокупность объектов, выделенных из предметной области, их свойств, необходимых для решения определённой задачи и связей между объектами называется **моделью предметной области (МПО)**.
В зависимости от типа решаемой задачи важны различные свойства одного и того же объекта.

Связь между объектами характеризуется направлением и количеством экземпляров объектов, которые могут в ней участвовать

Вид связи	**Пример**
Один-к-одному (1:1)	"Секретарь представляет кафедру", "Кафедра
Один-к-многим (1:n)	"Секретарь регистрирует абитуриента"
Многие-к-одному (n:1)	"Абитуриет регистрируется у секретаря"
Многие-к-многим (n:m)	"Студент сдает предмет"

Концептуальная модель - это представление МПО в терминах **модели данных**.
Процесс построения концептуальной модели называется **логическим проектированием базы данных.**
Концептуальная модель средствами СУБД отображается в соответствующие структуры **физической базы данных**

Реляционная модель данных (модель Кодда)

Relation (англ.) - отношение.
Пусть даны N множеств $D_1, D_2, \dots, D_N$.
Тогда R есть **отношение** над этими множествами, если R есть множество упорядоченых $m-$ кортежей вида $< d_1, d_2, \dots, d_n >$, где $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_N$
$D_1, D_2, \dots, D_n$ называеются **доменами** отношения R

Схема отношения: *Студент (ФИО, Дата_рождения, Курс, Специальность)*

ФИО	**Дата_рождения**	**Курс**	**Специальность**
Иванов И.И.	01.09.93	3	История
Петров П.П.			
Сидор С.С.			

Количество столбцов - **степень**, количество строк (кортежей) - **мощность**

Основные определения:

- **Реляционная модель** - набор отношений (двумерных таблиц), на которые накладываются некоторые ограничения. Такой набор может быть использован для хранения сведений об объектах реального мира и моделирования связей между ними.
- Каждая таблица (отношение) имеет **имя** и состоит из множества строк **(кортежей)** и **столбцов**
- Столбцы также имеют имена. **Имена** столбцов - это **атрибуты** отношения
- Число столбцов или атрибутов отношения называют его **степенью**, а текущее число строк - **мощностью.**
- Список атрибутов называют **схемой отношения**

Ограничения реляционной модели

- Все строки таблицы (или все кортежи) должны иметь одинаковую структуру, т.е. одно и то же количество атрибутов
- Значения каждого атрибута должны быть взяты из некоторого фиксированного множества значений (домена)
- Никакие две строки не могут совпадать. Каждая строка таблицы должна иметь свое значение **первичного ключа**
- В целях непротиворечивости базы данных должна соблюдаться ссылочная целостность для внешних ключей

Первичный ключ - атрибут или набор атрибутов, который может быть использован для однозначной идентификации конкретной строки. Первичный ключ не должен иметь лишних атрибутов.
Если отношение имеет несколько потенциальных первичных ключей, то выбирают тот из них, который короче. Другие потенциальные ключи называются **ключами-кандидатами**

Типы отношений реляционной БД

- **Объектное отношение** - хранит данных об экземлярах объекта предметной области
- **Связное отношение** - содержит атрибуты, характеризующие связь между объектами. Обязательно содержит первичные ключи объектных отношений, участвующих в связи, а также атрибуты, которые функционально зависят от этой связи.

Ограничения, накладываемые реляционной моделью на внешние связи
Атрибут отношения, который является первичным ключом другого отношения, называется **внешним** или **вторичным** ключом. Каждому внешнему ключу должна соответствовать строка какого-либо объектного отношения

Требования атомарности значений атрибутов
Отношения должны быть **нормализованы**, т.е. они должны находится в какой-либо из **нормальных форм** (например, 1НФ)
Это означает, что каждый атрибут отношения должен быть простым, т.е. содержать атомарные, неделимые значения. Значения атрибутов не могут быть ни списками значений, ни именами отношений, которые бы заменили собой списки значений.
Это очень жестко ограничение реляционной модели стало одной из основных причин появления постреляционных СУБД.

Связь между таблицами реляционной модели

- Связь осуществляется по общим полям (атрибутам) таблиц и может быть 1:1 или 1:n
- Существует понятие **главной (родительской)** и **подчинённой (дочерней)**
- Поле связи должно быть обязательно первичным ключом клавной таблицы
- Поля связи в таблицах не обязательно должны иметь одинаковые имена, но значения полей должны быть взяты из одного и того же множества (т.е. одинаковый тип, размер и т.п.)
- Возможно использование для связи не только одного поля, но и совокупности полей

Случаи, когда СУБД осуществляет автоматический контроль данных на непротиворечивость

- При добавлении данных в подчиненную таблицу.
  Нельзя добавить строку, в которой поле связи содержит значение, отсутствующее в главной таблице. При добавлении данных в главную таблицу контроль ссылочной целостности не требуется.
- При удалении данных из главной таблицы.
  Нельзя удалить из главное таблицы данные, связь с которыми есть в подчинённой таблице. Для удаления записи и всего, что с ней связано, есть **каскадное удаление**
- При модификации значения поля связи как в главной, так и в подчинённой таблице.
  В подчиненной таблице значение поля связи можно изменить на другое значение, но только случае, естли новое значение есть в главной таблице. Изменить значение поля связи в главной таблице нельзя, если новое значение есть в подчинённое таблице. Нужно использовать **каскадное удаление данных**

Составление реляционной модели
Реляционная модель определяет не только принципы хранения данных, но и способы их обработки
Реляционная модели состоит из трёх частей:

- Структурной
- Целостной
- Манипуляционной

В основу манипуляционной части модели положена теория множеств.

Набор операций манипуляционной составляющей реляционной модели

- Теоретико-множественные операции
  - **Объединение**
    $R_3 = R_1 \cup R_2$
    Результат - отношение с заголовоком R_1 и R_2 из кортежей, которые принадлежат или R_1, или R_2, или обоим.
  - **Пересечение**
    $R_3 = R_1 \cap R_2$
    Результат - отношение с заголовоком R_1 и R_2 из кортежей, которые принадлежат и R_1, и R_2
  - **Разность**
    $R_3 = R_1 \backslash R_2$
    Результат - отношение с заголовоком R_1 и R_2 из кортежей, которые принадлежат R_1, но не R_2
  - **Декартововo произведение**
    $R_3 = R_1 \times R_2$
    Заголовки
    $R_1(A_1, \dots, A_n)$ с телом $(a_1, \dots a_n)$
    $R_2(B_1, \dots, B_n)$ с телом $(b_1, \dots b_n)$
    $R_3(A_1, \dots, A_n, B_1, \dots, B_n)$ с телом $(a_1, \dots a_n, b_1, \dots, b_n)$
- Специальные реляционные операции
  - **Селекция**
    $R_2 = \sigma_F(R_1)$, где $F -$ условие
    Операция выполняется над одним отношением. Результирующет отношение R_2 содержит строки отношениz R_1, выбранные по некоторому заданному критерию.
  - **Проекция**
    $R_2 = \pi_{i_1, i_2, \dots, i_n}(R_1)$
    Операция выполняется над одним отношением на некоторые его атрибуты. Результирующее отношение включает ту часть атрибутов исходного отношения, на которые выполняется проекция.
  - **Соединение**
    $R_3 = R_1 \rhd i\theta j \lhd R_2 = \sigma_{i\theta j}(R_1 \times R_2)$
    Где $R_1, R_2 -$ исходные отношения, $R_3 -$ результирующее отношение, $i -$ атрибут отношения R_1, $j -$ атрибут отношения R_2, $\theta -$ оператор сравнения (=, >, <, >=, <=, <>)
    - **Естественное соединение**
      $R_3 = R_1 \rhd \lhd R_2$

$R_1(A_1, \dots, A_n, X_1, \dots, X_m); R_2(X_1, \dots, X_m, B_1, \dots, B_k)$
$R_3(A_1, \dots, A_n, X_1, \dots, X_m, B_1, \dots, B_k)$
Отношения R_1 и R_2 должны иметь в своем составе одинаковые атрибуты. Отношение R_3 включает все атрибуты обоих отношений, в которых значения равны.

- **Деление**
  $R_3 = R_1 \div R_2$
  $R_1(A_1, \dots, A_n, B_1, \dots, B_m)$ с телом $(a_1, \dots, a_n, b_1, \dots, b_m)$
  $R_2(B_1, \dots, B_n)$ с телом $(b_1, \dots, b_n)$
  $R_3(A_1, \dots, A_m)$ с телом $(a_1, \dots, a_m)$, таким, что
  $\forall(b_1, \dots, b_n) \in R_2: \exists(a_1, \dots, a_n, b_1, \dots, b_m) \in R_1$
  Делимое должно содержать подмножество атрибутов отношения-делителя. Результирующее отношении содержит только те атрибуты делимого, которые отсутствуют в делителе. В него включаются только те кортежи, для которых соответствующие значения из делимого включают все значения из делителя.

В состав реляционной алгебры включаются также операции **переименования** и **присваивания** аргументов

Свойства

- Операции объединения, пересечения и разности имеют смысл не для любых множеств. Поскольку результат каждой из них - отношение, то операндами должны быть отношения с совместимыми схемами. Отношения-операнды должны иметь:
  - Одинаковую степень
  - Одинаковые типы соответствующих атрибутов
  - Имена
- Для декартова проиведения надо, чтобы заголовки столбцов были разными
- Все теоретико-множественные операции ассоциативны
- Все теоретико-множественные операции, кроме разности, коммутативны

III. Метод декомпозиции

Wednesday, September 12, 2018 12:52

Упорядочение данных

Сортировка - физическое упорядочение данных. Результат сортировки - новая таблица.
Индексация - логическое упорядочение. Создается таблица индексов, порядок хранения данных во внешней памяти не изменяется

Сортировка производится по ключу. Если ключ состоит из нескольких полей, то сортировка по следующим полям производится при совпадении текущего.

Индекс может быть **простым** и **составным**. Наличие составного индекса для поиска по нескольким полям ускоряет поиск, но выбор индекса может представлять сложность по БД. В некоторых СУБД к каждому запросу явно указывается, какой индекс использовать.
Имея индекс из нескольких полей, можно осуществлять поиск и по части полей. Условие - поля должны идти друг за другом, и только последнее условие поиска должно быть неравенством.

Индекс всегда несёт накладные расходы, поэтому при небольших размерах таблиц или малом количестве запросов создание индекса не имеет смысла.

Цели проектирования баз данных

- Обеспечение возможности хранения в базе данных **всех** необходимых данных
- Исключение избыточности данных
- Сведение к минимуму количества хранимых в БД отношений
- Нормализация отношения для упрощения проблем, связанных с модификацией данных.

Методы проектирования:

- Метод декомпозиции
- Метод "сущность-связь" (ER-метод)
- Метод нормальных форм

Метод декомпозии

Декомпозиция - разделение отношения. Используется чтобы избежать избыточное дублирование.

Пример. Постановка задачи создания БД "Библиотека"

Пусть требуется создать БД для библиотекаря. В БД должны хранится сведения о книгах, о читателях и о книгах на руках у читателя.

- Сведения о книга состоят из *Шифра, Названия, Автора, Года издания, Количества Экземляров*
- Сведения о читателе - *Фамилия и Инициалы, Номер Билета, Номер Телефона*
- В сведениях о книгах на руках у читателя - *Дата* закрепления книги за читателем.

Можно попытаться поместить все эти данные в одно отношение - **универсальное отношение**.

Билет	Фио	Телефон	Шифр	Название	Автор	Год	Экз	Дата

Первые 3 столбца - про читателя, следующие 5 - про книги, последний - про дату. Такое отношение имеет следующие проблемы:

- Чтобы изменить, например, номер телефона у читателя, нужно провести количество операций, равное количеству взятых книг
- Пусть читатель отдал книгу. В этом случае есть риск удалить информацию о книге в целом. Решение - убрать часть таблицы про читателя и оставить сведения о книге, если сведений о книге больше нет и если сведений о читателе больше нет.
- Если читатель, который ничего не брал, возьмет книгу, которую никто не брал, нужно соединять строки.

Такие вариации при выполнении одинаковых операций называются **аномалиями** - **аномалии обновления, добавления** и **удаления**

Функциональная зависимость

Пусть есть два атрибута - A, B. Если в любой момент времени каждому значению A соответствует не более чем одно значение B, то B **функционально зависимо** от A: $A \rightarrow B$

A и B могут быть как простыми, так и составными.

Для предыдущего примера зависимости:

1. Шифр → Автор
2. Шифр → Название
3. Шифр → Год
4. Шифр → Экземляры
5. Билет → ФИО
6. Билет → Телефон
7. Шифр, Билет → Дата

Шифр однозначно определяет книгу, а номер билета - читателя. Но, например, название не обязательно определяет шифр, потому что могут быть разные издания книги.

Нормальная форма Бойса-Кодда (НФБК)

Определение находится в **НФБК**, если каждый детерминант отношения

является его возможным ключом. При этом A представляет собой **детерминант** B, если B функционально зависит от A и A не зависит от любого подмножества A.
НФБК позволяет избавиться от большей части (но не всех!) аномалий.

В примере ключ - только совокупность Шифр, Билет. Значит, отношение не находится В НФБК и его нужно подвергнуть декомпозиции.

Декомпозиция без потерь при естественном соединении
Пусть существует отношение $R(A, B, C, D, E, \dots)$
Атрибут $A -$ первичный ключ отношения, от него функционально зависят все остальные атрибуты. Кроме того, существует ФЗ $C \to D$, что мешает отношению находится в НФБК.
В таком случае следует разбить отношение R на два: $R_1\ (C, D)$ и $R_2(A, B, C, E, \dots)$
Отношение R_1 является **проекцией** отношения R. Атрибуты функциональной зависимости, на которую выполняется проекция, переходят в новое отношение.

Избыточные ФЗ
Функциональная зависимость называется избыточной, если не заключает в себе информацию, которую нельзя получить из других ФЗ. Избыточная ФЗ может и должна быть удалена из набора ФЗ.
Найти и удалить ФЗ можно с помощью правил, которые называются **правилами вывода:**

- **Транзитивность**
  $\exists A \to B, \exists B \to C \Rightarrow A \to C -$ избычтоно
  Нестандартный случай - циклическая зависимость. В таком случае нужно отключить любую зависимость
- **Объединение**
  $\exists A \to B, \exists A \to C \Rightarrow \exists A \to B, C$. Таким образом можно сократить число функциональных зависимостей от одного атрибута.
- **Добавление**
  $\exists A \to B \Rightarrow A, Z \to B$ или $A, X \to B, X$ являются корректными, но избыточными
- **Декомпозиция**
  $\exists A \to B, C \Rightarrow$ можно заменить на $A \to B, A \to C$
- **Псевдотранзитивность**
  $\exists A \to B, \exists B, C \to X \Rightarrow \exists A, C \to X$. Новая зависимость называется псевдотранзитивной и является избыточной.

Устранение избыточных ФЗ в примере:

1. Шифр → Автор, Название, Год, Экземляр
2. Билет → ФИО, Телефон
3. Шифр, Билет → Дата

Минимальное покрытие - набор не избыточных ФЗ, полученный путём удаления из исходного набора всех избыточных ФЗ с помощью правил вывода. Может существовать и несколько равноправных минимальных покрытий.

Декомпозиция примера

Есть два кандидата на проекцию - ФЗ 1 и ФЗ 2. Можно выбрать первую и получить два отношения:

- R_1 с атрибутам Шифр, Автор, Название, Год, Экземляр, между которыми существует единственная ФЗ
  Шифр → Автор, Название, Год, Экземляр
- Отношение R_2 с атрибутами Билет, ФИО, Телефон, Шифр, Дата
  Билет → ФИО, Телефон
  Шифр, Билет → Дата

Отношение R_2 мешает НФБК, значит, следующую декомпозицию нужноо выполнять на него:

- R_3 с атрибутами Билет, ФИО, Телефон и одной ФЗ
  Билет → ФИО, Телефон
- R_4 с атрибутами Шифр, Билет, Дата и одной ФЗ
  Шифр, Билет → Дата

Отношение R_1 можно назвать "Книга", R_3 — "Читатель", R_4 — "Читает".

Выбор ФЗ для первой проекции

Следует избегать выбора ФЗ, правая часть которой полностью или частично является детерминантом другой ФЗ. Т.е. ищутся цепочки вида A → B → C и выбирается крайняя правая зависимость.

Причина - если попробовать, имея цепочку A → B → C, выполнить проекуцию на ФЗ A → B, то получим отношения $R_1(A, B), R_2(A, C)$. Оба эти отношения находятся в НФБК, но теряется зависимость $B \rightarrow C$.

Последовательность шагов метода декомпозиции

1. Разработать универсальное отношение
2. Опеределить все ФЗ между атрибутами отношения
3. Исключить избыточные ФЗ и получить минимальное покрытия
4. Определить, находится ли отношение в НФБК
5. Если да, то конец
6. Если нет, то выбрать ФЗ для проекции и разложить отношение на два
7. Повторить шаг 4 для каждого нового полученного при разложении отношения
   Процесс декомпозиции завершается, когда окажется, что все полученные отношения находятся в НФБК
8. Полученный набор отношений проанализировать на предмет смысловой связности атрибутов, хранящихся в отношения
9. Проверить, поддерживает ли полученный набор отношений

предполагаемые запросы и операции обновления. Анализ запросов может потребовать добавления в БД новых атрибутов, связей между отношениями и даже новых, обычно связных, отношений

10. Проконтролировать отсутствие избыточных отношений
11. Проконтролировать отсутствие избыточных ФЗ.

Проверка отсутствия избыточных ФЗ

- Одна и та же ФЗ не должна присутствовать более чем в одном отношении
- Набор ФЗ, распределённых по раззным отношения, должен совпадать с минимальным покрытием.

Если последнее не так, то нужно показать, что перейти к минимальному покрытию от итогового набора ФЗ или наборот можно, последовательно применяя правила вывода. Если осуществить переход не удается, то в процессе проектирования была допущена ошибка.

Избыточные отношение

Отношение является избыточным в двух случаях:

- Если все его атрибуты присутствуют в другом отношении
  Например: $R_1(A, B), R_2(B, C, Y, Z), R_3(A, B, D)$
  Отношение R_1 в этом наборе является избыточным, поскольку оба его атрибута содержаться в отношении R_3
- Если все его атрибуты присутствуют в отношении, которое может получено из других отношений БД с помощью последовательности операций естественного соединения над ними.
  Пример: есть имеем БД из пяти отношений:
  $R_1(A, C, X), R_2(D, K, F), R_3(D, E, G, H), R_4(A, B, D), R_5(A, B, E, G)$
  Если применить операцию соединения к отношениям R_3 и R_4, то получим отношение $R_6(A, B, D, E, G, H)$. Поскольку в этом отношении содержаться все атриуты отношения R_5, то R_5 является избыточным отношений.
  Если применить ту же операцию к паре R_3, R_5, то избыточным окажется R_4

Пример

- **Избыточность ФЗ**
  В отношении Книга присутствует одна ФЗ -
  `Шифр → Авт, Назв, Год, Экз`
  В отношении Читатель одна ФЗ -
  `Билет → ФИО, Тел`
  В отношении Читает одна ФЗ
  `Шифр, Билет → Дата`
  Каждая ФЗ присутствует только в одном отношении, а набор из трёх ФЗ, распределённый по трём отношения, совпадает с исходным минимальным покрытием.

- **Избыточность отношений**
  Избыточных отношений нет, так как очевидно, что атрибуты ни одного из отношений полностью не присутствуют в другом. Выполнять операции соединения имеет смысл, когда атрибут, находящийся в одном отношении, находится и в другом. Это относится к атрибутам правых частей ФЗ. Атрибуты левых частей могут присутствовать в нескольких отношений при наличии связи между отношениями по этим атрибутам

Анализ набора отношений БД на предмет смысловой связности и соответствие запросам

- Смыслоовая связность атрибутов, составляющих каждое из отношений, не нарушена.
  Пример
  В отношении Читатель присутствуют атрибуты, характеризующие читателя - ФИО. Билет, Тел. Отношение Книга, содержит только атрибуты, характеризующие книгу, а Читает является связным отношением.
- Заключение о соответствии полученного набора предполагаемым запосам можно дать только, имея перечень запросов.
  Пример
  - Кто из читателей является читателем библиотеки уже более 10 лет
  - Кто из читателей брал книги в предыдущем месяце

  На эти запросы в существующей БД ответить невозможно, нужна коррекция атрибутов

Действия, которые выполняются на стадии реализации БД

- Средствами выбранной для реализации СУБД вводятся описания (схемы) всех составляющих базу данных отношений
- Разрабатываются интерфейсы пользователей разных категорий. Сюда входит разработка экранных форм для ввода и отображения данных, удобных экранных способов доступа к хранящимся в базе данным, создание выходных форм (отчётов), обеспечивающих максимальную информативность и удобство восприятия.
- Разрабатывается ПО прикладной системы, реализующее все необходимые операции с БД для всех категорий пользователей
- БД заполняется отладочными данными
- Производится отладка и тестирование разработанной прикладной системы

IV. Entity-Relation Model

Wednesday, September 26, 2018 12:20

ER-модель

Entity - сущность, Relationship - связь
Является **семантической**, а не реляционной моделью. Моделирование предметное области базируется на использовании **графических диаграмм**, включающих небольшое число разнородных компонентов (ER-диаграмм)

Основные понятия ER-модели

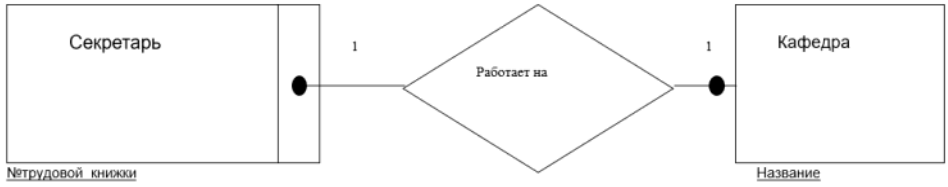
- **Сущность** - реальный или представляемый объект, имеющий значение при решении задачи, информация о котором должна хранится и быть доступной
  - Именуется, как правило, существительным
  - Имеет экземпляры, каждый из которых должен быть однозначно идентифицируем
  - На диаграммах изображается прямоугольником с её именем
- **Атрибут** сущности - свойство сущности, важное для решения задачи
  - На диаграмме перечисляются внутри сущности (нотация Чена и Баркера) или под прямоугольником, изображающим сущность (нотация Хау)
  - Допустимо изображение на диаграммах не всех атрибутов сущности, а только её идентификатора
- **Связь** - ассоциация (зависимость), устанавливаемая между двумя или более сущностями.
  - Именуется обычно глаголом
  - Чаще всего используются **бинарные** связи, то есть связи между двумя разными сущностями или сущностью и ею же самой. В последнем случае связь называется **рекурсивной**
  - В связи выделяются два конца, для каждого конца указывается
    - **Имя**
    - **Степень**, то есть сколько экземпляров сущности существует в связи
    - **Класс принадлежности сущности связи** (обязательная или необязательная)
  - На диаграммах связи изображают по-разному в разных нотациях.

Изображение связи (нотация Хау)

- Связь изображается ромбом
- Имя связи помещается внутри ромба
- Степень связи изображается цифрой 1, если в связи участвует один экземпляр сущности, и буквам, например: m, n, p, если в связи

участвуют несколько элементов
- Обязательность или необязательность связи обозначается точкой на линии связи внутри сущности или снаружи соответственно

Пример

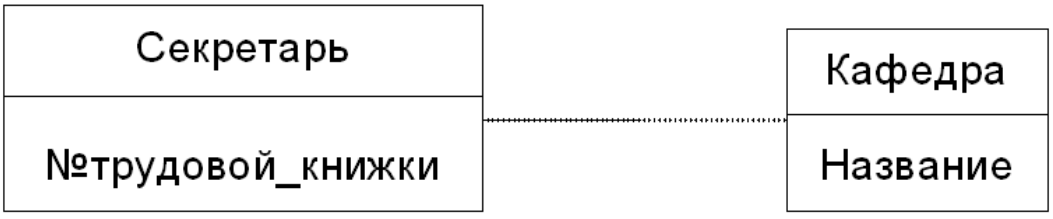


Связь между секретарем кафедры и кафедрой, является связью 1:1. Эта связь необязательна со стороны сущности «кафедра» и обязательна со стороны сущности «секретарь» , т.к. кафедра может иметь только 1 секретаря, а временно может и не иметь секретаря, но секретарь кафедры обязательно должен работать на какой-то, причем только на одной, кафедре.

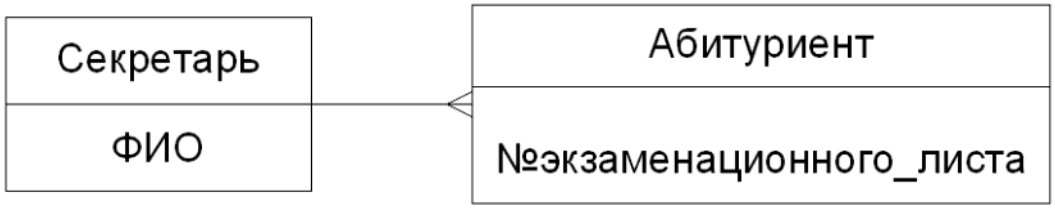
Изображение связи (нотация Баркера)

- Связь изображается линией, сплошной или пунктировной до середины, в зависимости от того, является она обязательной или нет
  - Сплошная линия означает обязательность участия в связи каждого экземпляра сущности, со стороны которой эта линия проведена
  - Пунктирная - необязательность
- Множественная связь имеет вид "вороньей лапы"

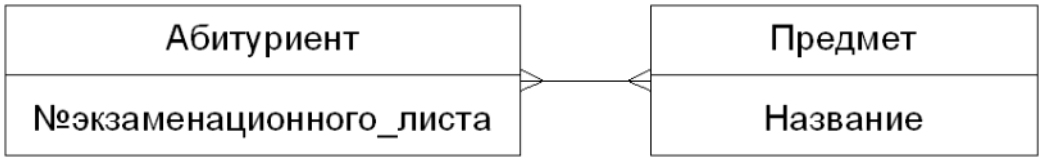
Пример



Пример связи 1:n, обязательной для обеих сторон



Пример связи n:m, обязательной для обеих сторон



Связь является обязательной с обеих сторон, поскольку каждый абитуриент сдает экзамен хотя бы по одному предмету, а каждый предмет сдает хотя бы один человек.

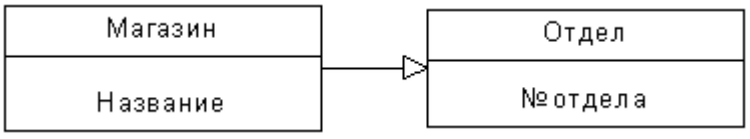
Пример рекурсивной связи

Связь 1:n, поскольку у каждого человека может быть несколько детей, но отец у каждого один. Связь обязательна со стороны n и необязательна со стороны 1, так как человек не обязательно является отцом, но отец есть у каждого человека.

Зависимая сущность

Зависиая сущность не может существовать сама по себе, а только относительно той сущности, от которой она зависит.
Зависимая линия связи представляется стрелкой, указывающей в сторону зависимой сущности (нотация Чена)

Пример 1. Сущность «задача» зависит от сущности «проект», если каждая задача на предприятии решается в рамках какого-нибудь проекта
Пример 2. Сущность «отдел в магазине» зависит от сущности «магазин», поскольку вне магазина отдел существовать не может.



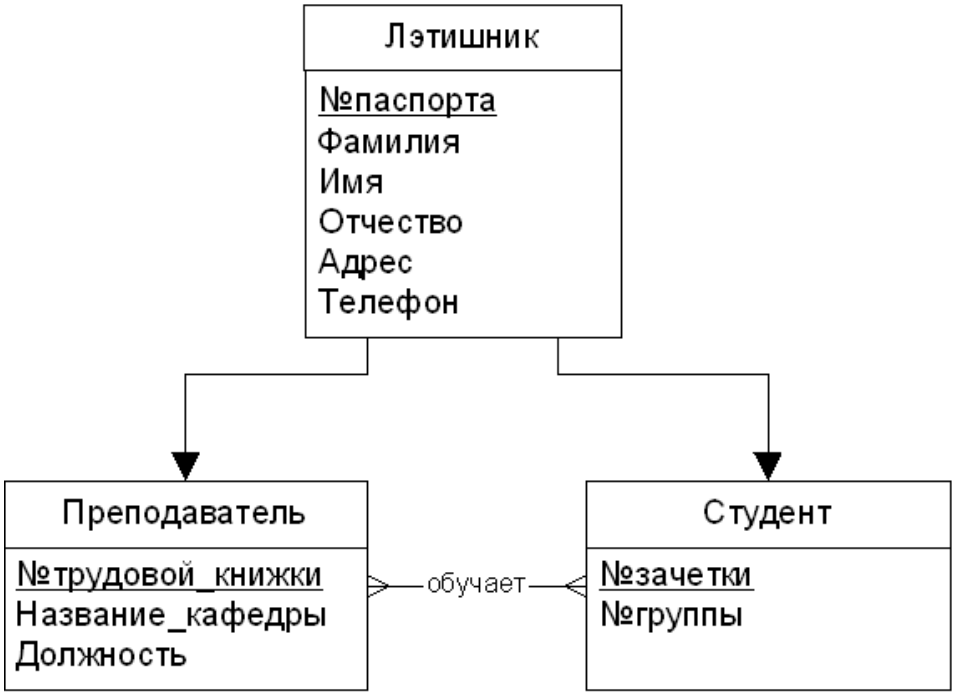
Общая сущность и её категории (понятия модели Чена)

- У Баркера понятию общей сущности соответствует **супертип**, а категории **- подтип,** у Хау - соответственно **сущность** и **роль**.
- Общая сущность обязательно имеет однозначный идентификатор. Категории могут не иметь дополнительного идентификатора, отличного от идентификатора общей сущности.
- Между категориями может существовать связь

Пример использования ER-модели категорий сущности, имеющих специфические по сравнению с общей сущностью атрибуты
Общая сущность - "Лэтишник"
Категории - "Студент" и "Преподаватель"

Сущности "Студент" и "Преподаватель":

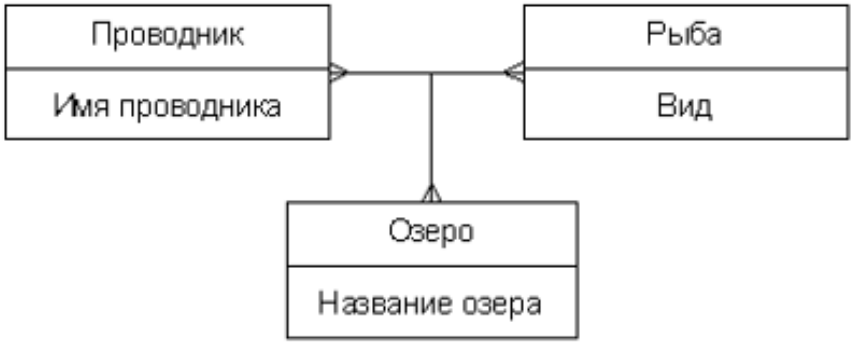
- Имеют некоторые общие атрибуты, например, номер паспорта, фамилию, имя, отчество, адрес и телефон,
- Чем-то отличаются друг от друга
  - Сущность «Студент», например, имеет атрибуты: «номер зачетки», «номер группы»,
  - Сущность «Преподаватель» имеет атрибуты «номер трудовой книжки», «название кафедры», на которой он работает, и «должность».



Пример связи третьего порядка

Проводник обслуживает несколько озёр, каждое озеро обслуживается несколькими проводниками, в каждом озере водится несколько видов рыбы, и каждый вид рыбы может водится в разных озерах, каждый проводник предпочитает ловить определённые виды рыбы

В ситуации, когда проводнику безразлично, в каком озере ловить тот или иной вид рыбы, достаточно трёх бинарных связей, но если проводник предпочитает ловить рыбу вида 1 в озере А, а рыбу вида 2 и 3 - в озере В, то для описания такой ситуации потребуется трёхсторонняя связь



Если в связи одновременно участвуют три и более сущностей, такая связь

называется **n-сторонней,** где n - количество участвующих в связи сущностей.

Правила генерации отношений из ER-диаграмм (правило 1)
Если:

- Степень бинарной связи равна 1:1
- Класс принадлежности обеих сущностей является обязательным

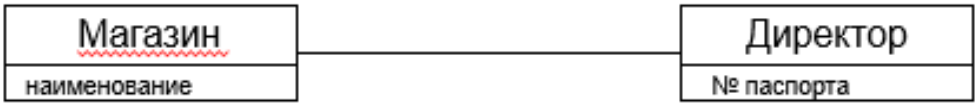
То:

- Требуется только 1 отношение
- Его первичным ключом может быть ключ любой из двух сущностей.

Пример. Имеем две сущности -

- "Магазин" с атрибутами "наименование", "факс", "телефон", "адрес", "специализация"
- "Директор" с атрибутами "Номер паспорта", "ФИО".

Каждый магазин обязательно имеет директора, а человек, являющийся директором, может быть директором только одного отношения



Получим отношение Магазин (наименование, факс, телефон, адрес, специализация, номер паспорта директора, ФИО директора)

Правила генерации отношений из ER-диаграмм (правило 2)
Если:

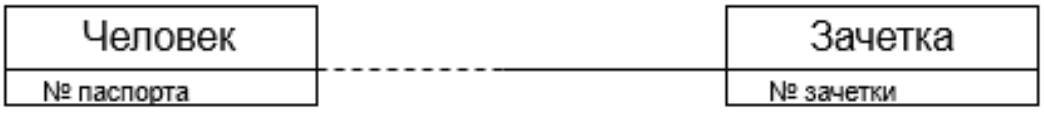
- Степень бинарной связи 1:1
- Класс принадлежности одной сущности является обязательным, а другой - необязательным

То:

- Требуется 2 отношения (по одному на каждую сущность)
- Ключ сущности является ключом соответствующего отношения
- Ключ сущности, для которой класс принадлежности необязателен, добавляется в качестве атрибута в другое отношение

Пример
Имееются сущности:

- "Человек" с атрибутами "№ паспорта", "ФИО", "год рождения", "адрес"
- "Зачетка" с атрибутами "№ зачётки", "учебное заведение", "факультет", "когда выдана".



Получим два отношения:

- Человек (№ паспорта, ФИО, год рождения, адрес)

- Зачётка(№ зачётки, учебное заведение, факультет, когда выдана, № паспорта)

Правила генерации отношений из ER-диаграмм (правило 3)

Если:

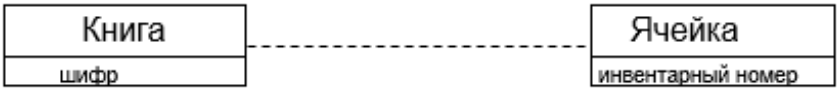
- Степень бинарной связи равна 1:1
- Класс принадлежности ни одной из сущностей не является обязательным

То:

- Требуется 3 отношения (по одному для каждой сущности - объектные, плюс ещё одно - связное)
- Ключами объектных отношений являются ключи соответствующих сущностей
- Связное отношение содержит ключи объектных отношений в качестве атрибутов

Пример

Имеем сущность «Книга в библиотеке» с атрибутами шифр, название, автор, год издания. Книга может находиться на полке в зале подручного фонда, а может и в запасниках. Если книга находится в зале, она занимает одну определенную ячейку на полке, в этой ячейке может быть несколько экземпляров одной книги, но не может быть другой книги, при этом ячейка может быть и пуста. Сущность «Ячейка» имеет атрибуты инвентарный номер, № зала, № стеллажа, № полки.



Получим три отношения:

- Книга (шифр, название, автор, год издания)
- Ячейка (инвентарный номер, № зала, № стеллажа, № полки)
- Подручный_Фонд (шифр, инвентарный номер, количество экземпляров)

Правила генерации отношений из ER-диаграмм (правило 4)

Если:

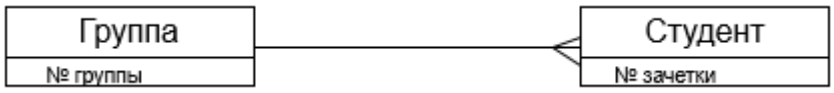
- Степень бинарной связи равна 1:n
- Класс принадлежности со стороны n-связной сущности является обязательным

То:

- Требуются два отношения (по одному для каждой сущности)
- Ключ сущности является ключом соответствующего отношения
- Ключ односвязной сущности добавляется как атрибут в отношении, соответствующее n-связной сущности

Пример

Сущность «Группа» имеет атрибуты № группы, факультет, специальность, курс. Группа состоит из студентов. Сущность «Студент» имеет атрибуты № зачетки, ФИО, адрес, год рождения. В группе обычно несколько студентов. Каждый студент обязательно относится к какой–нибудь группе.



Получим два отношения:

- Группа (№ группы, факультет, специальность, курс)
- Студент (№ зачетки, ФИО, адрес, год рождения, № группы)

Правила генерации отношений из ER-диаграмм (правило 5)

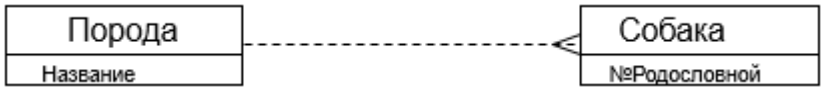
Если:

- Степень бинарной связи равна 1:n
- Класс принадлежности со стороны n-связной сущности является необязательным

То:

- Требуется 3 отношения (по одному для каждой сущности - объектные, плюс ещё одно - связное)
- Ключами объектных отношения являются ключи соответствующих сущностей
- Связное отношение содержит ключи объектных отношений в качестве своих атрибутов

Пример. В клубе собаководства есть перечень всех пород собак, которые будут представлены на очередной выставке, и список клубных собак. Сущность «Порода» имеет атрибуты: название породы, средний рост, средний вес, назначение, а сущность «Собака» имеет атрибуты: № родословной, кличка, возраст. Связь между сущностями «Порода» и «Собака» – 1:n, причем необязательная с обеих сторон, поскольку собак какой-то породы в данном клубе может и не быть, но могут быть собаки пород, не представленных на очередной выставке.



Получим три отношения:

- Порода (НазваниеПороды, СреднийРост, СреднийВес, Назначение)
- Собака (№Родословной, Кличка, Возраст)
- УчастникиВыставки (НазваниеПороды, №Родословной)

Правила генерации отношений из ER-диаграмм (правило 6)

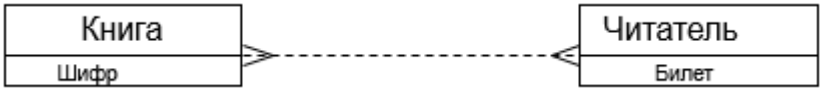
Если:

- Степерь бинарной связи m:n

То:

- Требуется 3 отношения (два объектных и одно связное)

Пример. Связь между сущностью Книга с атрибутами Шифр, Название, Автор, ГодИздания и сущностью Читатель с атрибутами Билет, ФИО, Телефон является связью n:m.



Получим три отношения:

- Книга (Шифр, Название, Автор, ГодИздания)
- Читатель (Билет, ФИО, Телефон)
- Взял (Шифр, Билет, Дата)

Правила генерации отношений из ER-диаграмм (правило 7)

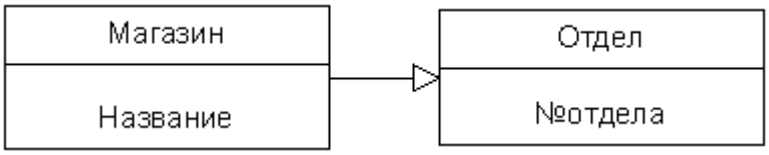
Если:

- Одна сущность является зависимой по отношению к другой сущности, т.е. не может существовать без неё

То:

- Будут образованы два отношения - по одному для каждой сущности со своими ключами
- В отношение, соответствующее зависимой сущности, добавится ключ отношения, соответствующего сущности, от которого она зависит
- Таким образом, ключом отношения, соответствующего зависимой сущности, будет составной клю, включающий ключи обеих сущностей.

Пример. Сущность «отдел в магазине» зависит от сущности «магазин», поскольку вне магазина отдел существовать не может.



Получим два отношения:

- Магазин (Название, …)
- Отдела (Название, №Отдела, …)

Правила генерации отношений из ER-диаграмм (правило 8)

Если:

- Одна сущность является общей сущностью, а другие сущности - её категориями, то возможны два варианта генерации отношений

Вариант 1:

- Общая сущность служит источником для генерации отношения, ключом которого является ключ этой сущности
- Генерируются отношения, соответствующие каждой из категорий
- Ключ общей сущности добавляется в атрибут отношений, соответствующих категориям

Вариант 2:

- Генерируются отношения, соответствующие каждой из категорий
- Общая сущность не порождает отдельного отношения, а все её

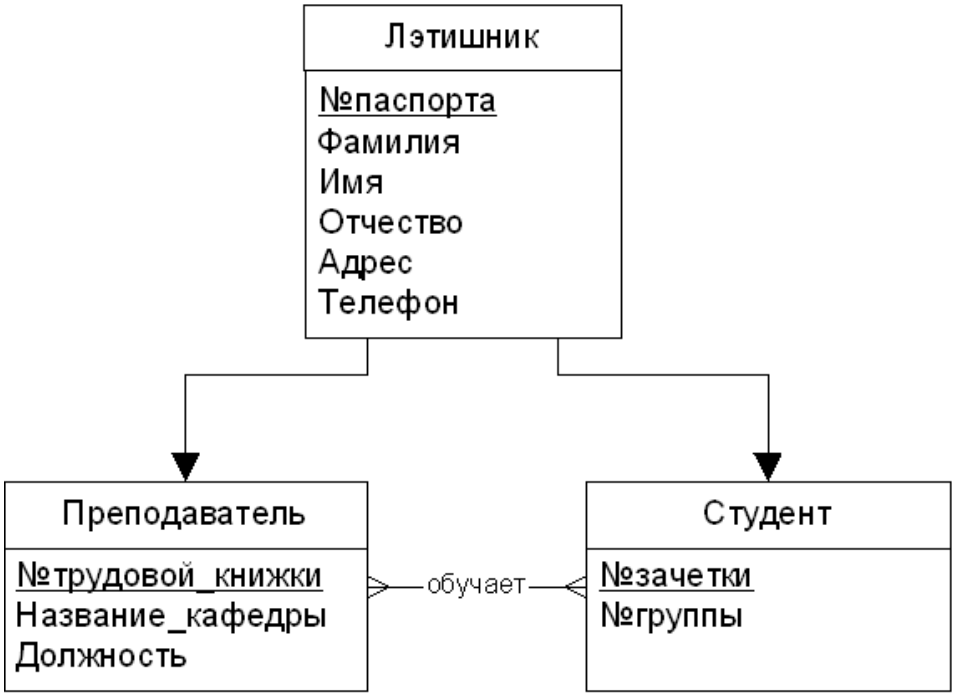
атрибуты становятся атрибутами объектных отношений, порожденных категориями

Выбор варианта зависит от количества общих атрибутов:

- Если количество атрибутов общей сущности больше, чем количество атрибутов её категорий, то обычно выбирают вариант 1
- Если количество общих атрибутов не превышает 2-3, то смысла в создании самостоятельного отношения для общей сущности нет и выбирают вариант 2.

Если категории имеют связи друг с другом, то порождается дополнительно такое число отношений, которое определяется правилами 1-6.

Пример



Вариант 1:

- Лэтишник (№паспорта, Фамилия, Имя, Отчество, Адрес, Телефон),
- Студент (№зачетки, №паспорта, №группы),
- Преподаватель (№трудовой книжки, №паспорта, Название кафедры, Должность),
- Обучает (№зачетки, №трудовой книжки, …).

Вариант 2:

- Студент (№зачетки, №паспорта, Фамилия, Имя, Отчество, Адрес, Телефон, №группы),
- Преподаватель (№трудовой книжки, №паспорта, Фамилия, Имя, Отчество, Адрес, Телефон, Название кафедры, Должность),
- Обучается (№Зачетки, №трудовой книжки, …).

Правила генерации отношений из ER-диаграмм (правило 9)

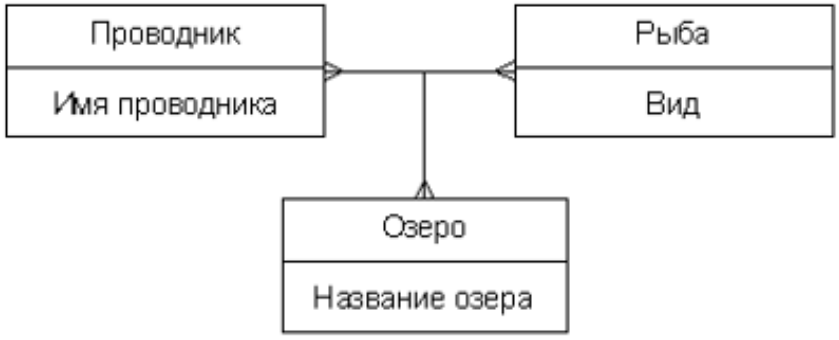
Если:

- Имеет место n-сторонняя связь между сущностями

То:

- Генерируется n+1 отношение - по одному для каждой сущности со своим первичным ключом и одно для связи
- Связное отношение имеет среди своих атрибутов ключи каждой участвующей в связи сущности

Пример



Получим 4 отношения:

- Проводник (ИмяПроводника, …),
- Рыба (Вид, …),
- Озеро (НазваниеОзера, …),
- ПредпочитаетЛовить (ИмяПроводника, Вид, НазваниеОзера).

Основные этапы проектирование БД методом "сущность-связь"

После того, как, используя перечисленные 9 правил, проектировщик получил предварительные отношения, он должен сделать следующее:

- Подготовить список значимых для задачи атрибутов, если это ещё не было сделано, и распределить их по предварительным отношениям.
- Определить функциональные зависимости (ФЗ) между атрибутами каждого отношения
- Проверить соответствие каждого отношения нормальной форме Бойса-Кодда (НФБК)
- Если некоторым атрибутам не нашлось места в предварительных отношениях, или если есть отношения, которые не находятся в НФБК, необходимо перестроить ER-диаграмму.

Данная последовательность действий представляет собой метод проектирования баз данных, который носит название **ER-метода**

Пример проектирования ER-методом

Постановка задачи

Пусть требуется создать базу данных для работника деканата.

В БД должны хранится сведения о группах и студентах:

ФИО студента, № зачетной книжки, №группы, кафедра, направление, год образования группы, адрес постоянной прописки студента, отметки о сданных зачётах и оценки, полученные студентом за курсовые работы и на экзамене по каждой из дисциплин в текущей сессии

Выделение сущностей

В предметной области можно выделить следующие сущности:

- Студент
  - № зачётки
  - Фио
  - Адрес
- Группа
  - № группы
  - Направление
  - Кафедра
  - Год образования
- Предмет
  - Название предмета
- Направление
  - № направления
  - Название направления
  - Срок подготовки
  - Квалификация

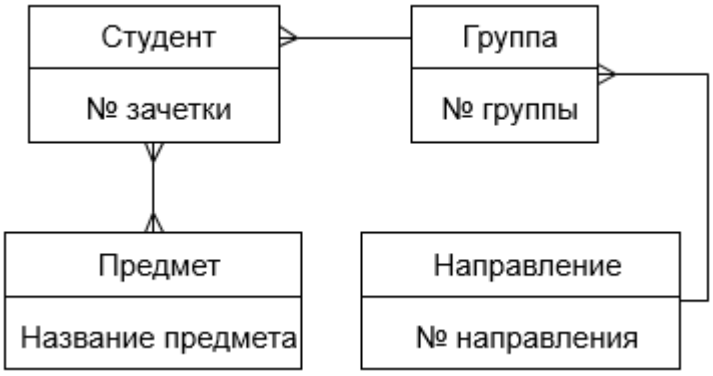
Добавление атрибутов

Студенты могут иметь в сессию по одному и тому же предмету до трёх записей в зачётной книжке, если сдают и зачёт, и экзамен, и курсовую работу. В ведомостях может быть и большее количество записей, если студент сделал несколько попыток сдачи экзамены или зачёта. Введём следующие атрибуты:

- Вариант 1:
  - Вид отчётности - зачет, экзамен, курсовая работа
  - Отметка - отлично, хорошо, удовлетворительно, неудовлетворительно, неявка, зачёт, незачёт
  - Дата
- Вариант 2:
  - Отметка зачета - зачёт, незачет, неявка
  - Отметка экзамена - отлично, хорошо, удовлетворительно, неудовлетворительно, неявка
  - Отметка курсовой работы - отлично, хорошо, удовлетворительно, неудовлетворительно, неявка
  - Дата зачёта
  - Дата экзамена
  - Дата курсовой работы

Во втором варианте возможно наличие пустых полей, что является недостатком. Кроме того, 2-й вариант хуже подвергается модификации.

ER-модель предметной области "Деканат"



- Студент (№_зачетки, №_группы)
- Группы (№_группы, №_направления)
- Специальность (№_направления)
- Предмет (Название_предмета)
- Сдал (№_зачётки, Название_предмета)

Набор предварительных отношений
Атрибуты, ещё не распределённые по отношениям: ФИО_студента, Адрес, Кафедра, год_образования, Название_направления, Квалификация, Срок_подготовки, Вид_отчётности, Дата, Отметка

Распределение
- Студент (№_зачетки, №_группы, ФИО_студента, Адрес)
- Группы (№_группы, №_направления, Кафедра, год_образования)
- Специальность (№_направления, Название_направления, Квалификация, Срок_подготовки)
- Предмет (Название_предмета)
- Сдал (№_зачётки, Название_предмета, Вид_отчётности, Дата, Отметка)

Отношение "Предмет" явно избыточно, так как все его атрибуты содержаться в отношении "Сдал"

ФЗ между атрибутами предварительных отношений. Проверка на соответствие НФБК
- ФЗ отношения Студент:
  - №_зачётки → № группы, ФИО_студента, Адрес

  Других зависимостей нет, так как ФИО может повторятся и в пределах группы, адреса студентов, живущих в общежитии, также могут совпадать. Отношение "Студент" по определению находится в НФБК
- ФЗ отношения Группа:
  - №_группы → №_направлния, Кафедра, Год_образования

  Обратная зависимость неверна, т.к. в один год на кафедре может быть сформировано несколько групп одного направления. Отношение "Группа" находится в НФБК
- ФЗ отношения Направление

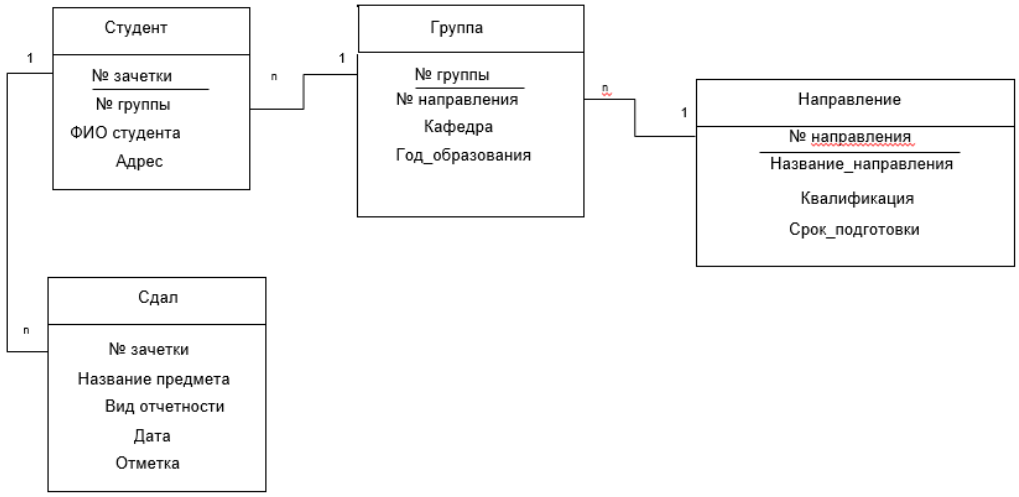
- №_направления → Название_направления, Квалификация, Срок_подготовки
- Название_направления → №_направления, Квалификация, Срок_подготовки

Как атрибут №_направления, так и атрибут Название_направления являются возможными ключами и детерминантами отношения. Отношение "Направление" находится в НФБК

- ФЗ отношения Сдал:
  - №_зачётки, Наазвание_предмета, Вид_отчётности, Дата → Отметка

Других зависимостей нет, следовательно, отношение Сдал находится в НФБК

Реляционная модель предметной области "Деканат"



V. Метод нормальных форм

Wednesday, October 3, 2018 12:37

Метод нормальных форм

Вторая нормальная форма (2НФ)

Отношение находится во 2НФ, если оно находится в 1НФ, и каждый неключевой атрибут функционально полно зависит от составного ключа, то есть зависит от всего ключа и не зависит от его частей.
Для приведения отношения ко 2НФ надо устранить частичную зависимость от ключа. Для этого отношение раскладывают на два, выполняя проекцию на часть составного ключа и зависящие от неё атрибуты.

Пример. Приведения ко 2НФ отношения "Библиотека"

- Шифр → Автор
- Шифр → Название
- Шифр → Год
- Шифр → Экземляр
- Билет → ФИО
- Билет → Телефон
- Шифр, Билет → Дата

Ключ - Шифр + Билет.

Выполним проекцию на функциональные зависимости от атрибута шифр. Получим 2 отношения:

- Отношение R1:
  - Шифр → Автор
  - Шифр → Название
  - Шифр → Год
  - Шифр → Экземляр
- Отношение R2:
  - Билет → ФИО
  - Билет → Телефон
  - Шифр, Билет → Дата

Ключ R1 - Шифр. R1 находится во 2НФ
Ключ R2 - Шифр + Билет. R2 не находится во 2НФ

Выполним проекцию на ФЗ от атрибута Билет. Получим вместо R2 два отношения:

- Отношение R3:
  - Билет → ФИО
  - Билет → Телефон
- Отношение R4

- Шифр, Билет → Дата

Клюс R3 - Билет, R4 - Шифр + Билет

Пример. *Приведение ко 2НФ отношения Преподаватель Предмет*

- Номер→ ФИО, Должность, Оклад, Кафедра, Телефон
- Должность → Оклад
- Кафедра ↔ Телефон
- Название → Часы

Ключ - Номер + Название.

Выполняется проекция:
Отношение "Преподаватель" с ФЗ:

- Номер→ ФИО, Должность, Оклад, Кафедра, Телефон
- Должность → Оклад
- Кафедра → Телефон

Ключ - Номер. Отношение "Преподаватель" находится во 2НФ.

В исходном отношение остались атрибуты Название, Часы и Номер:

- Номер
- Название → Часы

Ключ Номер + Название.

Выполняется проекция:
Отношение "Предмет":

- Название → Часы

Отношение "Преподает"

- Номер
- Название

Отношения "Предмет" и "Преподает" находятся не только во 2НФ, но и в НФБК. Отношение "Преподаватель" - нет.

Аномалии отношения "Преподаватель"

- Дублирование информации о телефоне для преподавателей одной кафедры или одинаковом для разных преподавателей должностном окладе приводит к тому, что изменение номера телефона кафедры влечет за собой необходимость поиска всех преподавателей, работающих на кафедре и изменения рабочего телефона каждого из них. То же и в случае изменения должностного оклада.
- Нельзя включить данные о новой кафедре, даже ее название, если на кафедре еще отсутствуют преподаватели. То же относится и к новой должности.
- При увольнении с кафедры всех преподавателей данные о ней не сохранятся. Не сохранятся данные и об окладе, например, ассистентов, если в БД не останется ни одного ассистента.

Устранить эти аномалии можно, перейдя к 3НФ

Третья нормальная форма (3НФ)

Отношение находится в 3НФ, если оно находится во 2НФ, и отсутствуют транзитивные зависимости неключевых атрибутов от ключа.
Для приведения отношения, находящегося во 2НФ, к 3НФ, нужно выполнить проекции на неключевые атрибуты отношения, связанные друг с другом функциональными зависимостями

Пример. Приведение к 3НФ отношения "Преподаватель"

Отношение "Преподаватель" с ФЗ:

- Номер→ ФИО, Должность, Оклад, Кафедра, Телефон
- Должность → Оклад
- Кафедра → Телефон
- Телефон → Кафедра

Выполним проекцию на атрибуты ФЗ: Должность → Оклад и Кафедра ↔ Телефон. Получим 3 отношения:

- Должность (Должность, Оклад)
- Кафедра (Кафедра, Телефон)
- Преподаватель (Номер, ФИО, Должность, Кафедра)

ИЛИ

- Должность (Должность, Оклад)
- Телефон (Телефон, Кафедра)
- Преподаватель (Номер, ФИО, Должность, Телефон)

Все эти отношения находятся в 3НФ и не противоречат определению усиленной 3НФ или НФБК

Различия между 3НФ и НФБК

В определении 3НФ говорится только о необходимости отсутствия в отношении транзитивных зависимостей неключевых атрибутов от ключа. Наличие в отношении зависимости части одного из возможных ключей от части другого возможного ключа не противоречит 3НФ, но противоречит НФБК.

Различие 3НФ и НФБК возможно только для отношений, удовлетворяющих следующим условиям:

- Отношение имеет два или более потенциальных Ключа
- Оба ключа являются сложными
- Потенциальные ключи отношения перекрываются, т.е. имеют хотя бы один общий атрибут

Для отношений, не удовлетворяющих хотя бы одному из этих условий, 3Нф и НФБК эквиваленты

Пример, демонстрирующий различие между 3НФ и НФБК

Пусть имеем отношение R со схемой:

R (Код_Поставщика, Имя_Поставщика, Код_Товара, Количество)

Допустим, что имена поставщиков уникальны, тогда потенциальными ключами этого отношения являются:

- Код_Поставщика, Код_Товара
- Имя_Поставщика, Код_Товара

Между атрибутами этого отношения существуют следующие ФЗ:

- Код_Поставщика → Имя
- Имя → Код_Поставщика
- Код_Поставщика, Код_Товара → Количество
- Имя, Код_Товара → Количество

Отношение не находится в НФБК, поскольку содержит детерминанты Код_Поставщика и Имя, определяющие друг друга, но не являющиеся ключами отношения.

Отношение находится в 3Нф, так как каждый неключевой атрибут, а именно атрибут Количество, функционально полно зависит от ключа, что соответствует определению 2Нф, и в отношении нет транзитивной зависимости неключевого атрибута от ключа.

Отношение R содержит некоторую избыточность, которая приводит к аномалиям обновления

Аномалии обновления 3НФ на примере

Если поставщик с кодом К1 по имени Иван Иванов поставляет товары с кодами Т1, Т2, Т3 и Т4,

Код_поставщика	Имя_поставщика	Код_товара	Количество
К1	Иван Иванов	Т1	100
К1	Иван Иванов	Т2	200
К1	Иван Иванов	Т3	150
К1	Иван Иванов	Т4	600

То при изменении кода поставщика придётся внести изменение в 4 строки. Чтобы избежать этой проблемы, отношение разбивают на два:

- R1 (Код_Поставщика, Имя)
- R2 (Код_Поставщика, Код_Товара, Количество)

Между этими отношениями должна быть установлена связь 1:n по атрибуту Код_Поставщика. Для поддержания ссылочной целостности следует использовать вариант каскадного обновления значения поля связи, тогда, заменив значение кода поставщика в одной строке отношения R1, получим такое же изменение в соответствующих сроках R2

Можно выбрать и альтернативное разбиение отношения R наЖ

- R1 (Имя, Код_Поставщика)
- R2(Имя, Код_Товара, Количество)

Связь 1:n между отношениями R1 и R2 устанавливается по полю Имя. В этом случае при изменении значения кода поставщика требуется изменить лишь одну строку отношения R1

Пример, когда декомпозиция не устраняет аномалии обновления ЗНФ

Имеем отношение Обучение (Студент, Предмет, Преподаватель) со следующими ограничениями:

- Каждый студент, изучающий данный предмет, обучается только одним преподавателем
  Студент, Предмет → Преподаватель
- Каждый преподаватель ведет только один предмет, но каждый предмет может преподаваться несколькими преподавателями
  Преподаватель → Предмет

Возможные ключи - Студент+Предмет или Студент+преподаватель/

Это отношение находится в ЗНФ, но не в НФБК, поскольку в НФБК каждый детерминант (левая часть отношения) должен быть возможным ключом. Здесь "Преподаватель" не является возможным ключом.

Выполнив проекцию отношения ФЗ Преподаватель → Предмет, получим:
R1 (Студент, Преподаватель)
R2 (Преподаватель, Предмет)
В результате декомпозиции мы потеряли ФЗ - Студент, Предмет →

Преподаватель.
Кроме того, мы не можем менять отношения независимо друг от друга.

Обучение

Студент	Предмет	Преподаватель
Иванов	математика	Белый
Иванов	физика	Черный
Петров	математика	Зеленый
Петров	физика	Красный
Сидоров	математика	Белый
Сидоров	физика	Красный
Митин	математика	Белый

R1

Студент	Преподаватель
Иванов	Белый
Иванов	Черный
Петров	Зеленый
Петров	Красный
Сидоров	Белый
Сидоров	Красный
Митин	Белый

R2

Преподаватель	**Предмет**
Белый	математика
Черный	физика
Зеленый	математика
Красный	физика

Попытка вставить в R1 строку Иванов Красный должна быть отвергнута, так как Красный преподает физику, а Иванов уже обучается физике у Черного.
Это пример ситуации, когда декомпозиция отношения на отношения, находящиеся в НФБК, и декомпозиция на независимые компоненты входят в противоречие.
Удовлетворить одновременно обеим целям удается не всегда.

Нормальные формы более высокого порядка, чем НФБК. 4НФ

4НФ отражает ограничения, накладываемые не на функциональные, а на многозначные зависимости между атрибутами отношения.
Многозначную зависимость следует рассматривать как обобщение понятия функциональной зависимости.
Отношение **находится в 4НФ**, если оно находится в НФБК и в нём отсутствуют многозначные зависимости.
Атрибут B отношения R **многозначно зависит** от атрибута A (A => B), если каждому значения атрибута A соответствует множество значений атрибута B, не связанного с другими атрибутами отношения R.

Многозначная зависимость возможна только при наличии в отношении не менее трёх атрибутов.
Связь между атрибутами A и B, означающую, что каждому значению A соответствует множество значений B, в отличие от многозначной зависимости, будем обозначать A ↠ B

Пример многозначной зависимости между атрибутами

Имеем отношение Читатель_Книга (Шифр, Билет, ФИО, Телефон). В нём существуют следующие ФЗ:

- Билет → ФИО
- Билет → Телефон

Атрибут Шифр многозначно зависит от атрибута Билет:

- Билет ⇒ Шифр

Каждому значению атрибута Шифр соответствует множество значений атрибута Билет:

- Шифр ↠ Билет

Каждому значению атрибута Телефон соответствует множество значений атрибута Билет и множество значений атрибута Телефон:

- ФИО ↠ Билет
- ФИО ↠ Телефон

При изменении номера телефона или ФИО читателя надо изменить несколько строк таблицы по числу книг, взятых читателем в данный момент. Устранение аномалии достигается разложением отношения на два:

- Читатель (Билет, ФИО, Телефон)
- Взял (Билет, Шифр)

Пример многозначной зависимости разных независимых друг от друга атрибутов от одного и того же ключевого атрибута

Имеем отношение:
Преподаватель (Личный_номер, ФИО, Дети, Предметы, Должность)
В нём существуют следующие ФЗ:

Личный_номер → ФИО
Личный_номер → Должность
И следующие многозначные зависимости:
Личный_номер ⇒ Дети
Личный_номер ⇒ Предметы

Имеет место аномалия вставки. Для её устранения надо выделить из отношения Преподаватель отношения:

- Дети (Личный_номер, Дети)
- Предметы(Личный_номер, Предметы).

Исходное отношение:

- Преподаватель (Личный_номер, ФИО, Должность)

Каждое из полученных отношений находится в 4НФ

Пятая нормальная форма (5НФ)

Если многозначных зависимостей более трёх, переход к 4НФ может не устранить избыточного дублирования, а, следовательно, и аномалий обновления. Тогда применяют 5НФ, которая гарантирует отсутствие аномалий, которые могут быть исключены разбиением на проекции (а не вообще всех аномалий)

Приведение отношения, находящегося в 4НФ к 5НФ выполняется таким образом, чтобы результат удовлетворял зависимости по соединению.
Зависимость по соединению является обобщение понятия многозначной зависимости и означает, что исходное отношение может быть восстановлено без потерь путём выполнения операции соединения по отношению к некоторым его проекциям.
Фраза **"без потерь"** означает, что при восстановлении исходного отношения мы не получим новых кортежей и не потеряем стары, не произойдёт также потери зависимстей между атрибутами.

Отношение находится в **5НФ** тогда, когда любая зависимость по соединению в исходное отношение определяется возможными ключами исходного отношения.

До 4НФ включительно, единственной операцией, допустимой или необходимой в процессе декомпозиции, была замена отношения двумя его проекциями. Для некоторых отношений нельзя выполнить декомпозицию без потерь на две проекции, но можно на три и более (n - декомпозируемое отношение, $n > 2$).
Для отношений, которые подвергаются декомпозиции на проекции и обратной композиции с соединением проекций, не существует более высокой степени зависимости, по отношению к которой ЗС является лишь частным случаем, но, если ввести другие операторы декомпозиции, то возможно появление других типов зависимостей, а следовательно, и

других нормальных форм.
Функциональные и многозначные зависимости обнаружить нетрудно, т.к. они имеют интерпретацию в терминах реального мира. Значения зависимостей по соединению не очевидны.

Пример приведения отношения к 5НФ
Имеем отноошение Изделие (Поставщик, Деталь, Изготовитель)
Между каждой парой атрибутов отношения Изделие существует связь "многие-ко-многим":

- Каждый поставщик может поставлять несколько различных деталей.
- Один и тот же тип детали может поставляться разными поставщиками.
- Изготовитель изделий (сборщик) может использовать разные детали
- Один и тот же тип детали может быть использован разными изготовителями.
- Изготовители могут быть связаны с различными поставщиками,
- Каждый поставщик может работать на несколько изготовителей.

Деталь ↞↠ Изготовитель
Деталь ↞↠ Поставщик
Поставщик ↞↠ Изготовитель
В отношении отсутствуют многозначные зависимости и оно состоит только из атрибутов, входящих в состав первичного ключа. Т.е. по определению отношение находится в 4НФ.

Пусть в неоторый момент времени информация в отношении Изделие имеет вид:

Поставщик	**Деталь**	**Изготовитель**
П1	Д1	А
П1	Д1	В
П2	Д2	А
П2	Д2	В
П3	Д1	А
П3	Д1	В
П3	Д2	А
П3	Д2	В

Выполним проекции отношения Изделие на зависимости 1,2,3
Получим три отношения - R1, R2, R3.
Выполнив операцию естественного соединения отношений R1 и R2, получим исходное отношение. Так же будет для любой пары. Значит, отношения R1, R2, R3 находятся в 5НФ

R1

Поставщик	Деталь
П1	Д1
П2	Д2
П3	Д1
П3	Д1
П3	Д2

R2

Деталь	Изготовитель
Д1	A
Д1	B
Д2	A
Д2	B

R3

Поставщик	Изготовитель
П1	A
П1	B
П2	A
П2	B
П3	A
П3	B

Иногда для того, чтобы получить исходное отношение, недостаочно выполнить одну операцию соединения проекций, может потребоваться последовательное выполнение двух или более операций.
Пусть имеет отношение R.

A	B	C
a1	b1	c1
a1	b2	c2
a3	b3	c3
a4	b3	c4
a5	b5	c5
a6	b6	c5

Разложим его на 3 отношения R1, R2, R3, выполнив сначала проекцию на атрибуты A,B, затем на атрибуты A,C, и наконец на атрибуты B,C.

R1

A	**B**
a1	b1
a1	b2
a3	b3
a4	b3
a5	b5
a6	b6

R2

A	**C**
a1	c1
a1	c2
a3	c3
a4	c4
a5	c5
a6	c5

R3

B	**C**
b1	c1
b2	c2
b3	c3
b3	c4
b5	c5
b6	c5

Выполнив операцию естественного соединения отношений R1 и R2, получим два лишних по сравнению с отношением R кортежа:

A	**B**	**C**
a1	b1	c1
a1	b1	c2
a1	b2	c1

a1	b2	c2
a3	b3	c3
a4	b3	c4
a5	b5	c5
a6	b6	c5

Выполним операцию естественоого соединения полученного отношения и отношения R3, для которых общими являются атрибуты B и C. Получим отношение (R1 ▻◅ R2) ▻◅ R3, совпадающее с исходным отношением R. Таким образом, отношения R1, R2, R3 находятся в 5НФ, при условии, что отношение R находится в 4НФ.

Пример, когда отношеение, находящееся в 4НФ, не удается привести к 5НФ

Заменим в третьей строке отношения Изделие Д2 на Д1 и, выполнив проекцию этого отношения на каждую пару атрибутов, получим отношения R1,R2,R3.

Поставщик	**Деталь**	**Изготовитель**
П1	Д1	A
П1	Д1	B
П2	Д1	A
П2	Д2	B
П3	Д1	A
П3	Д1	B
П3	Д2	A
П3	Д2	B

R1

Поставщик	**Деталь**
П1	Д1
П2	Д1
П2	Д2
П3	Д1
П3	Д2

R2

Деталь	**Изготовитель**
Д1	A

Д1	B
Д2	A
Д2	B

R3

Поставщик	**Изготовитель**
П1	A
П1	B
П2	A
П2	B
П3	A
П3	B

Выполнив операцию R1 ⊳⊲ R2, полуим отношение Изделие, в котором по отношению с исходным будут два новых кортежа - П2,Д1,В и П2,Д2,А. Выполнение операции (R1 ⊳⊲ R2) ⊳⊲ R3 не приведет к изчезновению этих кортежей, поскольку соединение выполняется по атрибутам Поставщик и Изготовитель, а отношение R3 имеет коретжи П2,В и П2,А. Таким образом, даже участие в операции соединения всех трёх проекций не приводит в получению исходного отношения. Результат разложения не удовлетворяет зависимости по соединению.

Привести отношение к 5НФ не удается, т.к. негласно оно содержит следующие ограничения:

- Поставщик П1 выпускает только деталь Д1 и поставляет ее как изготовителю А, так и изготовителю В.
- Поставщик П2 выпускает 2 детали – Д1 и Д2, при этом деталь Д1 он поставляет изготовителю А, а деталь Д2 – изготовителю В.
- Поставщик П3 поставляет обе выпускаемые им детали (Д1 и Д2) любому изготовителю.

Невозможность перехода к 5НФ объясняется наличием у поставщика П2 избирательного подхода к выбору изготовителя при поставке каждого вида выпускаемых им деталей.

VI. CASE-средства

Wednesday, October 10, 2018 12:32

VI. Средства автоматизации проектирования БД

К средствам автоматизации проектирования относятся программно-технологические средства, которые получили название **CASE-средств** Термин **CASE (Computer-Aided Software Engineering)** переводится как "автоматизированная разработка программного обеспечения". CASE-технология имеет ряд общих черт с такими методами разработки ПО, как визуальное программирование и методы **RAD (Rapid Application Development**). Во всех этих методах для создания программ применяют **графические интерфейсы, объектные методы** и **современные средства проектирования**

Особенности CASE-технологии

1. CASE-технология поддерживает все этапы жизненного цикла программной системы от анализа предметной области и формулировки требований к системе до сопровождения готового программного продукта. Разработка программной системы представляется в виде последовательности чётко выраженных этапов, переходы между которыми автоматизированы
   Этапы жизненного цикла ПС

2. CASE-технология в основном ориентирована на разработку больших программных систем в архитектуре "клиент-сервер", поскольку именно в этом случае этап анализа задачи и проектирования программного обеспечения представляет значительные трудности.
3. CASE-технология позволяет отделить этап проектирования ПО от его программирования, документирования и отладки. Она дает возможность разработчику работать на более высоком уровне абстракции, не отвлекаясь на детали, что позволяет, с одной стороны,

ускорить разработку, а с другой, избежать многих ошибок. При этом программирование и часть стандартных действий по проектированию автоматизированы.

4. При использовании CASE-технологии обязательным является наличие централизованной базы данных-репозитория, предназначенного для взаимодействия между разработчиками, поскольку CASE - это технология коллективной разработки программных систем
5. CASE-технология поддерживается инструментальными средствами разной функциональной ориентации и разного уровня сложности.

Классификация CASE-средств

При классификации CASE-средств используют такие признаки как **тип**, **категория, уровень.**

Классификация по типам

- Системы, предназначенные для создания спецификаций программной системы и её проектирования. Обычно они поддерживают известные методологии проектирования, такие как **SADT (Structured Analysis and Design Template)**, структурного системного анализа Гейна-Сарсона, система Джексона. Представителями систем этого типа являются CASE Аналитик, POSE, SELECT. По заданным свойствам разрабатываемой программной системы эти инструментальные средства создают спецификации компонентов будущей системы и спецификации интерфейсов, связывающих эти компоненты, а также позволяют получить архитектуру системы, алгоритмы и определения структур данных
- Средства, предназначенные для автоматизации этапа программированию. По спецификации они генерируют программный код. На выводе получаем готовую для выполнения документированную программу. К таким средствам относятся DECASE, NETRON, APS
- Средства, обеспечивающие логическое моделирование данных, автоматическую генерацию отношений БД и связей межу ними. К ним относятся ERwin, S-Designer, который, начиная с версии 6.0, имеет название PowerDesigner, Designer/2000, Silverrun
- Средства анализа программ и реинжиниринга. К ним относятся, например, Adpac CASE Tools, Incpetor/Recorder.
- Средства управления проектом. Типичным представителем является Project Workbench, которая поддерживает планирования, контроль выполнения проекта, взаимодействие разработчиков.

Классификация по категориям

- **Tools** - пакеты, решающие небольшую задачу, принадлежащую более глобальной проблема
- **Toolkit -** интегрированные программные системы, предназначенные

для одного из уровней разработки ПО
- **Workbench -** интегрированные средства, поддерживающие системный анализ, проектирование и реализацию программной системы. Такие системы используют репозиторий, обеспечивают автоматическую передачу информации между этапами разработки и между разработчиками.

Классификация по уровням
- Средства **верхнего уровня** - системы компьютерного планирования, их задачей является повышение эффективности деятельности руководства предприятия
- Средства **среднего уровня** - предназначены для поддержки этапов анализа требований к создаваемой системе, разработки спецификаций, архитектуры системы, алгоритмов, структур данных
- Средства **нижнего уровня** - средства программирования, они обычно использует спецификации, созданные средствами среднего уровня и автоматически генерируют до 90% кода. На средства нижнего уровня возложены также задачи тестирования и документирования ПО.

CASE-средства разработки БД

Делятся на два типа:
- **Независимые от СУБД**: ERwin, S-Designer. Поддерживает несколько платформ, то есть позволяют генерировать структуру БД для различных СУБД
- **Встроенные**. Обычно специализированы и ориентированы на СУБД, в состав которых входят

Пакеты, поддерживающие CASE-технологию разработки БД, содержат:
- Графические средства создания и редактирования ER-модели
- Средства автоматического отображения ER-модели в реляционную
- Средства создания по полученной реляционной модели реальной БД, ориентированной на заданную СУБД: описаний (структур) таблиц, составляющих БД, и постоянных связей между ними.

Реальная БД может создаваться либо непосредственно системой, либо с помощью сгенерированного сценария.
- В первом случае целевая СУБД должна быть установлена на жестком диске компьютера, используемого для проектирования, кроме того, требуется наличие ODBC драйвера целевой СУБД
- Во втором случае разработчик получает файл сценария, представляющий собой последовательность команд (программу) на диалекте SQL целевой СУБД. Для создания БД надо выполнить эту программу в среде целевой СУБД, установленной на любом компьютере.

Возможности CASE-средств разработки БД

Кроме создания новых БД такие системы позволяют:

- Используя **реинжиниринг**, по структуре существующей БД можно восстановить реляционную модель, а потом и ER-модель.
- **Модифицировать** существующую БД, причём содержащую данные, работая на уроне модели
- **Документировать** создаваемый проект БД
- Решить с помощью **триггеров и хранимых процедур** задачу обеспечения целостности данных
- Ввести уже на уровне модели **ограничения на значения атрибутов** БД
- Организовать работу группы проектировщиков.

CASE-система Designer/2000 фирмы Oracle, являясь встроенной системой, обладает по сравнению с независимыми от СУБД системами большими возможностями. Это средство разработки не только БД, но и приложений баз данных.

Система поддерживает все этапы жизненного цикла прикладной системы:

- Анализ деятельности предприятия
- Разработка модели предметной области
- Проектировании интерфейса приложения
- Генерация программного кода как для серверной, так и для клиентской частей разрабатываемой системы.
  Генератор серверной части позволяет получить последовательность команд на языке SQL, включающую команды создания:
  - Таблиц
  - Индексов
  - Внешних ключей
  - Триггеров хранимых процедур

  Генератор клиентских частей по заданным спецификациям позволяет получить тексты программных модулей различных типов:
  - Процедур обработки
  - Экранных форм
  - Отчётов

Повлиять на работу генераторов можно при помощи нескольких сотен параметров, манипулируя которыми можно получить клиентские приложения существенно отличающиеся друг от друга как по форме, так и функционально. Для обеспечения наглядности каждого этапа разработки в системе используется набор графических редакторов.

Преимущества и недостатки использования CASE-средств

- ✓ Case средства незаменимы при создании больших программных систем группой проектировщиков
- ✓ Использование CASE-средств позволяет сократить сроки разработки и

уменьшить число ошибок
- Возможности CASE-средств не безграничны: они позволяют моделировать только бинарные связи
- CASE-средства генерируют отношения по упрощённым правилам
- Около 80% CASE-проектов по разным причинам признаны неудачными

VII. Проектирование интерфейса пользователя

Wednesday, October 31, 2018 12:32

VII. Проектирование интерфейса пользователя

При разработке интерфейса следует исходить из семантики операций, которые будет выполнять пользователь при помощи создаваемой программной системы. Последовательность шагов, описывающих взаимодействие между пользователем и программной системой, называется **сценарием**.
В зависимости от ситуации, это взаимодействие может развиваться в разных направлениях, т.е иметь варианты развития. Множество сценариев, объединены общей задачей пользователя, называется **вариантом использования (Use Case).** В некоторых источниках Use Case переводится как **прецедент**.
Определить варианты использования помогает выделение в предметной области событий, на которые необходимо реагировать.
Приступая к разработке интерфейса, необходимо выполнить анализ **функциональных требований** к системе. Каждый прецедент является потенциальным требованием к системе.

Использование диаграмм прецедентов языка UML для проектирования интерфейса

На диаграмме прецедентов кроме них изображаются также пользователи разрабатываемой системы - **актёры (Actors)**

Прецедент начинается с некоторого действия актёра и представляет собой завершенную последовательность событий, описывающую один из вариантов взаимодействия актера и системы. Прецедент должен приводить к получению какого-то **результата.**
Актеры - не обязательно люди, хотя на диаграмме они изображаются в виде человечков. Некоторые из них могут представлять собой внешнюю систему, которой нужно получить информацию от разрабатываемой системы или передать ей информацию.
Актеры связаны с вариантами использования, исполняют их. Эта связь изображается на диаграмме прецедентов сплошной линией.

Между прецедентами существуют отношения трех видов:

- **Включение (Include)**. Обычно имеет место, когда какой-либо фрагмент поведения системы повторяется более, чем в одном варианте использования, и хотелось бы, чтобы его описание не дублировалось. Изображается на диаграмме пунктирной линией со стрелкой, направленной в сторону повторяющегося фрагмента, с подписью <include> или "включение"

- **Обобщение (Generalization)**. Из прецедента выделяется его частный случай - один из вариантов развития сценария. Изображается на диаграмме сплошной линией со стрелкой, направленной в сторону обобщающего прецедента.
- **Расширение (Extend)**. Связывает базовый и дополняющий его варианты использования. В этом случае в базовом варианте должны быть определены **точки расширения**. Только в этих точках расширяющий вариант использования может дополнять базовый. Изображается на диаграмме пунктирной линией со стрелкой, указывающий в сторону базового прецедента, с надписью "extend" или "расширение"

Как обобщение, так и расширение, позволяют выполнить расщепление прецедентов, разбить их на более мелкие части.

Пример

Требуется разработать программную систему для библиотеки. Будем считать, что взаимодействие библиотекаря с будущей автоматизированной система ограничивается тремя операциями:

- Записать в библиотеку нового читателя
- Выдать читателю книгу
- Зафиксировать возврат книги читателям

Поскольку выполнение каждой из операций дает разные результаты, их следует моделировать как разные прецеденты, но можно рассматривать как части одного более крупного прецедента, который можно назвать "обслужить читателя"

Описание последовательности шагов, реализующих прецедент

Каждый прецедент документируется в модели по следующей схеме:

- **Имя прецедента - к**раткое описание назначения прецедента, состоящего из одной-двух фраз
- **Актеры**, инициирующие прецедент и/или участвующие в нем
- **Предусловие**. Одно или несколько условий, которые должны быть истинными в начале прецедента

- **Описание прецедента**. Словесное описание главной последовательно. Перечисляются действия актера и ответная реакция системы на них
- **Альтернативы**. Словесное описание альтернативных ветвей прецедента
- **Постусловие**. Условие, которое должно быть истинным в конце предецента, если исполнение шло по главной последовательности
- **Неясные вопросы**. Вопросы документируются для обсуждения с заказчиками разрабатываемой системы.

Пример.
Описание прецедента "Обслужить читателя"

- **Имя** - "Обслужить читателя"
- **Краткое описание** - Выбор операции по обслуживанию читателя, которую необходимо выполнить в данный момент
- **Актер** - Библиотека
- **Предусловие** - На экране данные о читателях библиотеки и список доступных операций, связанных с обслуживанием читателей

Описание:

1. Библиотекарь инициирует выполнение необходимой ему операции, то есть запускает один из трех возможных прецедентов: "Записать читателя", "Выдать книгу" или "Зафиксировать возврат книги"
2. Система выполняет выбранный вариант развития сценария и возвращается в состояние, описанное в предусловии

Описание прецедента "Записать читатель"

- **Имя -** "Записать читателя"
- **Краткое описание** - Добавление в список читателей библиотеки данных о новом читателе, таких как номер читательского билета, паспортные данные, телефон
- **Актер** - Библиотекарь
- **Предусловие** - На экране данные о читателях библиотека
- **Описание**
    1. Система переходит в конец списка читателей и добавляет новую пустую запись (пустой бланк)
    2. Библиотекарь вводит данных нового читателя
    3. Система проверяет правильность ввода в каждое поле
    4. Если данные во все поля введены правильно, библиотекарь завершает регистрацию нового читателя
    5. Система проверяет правильность ввода строки в целом и отсутствие в исходном списке добавляемого в настоящий момент читателя
    6. Если проверка прошла успешно, система добавляет нового читателя в список
- **Альтернативы**

4. а. Если библиотекарь ввел в поле неправильные данные, то система выдает сообщение об ошибке и предоставляет библиотекарю возможность ввести данные повторно.
6. а. Если данные в полях строки вступают в противоречие друг с другом, система выдает сообщение об ошибке и предоставляет возможность изменения значения любого поля записи.
   б. Если человек, которого пытается зарегистрировать библиотекарь, уже есть в списке читателей библиотеки, система выдает сообщение и не добавляет этого читателя в список.

Описание прецедента "Выдать книгу"

- **Имя** - "Выдать книгу"
- **Краткое описание -** Закрепление книги за читателем. Указывается какой читатель, когда и какую книгу взял.
- **Актер** - Библиотекарь
- **Предусловие** - На экране данные о читателях библиотеки
- **Описание**
  1. Если данные о нужном читателе не видны, библиотекарь инициирует его поиск
  2. Система предоставляет библиотекарю возможность ввода критерия поиска. Можно искать по номеру читательского билета или по ФИО
  3. Если нужный читатель найден, то на экран выводятся сведение о нем, включая список книг, которые находятся у него на руках
  4. Если книг меньше трёх и нет книг, взятых более месяца назад, то библиотекарь добавляет в список книг читателя ещё одну. Для этого он должен ввести только шифр книги
  5. Система помещает в новую строку все остальные сведения о книге, соответствующие введенному шифру: автор, название, год издания. Дата закрепления книги за читателем – текущая дата также вводится автоматически.
- **Альтернативы**
  1. а. Если на экране видны данные о нужном читателе, то сразу выполняется пункт 3 основного сценария.
  3. а. Если читатель не найден, то , получив соответствующее сообщение, библиотекарь либо меняет критерий поиска, либо прерывает выполнение прецедента.
  4. а. Если книг больше двух или есть книги, взятые больше месяца назад, то библиотекарь прерывает выполнение прецедента.

Описание прецедента "Найти читателя"

Поскольку как прецедент «Выдать книгу», так и прецедент «Зафиксировать возврат книги» включает поиск читателя, то последовательность взаимодействия актера и системы, реализующую эту операцию, можно представить, как самостоятельный прецедент «Найти читателя»:

- **Имя** - Найти читателя
- **Краткое описание** - По номеру читательского билета или по Ф.И.О. найти человека в списке читателей библиотеки.
- **Актер** - Библиотекарь.
- **Предусловие** - На экране данные о читателях библиотеки.
- **Описание**
    1. Библиотекарь инициирует поиск читателя.
    2. Система предоставляет библиотекарю возможность ввода критерия поиска. Можно искать по номеру читательского билета или по Ф.И.О.
    3. Если поиск удачен, то на экран выводятся все данные о читателе, включая список книг, которые находятся у него на руках.
- **Альтернативы**
    3. а. Если читатель не найден, то система выводит соответствующее сообщение и предоставляет возможность библиотекарю либо поменять критерий поиска, либо прекратить поиск, то есть прервать выполнение прецедента.
- **Постусловие -** На экране видны сведения о нужном читателе.

Описание прецедента "Зафиксировать возврат книги"

- Описание прецедента «Зафиксировать возврат книги» после внесения изменений примет вид:
    1. Если данные о читателе, который хочет вернуть книгу, на экране не видны, то библиотекарь инициирует выполнение прецедента «Найти читателя».
    2. Система выполняет прецедент «Найти читателя».
    3. Если читатель найден, библиотекарь просматривает список закрепленных за ним книг.
    4. Если в списке книг, находящихся на руках у читателя, есть книга, которую он собирается вернуть, библиотекарь удаляет ее из списка книг читателя.
- **Альтернативы**
    1. Если данные о читателе, который хочет вернуть книгу, уже есть на экране, то сразу выполняется пункт 3 основного сценария.
    3. а. Если прецедент «Найти читателя» завершен, а читатель не найден, библиотекарь прерывает выполнение прецедента «Зафиксировать возврат книги».
    4. а. Если возвращаемой книги нет в списке книг читателя, то библиотекарь прерывает выполнение прецедента «Зафиксировать возврат книги».

Диаграмма прецедентов системы "Библиотека после выделения прецедента "Найти читателя"

Модель прецедентов позволяет сформулировать и наглядно представить функциональные требования к системе. Несмотря на некоторую громоздкость, она позволяет разработчику системы найти общий язык с заказчиком и будущим пользователем системы.

Реализация интерфейса пользователя

Основным средством разработки интерфейса является **экранная форма**, на которой располагаются элементы, обеспечивающие пользователю возможность доступа к данным, хранящимся в БД, а также возможность ввода данных с целью управления работой приложения. Такие элементы называются **элементами управления**. К ним относятся:

- **Поля ввода/вывода**. Предназначены для работы с данными, хранящимися в БД. С их помощью пользователь может просмотреть данные, добавить их, удалить или заменить. Можно ограничить возможности пользователя, разрешив ему, например, только добавлять и просматривать данные, но запретив удаление и редактирование данные. С помощью поля ввода можно задать значение переменной, а затем использовать это значение для управления приложением
- **Списки**. Позволяют выбрать один вариант из множества
- **Флажки (индикаторы)**. Используются в случае, когда нужно включить или отключить некоторую возможность. Можно выбрать одновременно несколько индикаторов или не выбрать ни одного
- **Радиокнопки (переключатели)**. Обычно объединяются в группу и позволяют выбрать один вариант из множества взаимоисключающих вариантов. Если вариантов больше пяти, то рекомендуется использовать не радиокнопки, а разворачивающиеся списки
- **Командные кнопки**. С ними связывать действие (операции), доступные пользователю, в .т.ч. операции перехода к другой форме, вывод на экран результата запроса или отчёта, изменение вида экранной формы

Пример реализиации интерфейса пользователя

Функциональность, изображенную на диаграмме прецедентов, можно реализовать при помощи пункта меню "Обслужить читателя", открывающего доступ к подменю, содержащему пункты:

- Записать читателя

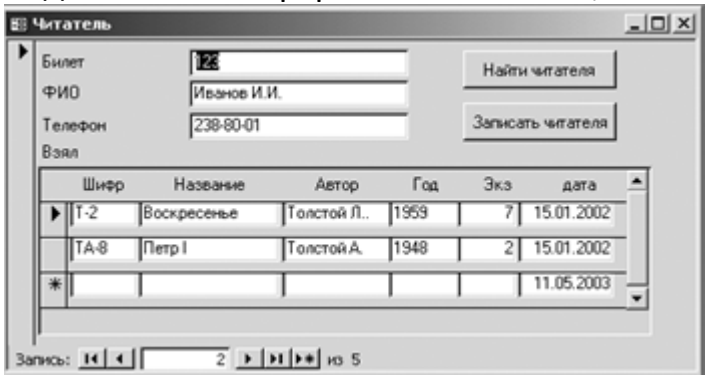
- Выдать книгу
- Зафиксировать возврат книги

Поскольку каждый из прецедентов, соответствующих трем указанным пунктам подменю, имеет одно и то же предусловие - "на экране данные о читателях библиотеки", то с выбором пункта меню "Обслужить читателя" (это может быть и кнопочное меню), целесообразно связать открытые формы "Читатель", позволяющей просматривать и изменять данные о читателях библиотеки.

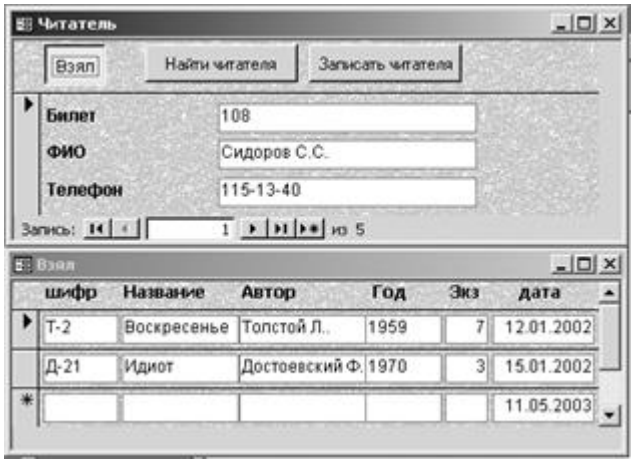
Форма должна представлять собой электронный аналог формуляров читателей. В каждый момент времени библиотекарь должен видеть на экране формуляр только читателя, с которым работает в данный момент. Перейти к формуляру другого читателя можно при помощи кнопок листания или кнопки поиска, с которой свяжем операции "Найти читателя". Из этой формы также должны быть доступны операции "Записать читателя", "Выдать книгу", "Зафиксировать возврат книги", которые можно связать с соответствующими командными кнопками

Примеры реализации формуляра в Access

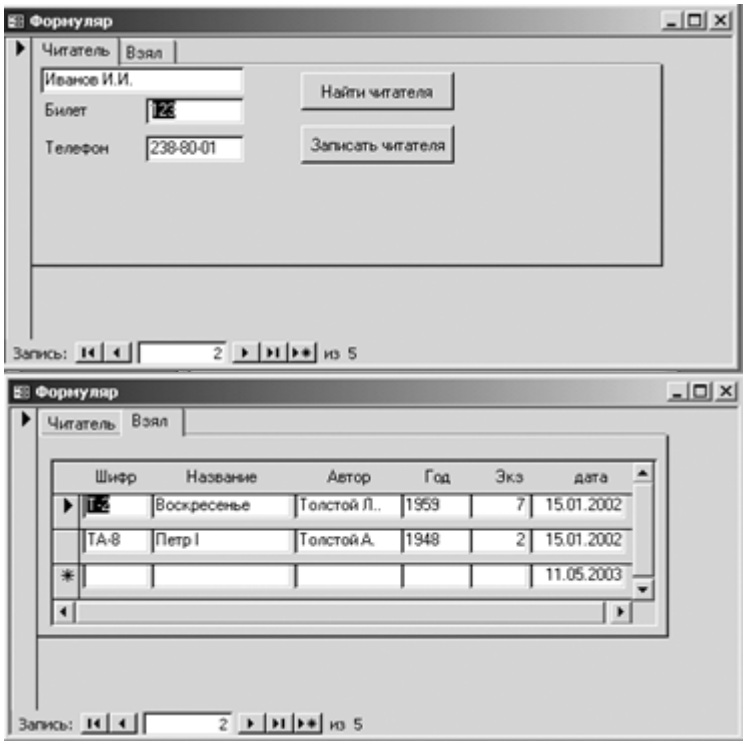
- **Форма с подчинённой**. Главная форма будет содержать ФИО читателя, номер его читательского билета и номер телефона, а подчинённая ей форма - список книг, взятых читателем



- **Форма со связанной формой**. То же, что и с подчинённой, но список взятых читателем книг находится в отдельной форме



- **Двухстраничная форма с вкладками**. На первой странице будут размещены ФИО читателя, номер его билета и телефона, а на второй - список книг, взятых читателем.



Рекомендации разработчику интерфейса (начало)

1. Сделать возможным доступ пользователя к хранящимся в БД данным т**олько через экранные формы**, максимально задействовав возможности СУБД по контролю достоверности и целостности данных
2. Предоставить пользователю возможность **возвратиться из любой точки приложения на шаг назад**, если при навигации по приложению, он выбрал неправильный вариант или забыл, что только что сделал.
3. Предоставить пользователю **в любой момент** по его желанию **завершить работу с приложением** или вернуться к точке старта - в главное окно приложения
4. **Снизить вероятность ошибок** пользователя за счёт предоставления ему понятных вариантов выбора. Наилучшее поведение для интерактивной системы - предвосхищение следующего шага пользователя. Поэтому данные для выбора пользователю надо предоставлять не в произвольном порядке, а в порядке, отвечающем логике использования данных. Например, если пользователь на первом этапе выбрал вариант А, то на втором этапе надо предоставить только варианты, относящиеся к А, и т.д. Если ошибка всё же произошла, приложение должно её обработать и вывести на экран краткое и ясное сообщение.
5. **Не загромождать экран данными.** Вспомогательную и редко используемую информацию рекомендуется выносить в отдельные подокна, а неосновные действия ограничить не кнопками, а выносить в меню.
6. **Избегать ситуации, когда на экране одновременно находится много окон, развернутых по принципу "матрешки"**. Одновременное

нахождение на экране и взывающего и вызванного окна оправдано только в ситуации, когда вызвано окно представляет собой вспомогательное окно, из которого в скором времени последует возврат к основному. В остальных случаях на времени работы с вызванным окном вызывающее окно лучше закрыть

7. С осторожностью предоставлять пользователю **возможность изменения размера окна**, поскольку, уменьшим размер окна, он может закрыть необходимые для работы элементы экранной формы,
8. **Максимально использовать стандарты, принятые в графическом интерфейсе Windows.** Если диалоговое окно содержит кнопки OK и Cancel, то они должны быть расположены в нижней части окна, причем кнопка OK левее кнопки Cancel. Допустимо расположение этих кнопок в правой части экрана, тогда кнопка OK должна располагаться над кнопкой Cancel.
9. При **размещении на экране командных кнопок** рекомендуется придерживаться следующих правил:
    - Кнопки должны быть сгруппированы по функциям и следовать друг за другом в горизонтальном и вертикальном направлении. Размещение кнопок должно быть выполнено с учётом частоты их использованию
    - Зазоры между кнопками одной группы должны быть в два раза меньше зазоров между группами
    - Размеры кнопок должны быть достаточно большими, чтобы вместить их названия
    - При вертикальном расположении кнопок их размеры должны быть одинаковыми, надписи на кнопках должны быть выровнены по центру
    - Если тексты на кнопках значительно отличаются по длине, лучше располагать такие кнопки горизонтально. Ширина кнопок в этом случае может быть различной.
10. Надписи элементов управления рекомендуется располагать по отношению к элементу управления следующим образом: внутри командной кнопки, справа от флажка, слева, над списком, или полем ввода/вывода.
11. **Избегать броских цветовых решений**. При работе с цветом придерживаться следующих правил
    - Экран не должен быть слишком ярким, следует использовать мягкие тона. Яркие цвета можно использовать только для выделения небольших областей
    - Цветов должно быть, как можно меньше. В большинстве случаем можно ограничится двумя.
    - Цветом можно выделить объекты экрана, относящиеся к одной группе, однородные объекты
    - Цветом можно отразить доступные и недоступные для выбора элементы интерфейса. То, что в настоящий момент нельзя

выбрать, должно быть бледным, слегка затуманенным
- Для текста и фона следует подбирать пары, в которых цвета контрастны, но дополняют друг друга, т.е., смешиваясь, образуют белый цвет, например, желтый и синий
- Сохранять общепринятые ассоциации:
  - Красный - ошибка
  - Желтый - предупреждение
  - Синий - спокойствие

  Если красный цвет означает "стоп", то противоположным будет зеленый, означающий "вперед", а если красный означает высокую температуру, то контрастным будет синий, ассоциирующийся с низкой температурой

12. Поскольку читать с экрана труднее, чем с листа бумаги, люди делают это медленнее, при работе с текстами необходимо придерживаться следующих правил:
    - При выводе предупреждений и сообщений об ошибках будьте лаконичны. Сообщение должно содержать четкую формулировку проблемы и способ её устранения. Если избежать длинного сообщения нельзя, то не создавайте хотя бы длинный строк, пусть лучше будет текст, вытянутый по вертикали, а не по горизонтали.
    - Избегайте слова "ошибка", иначе пользоватль может почувствовать себя неспособным освоить вашу программу и отказаться от её использования.
    - Не употреблять в текстах жаргонных слов и выражений, будьте внимательны к грамматике и пунктуации
    - При создании меню, экранных форм и отчётов не используйте много шрифтов, лучше ограничится двумя. Используйте стандартные **True Type** шрифты, чтобы напечатанный документ выглядел всегда, как и при предварительном просмотре. Для надежного форматирования полей ввода/вывода используйте равноширинные шрифты типа Courier. Для заголовков подойдет Arial.

VIII. Запросы. SQL: SELECT

Wednesday, November 7, 2018 11:55

Языки запросов

Запрос к БД формирует:

- Критерий отбор интересующих пользователя данных
- Способ преобразования отобранных данных к виду, удобному для восприятия

Существует два стандарта языка запроса к БД:

- Язык запросов по образцу QBE - Query By Example
- Язык структурированных запросов SQL - Structured Query Language

Языки запросов позволяют:

- Осуществлять выборку из БД
- Создавать новые таблицы БД
- Удалять, добавлять и изменять хранящиеся в таблицах данные

Язык QBE

QBE ориентирован на неквалифицированного пользователя. Предназначен для создания заранее не запланированных (ad hoc) запросов. Это язык визуального представления операции отбора или модификации данных. При формировании запроса на QBE пользователь должен указать:

- Источник данных запроса
- Столбцы таблицы - результата запроса
- Условие отбора данных
- Требуется ли сортировка данных и, если да, то по каким полям
- Требуется ли группировка данных и, если да, то по каким полям

Язык SQL

SQL - язык профессионалов. Это командный язык.
Позволяет формировать сложные и вложенные запросы. Имеет некоторые дополнительные возможности по сравнению с QBE.
Официальный стандарт ANSI/ISO опубликован в 1986 году.

Команды SQL

- Обработки данных
- Определения данных
- Управления доступом
- Управления транзакциями
- Команды программного SQL

Команды обработки данных

SELECT	Выборка данных из БД

INSERT	Добавление новых строк в БД
DELETE	Удаление строк из БД
UPDATE	Обновление данных

Команды управления транзакциями

COMMIT	Завершение транзакции
ROLLBACK	Отмена транзакции
START TRANSACTION	Начало транзакции

Транзакция - набор команд, которые должны быть либо выполнены все, либо не выполнена ни одна

Красным выделены команды, которых входят в большинство диалектов, но не в стандарт SQL

Команды определения данных

CREATE TABLE	Создание таблицы
CREATE VIEW	Создание представления
DROP TABLE	Удаление таблицы
DROP VIEW	Удаление представления
ALTER TABLE	Изменение структуры существующей таблицы
CREATE INDEX	Создание индекса
DROP INDEX	Удаление индекса

Команды предоставления доступа

GRANT	Предоставление доступа
REVOKE	Отмена доступа

Программный SQL

Программный SQL позволяет:

- Использовать операторы интерактивного SQL в тексте программы на ЯП высокого уровня
- Наряду с операторами интерактивного SQL использовать новые специальные конструкции, дополняющие SQL и увеличивающие его возможности
- Для передачи параметров в запрос использовать в тексте запроса переменные, объявленные в программе
- Для возврата в программу результатов запроса использовать специальные конструкции, отсутствующие в интерактивном SQL
- Осуществлять компиляцию запросов совместно с программой, обеспечивая впоследствии согласованную работу программы и СУБД.

Варианты встраивания запросов на SQL в программу

- **Статический SQL** - запрос определяется на стадии написания программы, в программу вставляется текст запроса
- **Динамический SQL** - позволяет формировать запрос к БД во время работы программы, реагируя на те или иные произошедшие события
- Метод, основанный на различных интерфейсах программирования приложения (**API**)

Команды программного SQL

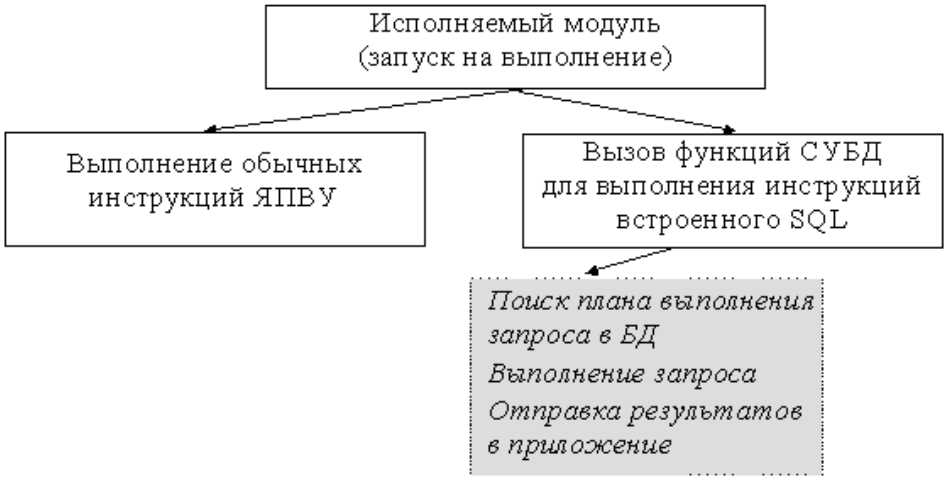
DECLARE	Определяет (объявляет) курсор для запроса
OPEN	Выполняет запрос, связанный с курсором, открывает доступ для чтения результата запроса
CLOSE	Закрывает курсор, освобождает память
FETCH	Считывает текущую строку из результата запроса
EXPLAIN	Возвращает описание плана доступа к данным для запроса
PREPARE	Подготавливает оператор SQL к динамическому выполнению (анализ синтаксиса, проверка параметров, выбор плана)
EXECUTE	Обеспечивает динамическое выполнение операторов SQL (подстановка параметров, выполнение плана, возврат результата в приложение)
DESCRIBE	Возвращает информацию о структуре курсора

Курсор - объект, хранящий результаты запроса

Статический SQL

Компиляция программы с встроенными инструкциями статического SQL

Выполнение программы



Преимущества и недостатки статического SQL

- ✓ Можно использовать SQL совместно с программой на языке высокого уровня
- ✓ Проверка синтаксиса запросов и оптимизация их выполнения осуществляется заранее один раз на этапе компиляции, что позволяет уменьшить время выполнения запроса
- – Переменные в запросах могут использоваться только в тех местах, где в интерактивных запросах стоят константы, например, нельзя задавать как параметр имя таблицы, из которой производится выборка, а также названия столбцов

Динамический SQL

Выполнение программы

Преимущества и недостатки динамического SQL

- ✓ Большая гибкость
- – Большее время выполнения, чем у статического запроса

Интерфейсы программирования приложений

API представляют собой библиотеки функций, разработанные для связи прикладной программы с СУБД посредством выполнения SQL-запросов.

Схема работы приложения совместно с SQL

- Программа получает доступ к БД путём вызова одной или нескольких API-функций, подключающих программу к СУБД и конкретной БД
- Для пересылки инструкций SQL в СУБД программа формирует инструкцию в виде текстовой строки и затем передает эту строку в качестве параметра при вызове API-функции
- Программа вызывает выполнение API-функции для проверки состояния переданной в СУБД инструкции и обработки ошибок
- Если инструкция SQL представляет собой запрос на выборку, то, вызывая API-функции, программа записывает результаты запросы в свои переменные. Обычно за один вызов возвращается одна строка или столбец данных
- Свое обращение к БД программа заканчивает вызовом API-функции, отключающей её от СУБД

Соглашения, используемые при описании синтаксиса SQL

Соглашение	Толкование
ПРОПИСНЫЕ БУКВЫ	Используются для написания ключевых и зарезервированных слов языка

Курсив	Заменяет имена переменных и элементов приложения, которых должны быть заданы пользователем
<Угловые скобки>	Характеризует элемент, которых должен быть задан пользователем
[Квадратные скобки]	Заключают в себе необязательные элементы. Если элементов несколько, они отделяются друг другом знаком "|". Можно выбрать один или ни одного
{Фигурные скобки}	Заключают в себя несколько элементов, разделённых "|", из которых нужно выбрать один
Многоточие ...	Показывают возможность многократного повторения элемента, после которого стоит

Синтаксис команды SELECT СУБД Access

```
SELECT [ALL | DISTINCT | DISTINCTROW | TOP <число>
[,PERCENT]] <спецификация выбора столбцов>
FROM <список таблиц и/или запросов>
[WHERE <критерий отбора>]
[GROUP BY <операция группировки>]
[HAVING <спецификация выбора групп>]
[UNION [ALL] <инструкция выбора>]
[ORDER BY <поле сортировки>]
[IN <”имя БД-источника”>[<строка подключения>]]
[WITH OWNERACCESS OPTION];
```

Примеры простейших запросов

Таблица Dis_Gr (Pr, FAK, DIS, GR)

Исходные данные - таблица DIS_GR

ФИО	Факультет	Дисциплина	Группа
Малышев С.В.	ФКТИ	БДиБЗ	5341
Матвеев Л.Б.	ФКТИ	ОП	2351
Сидоров С.С.	ФКТИ	ОП	3456
Иванов Н.П.	ФКТИ	ОСРВ	2351
Никитин Е.В.	ФКТИ	ТУ	2351
Соколова Н.Н.	ФКТИ	ТУ	2352
Сидоров С.С.	ФКТИ	ТУ	3456
Андреева А.И.	ФКЭА	ОП	5531
Сидоров С.С.	ФКЭА	ТУ	2345
Никитин Е.В.	ЭФФ	ОП	3421
Наумов С.А.	ЭФФ	ОП	3422
Васильев С.Л.	ЭФФ	ОП	4441

Таблица PREP

ФИО	Категория	Нагрузка	Контракт
Андреева А.И.	ассистент	880	05.01.1996
Васильев С.Л.	профессор	580	17.05.1997
Иванов Н.П.	профессор	620	20.01.1996
Малышев С.В.	асс.к.н.	780	25.03.1998
Матвеев Л.Б.	доцент	790	09.01.1996
Наумов С.А.	асс.к.н.	760	12.01.2002
Никитин Е.В.	доцент	650	02.01.1999
Сидоров С.С.	профессор	95	01.03.1997
Соколова Н.Н.	доцент	720	31.08.2001
Фомичева А.В.	ассистент	770	01.05.2000
Фомичева Т.Г.	доцент	750	01.09.1999

№	1
Пояснение	Вывести всю таблицу
SQL	`SELECT Pr, FAK, DIS, GR` `FROM Dis_Gr`

№	2
Пояснение	Вывести всю таблицу
SQL	`SELECT *` `FROM Dis_Gr;`

Опция WHERE

Содержит критерий отбора. Он может содержать

- Элементарные логические условия (>, >=, <, <=, =, <>)
- Предикаты

- Between <н.г.> AND <в.г.>
- Like "<шаблон>"
- IN (<список>)

- Сложные логические условия, в которых элементарные условия объединяются логическими операциями AND, OR, NOT, XOR, EQV, IMP

Примеры

№	4
Пояснение	Какие дисциплины и в каких группах и ведет преподаватель Сидоров С.С.?

SQL	`SELECT Pr, Dis, Gr`
	`FROM Dis_Gr`
	`WHERE Pr = ”Сидоров С.С.”;`

№	4
Пояснение	Кто кроме Сидорова С.С. ведет дисциплину ОП?

SQL	`SELECT Pr`
	`FROM Dis_Gr`
	`WHERE Pr <> ”Сидоров С.С.” AND Dis=”ОП”;`

Запросы с параметром

Пример запроса с параметром

№	5
Пояснение	В каких группах и какие дисциплины ведет заданный преподаватель?

SQL	`SELECT Dis_Gr.PR, Dis_Gr.DIS, Dis_GR.GR`
	`FROM DIS_GR`
	`WHERE Dis_Gr.PR = [ФИО преподавателя?]`

Результат	При значении "Сидоров С.С." **ФИО** **Дисциплина** **Группа** Сидоров С.С. ОП 3456 Сидоров С.С. ТУ 2345 Сидоров С.С. ТУ 3456 При значении "Никитин Е.В." **ФИО** **Дисциплина** **Группа** Никитин Е.В. ОП 3421 Никитин Е.В. ТУ 3512

На место ФИО преподавателя будет подставлено значение, запрошенное у пользователя

Поле может участвовать в условии отбора, но при этом не фигурировать в результате запроса.

№	6

Пояснение	Не обязательно выводить фамилию и инициалы преподавателя, если они введены, как значения параметра

SQL	`SELECT Dis_Gr.DIS, Dis_GR.GR`
	`FROM DIS_GR`
	`WHERE Dis_Gr.PR = [ФИО преподавателя?]`

Результат	При значении "Никитин Е.В." **Дисциплина** — **Группа** ОП — 3421 ТУ — 2351

Для обращения к форме используется конструкция
`[FORMS]![<имя формы>]![<имя элемента управления формы>]`
`[FORMS]![<имя формы>]![<имя подчинённой формы>].[FORM]![<имя элемента управления формы>]`

№	7
Пояснение	Пусть фамилия и инициалы интересующего нас преподавателя выбираются из списка с именем ФИО, расположенного на экранной форме «Преподаватель».

SQL	`SELECT Dis_Gr.DIS, Dis_Gr.GR`
	`FROM Dis_Gr`
	`WHERE Dis_Gr.PR = [Forms]![Преподаватель]![ФИО];`

Извлечение информации из нескольких таблиц
Информацию из нескольких таблиц в стандартном SQL можно извлечь, перечисляя нужные таблицы в FROM. В этом случае обязательно указание имени таблицы перед именем поля

№	8
Пояснение	Вывести список преподавателей с указанием их нагрузки, группы, в которой они работают и преподаваемой дисциплины.

SQL	`SELECT Prep.FIO, Prep.NAGR, DIS_GR.GR, DIS_GR.DIS`
	`FROM Prep, DIS_GR`
	`WHERE Prep.FIO=Dis_Gr.PR;`

Результат	

ФИО	Нагрузка	Группа	Дисциплина
Малышев С.В.	780	5341	БДиБЗ
Васильев С.Л.	580	4441	ОП
Сидоров С.С.	95	3456	ОП
Наумов С.А.	760	3422	ОП
Матвеев Л.Б.	790	2351	ОП
Андреева А.И.	880	5531	ОП
Никитин Е.В.	650	3421	ОП
Иванов Н.П.	620	2351	ОСРВ
Сидоров С.С.	95	2345	ТУ
Сидоров С.С.	95	3456	ТУ
Соколова Н.Н.	720	2352	ТУ
Никитин Е.В.	650	2351	ТУ

Если не указать последнее условие, то запрос будет выполнен по декартову произведению

№	9
Пояснение	Вывести список преподавателей, которые ведут дисциплину ОП и имеют нагрузку не более 700 часов.

SQL	`SELECT Prep.FIO, Dis_Gr.Dis, Prep.Nagr`
	`FROM Prep, Dis_Gr`
	`WHERE Prep.FIO= Dis_Gr.Pr AND (Dis_Gr.Dis=”ОП” AND Prep.Nagr <= 700);`

Результат	

ФИО	Дисциплина	Нагрузка
Никитин Е.В.	ОП	650
Васильев С.Л.	ОП	580
Сидоров С.С.	ОП	95

Спецификация выбора столбцов

Это список, в котором можно использовать

- Поля таблиц
- Текстовые константы
- Арифметические выражения, в т.ч. Содержащие функции

Синтаксис спецификации выбора столбцов

```
[ + | - ] { функция | (выражение) | литерал | имя_поля }
[{ + | - | * | / | \ | ^ | MOD | & } { функция |
(выражение) | литерал | имя_поля}] …
```

Операндами выражения могут быть:

- Поля таблиц и запросов, указанных в списке предложения FROM;
- Литералы (числовые, буквенно-цифровые константы, константы типа «дата/время»;
- Функции, встроенные в Access и/ или написанные пользователем.

Примеры операндов выражения

- `AVG(Nagr)` - функция, вычисляющая среднюю нагрузку преподавателей по данных таблицы Prep
- `Year(Contract)` - функция, вычисляющая год заключения контракта по дате, содержащейся в поле Contact таблицы Prep
- `Year(#01.04.2011#)` - возвращает значение 2011
- `Month(Date())` или `Month(Now())` - функция, определяющая месяц по текущей дате

Пример

Преобразования полей Фамилия, Имя, Отчество в фамилию с инициалами

```
[Фамилия] & " " & Left([Имя]; 1) & " " Left([Отчество]; 1) & "."
```

Использование выражений в запросе

В запросе можно наравне с полями таблицы выводит константы и выражения:

Исходные данные

ФИО	Оклад	Премия
Иванов И.И.	5000	2000
Петров П.П.	6000	1000
Сидоров С.С.	3000	1000
Михайлов М.М.	12000	2000

SQL

```
SELECT ФИО AS Работник, “Оклад+Премия” AS Выплаты, Оклад+Премия AS [К выдаче]
FROM TEST;
```

Результат

Работник	Выплаты	К выдаче
Иванов И.И.	Оклад+Премия	7000
Петров П.П.	Оклад+Премия	7000
Сидоров С.С.	Оклад+Премия	4000
Михайлов М.М.	Оклад+Премия	14000

Таблица-объединение

№	10
Пояснение	Диалект Access использует таблицу-объединение. При конструировании запроса 9 на QBE будет получен следующий код

SQL	`SELECT Prep.FIO, Dis_Gr.Dis, Prep.Nagr`
	`FROM Prep INNER JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR`
	`WHERE Dis_Gr.Dis=”ОП” AND Prep.Nagr <=700`

№	11
Пояснение	Запрос №8 с использованием таблицы-объединения

SQL	SELECT Prep.FIO, Prep.NAGR, Dis_Gr.GR, Dis_Gr.DIS FROM Prep INNER JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR;

Операция JOIN

Объединяет (соединяет) таблицы в логический набор данных. Для задания типа этого объединения используются слова перед JOIN:

- **INNER** - внутреннее (симметричное) объединение, в набор включаются соответствующие строки обеих таблиц
- **LEFT** - внешнее объединение, при котором все строки из первой таблицы объединяются с теми строками из второй таблицы, для которых выполняется условие объединения.
- **RIGHT** - внешнее объединение, при котором все строки из второй таблицы объединяются с теми строками первой таблицы, для которых выполняется условие объединения.

Пример

№	11 а
Пояснение	Запрос 11 со словом LEFT вместо INNER
SQL	SELECT Prep.FIO, Prep.NAGR, Dis_Gr.GR, Dis_Gr.DIS FROM Prep LEFT JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR;
Результат	

ФИО	Нагрузка	Группа	Дисциплина
Малышев С.В.	780	5341	БДиБЗ
Васильев С.Л.	580	4441	ОП
Сидоров С.С.	95	3456	ОП
Наумов С.А.	760	3422	ОП
Матвеев Л.Б.	790	2351	ОП
Андреева А.И.	880	5531	ОП
Никитин Е.В.	650	3421	ОП
Иванов Н.П.	620	2351	ОСРВ
Сидоров С.С.	95	2345	ГУ
Сидоров С.С.	95	3456	ГУ
Соколова Н.Н.	720	2352	ГУ
Никитин Е.В.	650	2351	ГУ
Фомичева А.В.	**770**	**Null**	**Null**
Фомичева Т.Г.	**750**	**Null**	**Null**

Пример использования в запросе внешнего объединения RIGHT

Для связанных таблиц запрос со словом RIGHT никак не отличается от запроса со словом INNER.

Разорвав постоянные связи между таблицами Prep и DIS_GR и добавив в таблицу DIS_GR строку:

Павлов С.М.	ФКТИ	ОП	2341

Выполним запрос 11 со словом RIGHT перед JOIN

```
SELECT Prep.FIO, Prep.Nagr, Dis_Gr.GR, Dis_Gr.DIS
FROM Prep RIGHT JOIN Dis_Gr on Prep.FIO=Dis_Gr.PR
```

Результатом будет таблица, отличающаяся от результата запроса со словом INNER дополнительной строкой

Null	Null	ОП	2341

Если бы фамилии и инициалы преподавателей выводились не из таблицы Prep, а из таблицы DIS_GR, то дополнительная строка имела бы вид:

Павлов С.М.	Null	ОП	2341

Пример использования при задании условия объединения таблиц слова LEFT

№	12
Пояснение	Запрос, возвращающий фамилии, инициалы и должности преподавателей, которые ничего не преподают.

SQL	`SELECT Prep.FIO, Prep.CATEG`
	`FROM Prep LEFT JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR`
	`WHERE (Dis_Gr.PR) Is Null;`

Результат	**ФИО** — **Категория** Фомичева А.В. — ассистент Фомичева Т.Г. — доцент

Использование неравенства при задании условия объединения таблиц

№	13
Исходные данные	Таблица "Предметы" **Код** — **Название** — **Количество_часов** БДиБЗ — Базы данных и базы знаний — 32 ВТвИР — ВТ в инженерных и экономических расчетах — 60 ОП — Основы программирования — 128 ОСРВ — Операционные системы реального времени — 32 ТУ — Теория управления — 48 ОЭ — Основы экономики — 40
Пояснение	Определить преподавателей, годовая нагрузка которых меньше, чем количество часов, отводимых на какой-нибудь отдельный предмет

SQL	`SELECT Prep.FIO, Prep.CATEG, Prep.NAGR,`
	`Предметы.Количество_часов,`
	`Предметы.Код FROM Prep`
	`INNER JOIN Предметы ON Prep.Nagr < Предметы.Количество_часов;`

Результат	**ФИО** — **Категория** — **Нагрузка** — **Количество_часов** — **Код** Сидоров С.С. — профессор — 95 — 128 — ОП

Задание альтернативного имени таблицы (запроса)

Альтернативное имя используется как псевдоним полного имени:

- Перед именами столбцов в списке выбора
- В предложении WHERE или подчинённых запросах, которые будут рассмотрены позже

Использование альтернативного имени удобно, если полное имя таблицы длинное.
В случае, когда происходит соединение (JOIN) таблицы с неё самой, употребление альтернативного имени становится обязательным

Пример

№	14
Пояснение	Имя таблицу Сотрудники (№_сотрудника, Фамилия, Имя, Отчество, должность, отдел, начальник), получить список сотрудников отдела №20, который кроме указания всех данных о каждом сотруднике должен содержать фамилию и номер его начальника. В качестве альтернативного имени таблицы "Сотрудники" используем "Подчиненные"

SQL	`SELECT Подчиненные.№_сотрудника, Подчиненные.Фамилия, Подчиненные.Имя, Подчиненные.Отчество, Подчиненные.должность, Подчиненные.отдел, Сотрудники.Фамилия, Сотрудники.№_сотрудника`
	
	`FROM Сотрудники`
	`INNER JOIN Сотрудники AS Подчиненные`
	`ON Сотрудники.№сотрудника=Подчиненные.начальник`
	`WHERE Подчиненные.отдел="20";`

Вложенные объединения

№	15
Пояснение	Вывести список преподавателей дисциплины «Основы программирования» с указанием их нагрузки.

SQL	`SELECT Prep.FIO, Prep.Nagr`
	`FROM Prep INNER JOIN (Предметы INNER JOIN Dis_Gr`
	`ON Предметы.Код = Dis_Gr.DIS) ON Prep.FIO = Dis_Gr.PR`
	`GROUP BY Prep.FIO, Предметы.Название`
	`HAVING Предметы.Название = "Основы программирования";`

Таблицы "Предметы" и Dis_Gr связаны по полю, содержащему код предмета, а их объединения связано с таблицей Prep по полю, содержащему фамилии и инициалы преподавателей. Группировка используется для того, чтобы не было многократного повторения в списке фамилии одного и того же преподавателя, если он преподает указанную дисциплину в разных группах

Сортировка
Строки таблицы, представляющей собой результат запроса, располагаются в разном порядке, установленном "по умолчанию". Принцип умолчания в разных версиях СУБД может быть различным:

- В Access >=2007 упорядочение результата производится в

соответствии с порядком строк таблицы-источника данных запроса
- В более ранних версиях применялся порядок по возрастания значения первого поля, указанного в списке SELECT

Если этот порядок нужно изменить, то используется предложение **ORDER BY <спецификация сортировки>**, позволяющее указать поле или список полей, по которым производится упорядочение.

Пример

```
SELECT Prep.FIO, Prep.NAGR, DIS_GR.GR, DIS_GR.DIS
FROM Prep, DIS_GR
WHERE (Prep.FIO=Dis_Gr.PR)
```

Этот запрос надо дополнить:
- Строкой ORDER BY Dis_Gr.Dis, если надо упорядочить результат по полю, содержащему название дисциплины
- Строкой ORDER BY Prep.Nagr DESC, если результат должен быть упорядочен по убыванию значений нагрузки.

Группировка

Для группировки данных команда SELECT использует две опции:
- **GROUP BY** <спецификация группировки>
  Позволяет объединить строки результата запроса в группы с одинаковым значением заданных после GROUP_BY полей
- **HAVING** <спецификация выбора групп>
  Используется только вместе с предложением GROUP BY и оказывается действие, подобное WHERE, но условие относится к группам данных, а не к отдельным строкам таблицы

Над группами можно выполнять различные операции, используя итоговые функции:
- COUNT({* | выражение}) - количество
- SUM(выражение) - сумма
- MIN(выражение) - минимум
- MAX(выражение) - максимум
- AVG(выражение) - среднее
- VAR(выражение) - дисперсия
- VARP(выражение) - смещённая оценка
- STDEV(выражение) - стандартное отклонение
- STDEVP(выражение) - смещённая оценка стандартного отклонения

Эти функции могут включатся в список после слова SELECT наряду с именами полей. При использовании любой итоговой функции в вычислении результата не участвуют значения Null

Примеры запросов с группировкой

№	16

Пояснение	Вывести количество преподавателей каждой из категорий:

SQL	`SELECT Prep.CATEG, Count (Prep.FIO) AS Количество`
	`FROM Prep`
	`GROUP BY Prep.CATEG;`

№	17
Пояснение	Подсчитать среднюю, максимальную и минимальную нагрузку для каждой категории преподавателей:

SQL	`SELECT CATEG, Avg(NAGR), Max(NAGR), Min(NAGR)`
	`FROM Prep`
	`GROUP BY CATEG;`

Результат	

Категория	Expr1001	Expr1002	Expr1003
асс.к.н.	770	780	760
ассистент	825	880	770
доцент	728	790	650
профессор	432	620	95

№	18
Пояснение	Чтобы вместо используемых по умолчанию заголовков столбцов Expr1001, Expr1002, Expr1003 выводились заголовки «Средняя нагрузка», «Максимальная нагрузка», «Минимальная нагрузка», надо в списке полей использовать слово AS

SQL	`SELECT CATEG, Avg(NAGR) AS [Средняя нагрузка], Max(NAGR) AS [Максимальная нагрузка], Min(NAGR) AS [Минимальная нагрузка]`
	`FROM Prep`
	`GROUP BY CATEG;`

Итоговые функции могут применяться не только к группам строк, но и к таблице или запросу в целом.

№	19
Пояснение	Подсчёт количества строк в таблице DIS_GR
SQL	`SELECT COUNT(*) FROM Dis_Gr;`

№	20
Пояснение	Если требуется вывести данные не по всем категориям преподавателей, а только по тем, для которых разница между максимальной и минимальной нагрузкой превышает 100 часов, следует использовать слово HAVING для задания условия отбора групп в результат запроса
SQL	`SELECT CATEG, Avg(NAGR) AS [Средняя нагрузка], Max(NAGR) AS [Максимальная нагрузка], Min(NAGR) AS [Минимальная нагрузка]` `FROM Prep`

	GROUP BY CATEG HAVING (Max([Nagr])-Min([Nagr]))>100;
Результат	Категория / Средняя нагрузка / Максимальная нагрузка / Минимальная нагрузка ассистент 825 880 770 доцент 728 790 650 профессор 432 620 95

№	21
Пояснение	В предложении HAVING также могут использоваться итоговые функции. Например, запрос 21 выведет список преподавателей, которые читают более одного предмета.
SQL	SELECT PR FROM Dis_Gr GROUP BY PR HAVING Count(DIS)>1;
Результат	ФИО Никитин Е.В. Сидоров С.С.

Никитин читает дисциплину ТУ в группе 2351 и дисциплину ОП в группе 3421, а Сидоров читает дисциплину ТУ в группах 3456 и 2345, а дисциплину ОП в группе 3456. Если бы Сидоров читал только дисциплину ТУ, но в двух или более группах, он все равно попал бы в список, и результат был бы неверен. Решить проблему можно, выполнив вместо запроса 21 последовательно два запроса 21_a и 21_b.

№	21 a
SQL	SELECT PR FROM Dis_Gr GROUP BY PR,Dis

№	21 b
SQL	SELECT PR FROM Запрос 21_a GROUP BY PR HAVING Count(DIS) > 1 s

№	21 c
Пояснение	То же, что и 21 a, b, но в одном запросе
SQL	SELECT PR FROM (SELECT PR, Dis FROM Dis_Gr GROUP BY PR, Dis) GROUP BY PR

```
HAVING Count (DIS) > 1;
```

Использование в команде SELECT слов ALL, DISTINCT, DISTINCTROW, TOP

- **ALL** - означает, что результатом запроса будут все строки, удовлетворяющие условию, среди которых могут оказаться и одинаковые
- **DISTINCT** - позволяет получить в качестве результата запроса только набор уникальных строк (дублирование не допускается)
- **DISTINCTROW** (диалект Access) - позволяет включить в результат запроса только строки, в которых конкатенация первичных ключей всех таблиц, участвующих в формировании результата запроса, поскольку не все столбцы каждой таблицы включаются в результат, но каждая строка выходной таблицы извлекается из уникальной комбинации строк базовых таблиц
- **TOP <число> [PERCENT]** - позволяет включить в результирующий набор первые n строк или первые n процентов**.**

Примеры

№	22
Пояснение	Результат запроса 22 (12 строк) будет содержать две одинаковые строки: «Сидоров С.С. 95 ТУ», поскольку Сидоров ведет дисциплину ТУ в двух группах, а номера групп в результат запроса не включаются. Использование слова DISTINCT в запросе 22 позволит избежать повторения в результирующей таблице двух одинаковых строк
SQL	`SELECT Prep.FIO, Prep.NAGR, Dis_Gr.DIS` `FROM Prep INNER JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR`

№	23
SQL	`SELECT TOP 2 Prep.FIO, Prep.NAGR, Dis_Gr.DIS` `FROM Prep INNER JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR;`

№	24
SQL	`SELECT TOP 10 PERCENT Prep.FIO, Prep.NAGR, Dis_Gr.DIS` `FROM Prep INNER JOIN Dis_Gr ON Prep.FIO = Dis_Gr.PR;`

Предложение TRANSFORM

Предназначено для формирования перекрестного запроса.

TRANSFORM <выражение с итоговой функцией> <инструкция SELECT с предложением GROUP BY> PIVOT <выражение, определяющее столбец>

- GROUP BY определяет, из какого столбца исходной таблицы/запроса

берутся значения заголовков строк перекрестной таблицы.
- После слова PIVOT указывается столбец исходной таблицы/запроса, из которого берутся значения, используемые в качестве заголовков столбцов перекрестной таблицы.
- Выражение с итоговой функцией определяет выражение для вычисления числовых значений, которые разместятся в ячейках перекрестной таблицы

Пример

№	25
Исходные данные	Таблица PRED КОД_ПРЕДМЕТА СЕМЕСТР КОЛИЧЕСТВО БДиБЗ 4 32 ВТвИР 3 60 ОП 1 64 ОП 2 48 ОП 3 16 ОСРВ 2 32 ТУ 2 32 ТУ 3 16
SQL	TRANSFORM Sum([КОЛИЧЕСТВО]) SELECT [КОД_ПРЕДМЕТА], Sum([КОЛИЧЕСТВО]) AS [Итого часов] FROM PRED GROUP BY [КОД_ПРЕДМЕТА] PIVOT [СЕМЕСТР];
Результат	КОД_ПРЕДМЕТА Итого часов 1 2 3 4 БДиБЗ 32 32 ВТвИР 60 60 ОП 128 64 48 16 ОСРВ 32 32 ТУ 48 32 16

Если столбец «Итого часов» не нужен, итоговую функцию в запросе 25 следует убрать из списка после слова SELECT

Предложение IN

Предложение IN команды SELECT представляет возможность получения данных из удаленного источника. Таким источником может быть другая БД Access или файлы в формате других СУБД, для которых есть ODBC - драйвер

IN <"имя БД-источника"> [<[строка подключения источника данных]>]

Если удаленным источником является Access, строка подключения не нужна.

Пример

№	26

Пояснение	Пусть хотим в запросе иметь доступ к БД с именем MY.mdb, расположенной на диске H: в каталоге Access. БД MY содержит таблицу Table1, в которой есть поле A. Чтобы иметь возможность вывести содержимое столбца A, не присоединяя таблицу Table1 к текущей БД, надо выполнить запрос 26.

SQL	`SELECT A`
	`FROM Table1`
	`IN "H:\Access\MY.mdb";`

№	27
Пояснение	Надо написать запрос, аналогичный запросу 26, но адресованный БД FoxPro 6.9, файлы которой располагаются в каталоге VFP на диске H. Потребуется изменить строку после IN

SQL	`SELECT A`
	`FROM Table1`
	`IN "H:\Access\MY.mdb" [FoxPro 6.0;];`

№	28
Пояснение	Аналогично запросу 27

SQL	`SELECT A`
	`FROM Table1`
	`IN "" [FoxPro 6.0; DATABASE=H:\VFP];`

Предложение WITH OWNERACCESS OPTION
Имеет смысл только в многопользовательской системе. Пользователь, выполняющий запрос, должен иметь право доступа не только к самому запросу, но и к таблицам, на основе которых он создан.

Пусть пользователь должен иметь возможность выполнить запрос, но не должен иметь возможности изменить данных в таблицах. Для защиты данных владелец должен установить для людей, использующих его запрос, ограниченные права доступа.

Включение в запрос предложения WITH OWNERACCESS OPTION наделяет пользователя теми же правами доступа, которыми обладает и его владелец, тем самым обеспечивается доступ к таблицам. Однако эти права обеспечиваются ему только на время выполнения данного запроса. В любой момент, когда пользователь не выполняет запрос, он теряет право доступа к таблицам

Предложение UNION
`SELECT … [UNION [ALL]<инструкция выбора>]`

Позволяет объединить таблицы, которые получаются в результате выполнения нескольких SELECT-команд.
Все строки-дубликаты автоматически удаляются. Если удалять строки-дубликаты не нужно, после слова UNION следует поставить слово ALL.
Таблицы-результаты всех SELECT-команд должны быть совместимы, т.е. иметь одинаковое количество столбцов и совместимые типы данных в соответствующих столбцах.
Имена столбцов каждой из SELECT-команд могут и не совпадать, но в объединенной таблице
Для указания имени столбца, по которому следует упорядочить данные результирующей таблицы, в последнюю SELECT-команду надо включить предложение ORDER BY. Указывается, однако, имя, возвращаемое первой, а не последней SELECT-командой.

Пример использования UNION

№	29
Пояснение	Если данные о студентах и сотрудниках учебного заведения, включая их адреса, хранятся в разных таблицах, то чтобы получить список людей, работающих или обучающихся в этом учебном заведении и проживающих в Московском районе, надо выполнить запрос 29

SQL	`SELECT Студенты.№паспорта, Студенты.Фамилия`
	`FROM Студенты`
	`WHERE Студенты.Район = "Московский";`
	`UNION`
	`SELECT Сотрудники.№паспорта, Сотрудники.Фамилия_Сотрудника`
	`FROM Сотрудники`
	`WHERE Сотрудники.Район = "Московский"`
	`ORDER BY Фамилия;`

Вложенные запросы
В состав условий предложений WHERE или HAVING могут входить команды SELECT. Таким образом определяется внутренний запрос. Данные, полученные после выполнения внутреннего запроса, используются потом для выполнения внешнего запроса.
Внутренние запросы могут возвращать как одно, так и несколько значений. От количества значений, возвращаемых внутренним запросов, зависит конструкция внешнего запроса.
Существуют два типа вложенных запросов:

- **Простой**. Внутренний запрос выполняется независимо от внешнего, его результаты используются для выполнения внешнего запроса
- **Коррелированный**. Внутренний запрос обращается одновременно к своей таблице данных и к таблица данных внешнего запроса.

Во внутреннем запросе нельзя использовать предложения ORDER BY и

UNION. Можно использовать предложения GROUP BY и HAVING, но тогда их уже нельзя использовать повторно внешнем запросе

Примеры простых вложенных запросов

№	30
Пояснение	Вывести фамилию преподавателя, который ведет дисциплину ОП в группе 2351, с указанием его годовой нагрузки

SQL	`SELECT FIO, NAGR`
	`FROM Prep`
	`WHERE FIO =(`
	`  SELECT PR`
	`  FROM Dis_Gr`
	`  WHERE DIS="ОП" AND GR="2351"`
	`);`

№	31
Пояснение	Запрос 30, переписанный с помощью обычного запроса

SQL	`SELECT prep.FIO, prep.NAGR`
	`FROM prep INNER JOIN Dis_Gr ON prep.FIO = Dis_Gr.PR`
	`WHERE Dis_Gr.DIS="ОП" AND Dis_Gr.GR="2351";`

№	32
Пояснение	Выбрать из таблицы Prep преподавателей, нагрузка которых выше средней.

SQL	`SELECT FIO, NAGR`
	`FROM Prep`
	`WHERE NAGR > (`
	`  SELECT AVG(NAGR)`
	`  FROM Prep`
	`);`

Примеры внутренних запросов, объединённых логическими операциями

№	33
Исходные данные	Таблица "Студенты"

№зачетки	Имя	Отчество	Фамилия	Специализация	Группа	Иногородний
1	Мария	Владимировна	Белова	22.01	10	Нет
2	Александр	Ильич	Белов	22.04	11	Да
3	Ия	Георгиевна	Шпаковская	01.02	12	Да
4	Наум	Яковлевич	Лев	22.04	11	Нет
5	Настасья	Владимировна	Фомичева	01.02	12	Нет
6	Иван	Андреевич	Белов	01.02	12	Нет

Таблица "Предметы"

Код	Название	Количество_часов
БДиБЗ	Базы данных и базы знаний	32
ВТвИР	ВТ в инженерных и экономических расчетах	60
ОП	Основы программирования	128
ОСРВ	Операционные системы реального времени	32

Код	Название	Количество_часов
БДиБЗ	Базы данных и базы знаний	32
ВТвИР	ВТ в инженерных и экономических расчетах	60
ОП	Основы программирования	128
ОСРВ	Операционные системы реального времени	32
ТУ	Теория управления	48
ОЭ	Основы экономики	40

Таблица "Экзамены"

№зачетки	Код_дисциплины	Оценка	Дата
1	ОП	5	02.01.2002
2	ОП	4	02.01.2002
3	ОП	3	02.01.2002
4	ОП	5	02.02.2002
5	ОП	3	02.02.2002
1	ТУ	4	04.01.2002
2	ТУ	4	04.01.2002
3	ТУ	4	12.01.2002
4	ТУ	4	12.01.2002
6	ОП	3	12.02.2002

Пояснение	Выяснить, какие оценки и когда были получены студентом Беловым из группы 11 по дисциплине «Основы программирования». Предполагается, что в пределах одной группы фамилии студентов не повторяются.

SQL	`SELECT Оценка, Дата`
	`FROM Экзамены`
	`WHERE №зачетки =(`
	`SELECT №зачетки`
	`FROM Студенты`
	`WHERE Фамилия ="Белов" AND Группа= "11"`
	`) AND Код_дисциплины = (`
	`SELECT Код`
	`FROM Предметы`
	`WHERE Название = "Основы программирования"`
	`);`

Вложенные запросы, в которых внутренний запрос возвращает несколько значений

Если внутренний запрос возвращает не одно, а несколько значений, то в предложение WHERE включают ключевые слова, позволяющие воспринять несколько значений, такие как **IN, ANY, SOME, ALL**. Чаще всего используется слово IN.

№	34
Пояснение	Аналогично запросу 33, но нужно узнать даты и экзаменационные оценки, полученные по дисциплине «Основы программирования» всеми студентами группы 11, а не только Беловым

SQL	<pre>SELECT №зачетки, Оценка, Дата
	FROM Экзамены
	WHERE №зачетки IN(
	SELECT №зачетки
	    FROM Студенты
	) AND Код_дисциплины = (
	SELECT Код
	    FROM Предметы
	    WHERE Название = "Основы программирования"
	);</pre>

При использовании во внешнем запросе ключевого слова **ANY** для приема нескольких значений, возвращаемых внутренним запросов, значение поля внешнего запроса поочерёдно сравнивается с каждым значением из возвращаемого множества. Если хотя бы одно из этих сравнений даёт результат "ИСТИНА", то и в целом проверка **ANY** дает результат "ИСТИНА". При сравнении можно использовать не только равенство, но и все остальные операции сравнения (<>, <, <=, >, >=). Слово **SOME** является синонимом слова **ANY**

Пример

Имеем две таблицы: Заказы и Служащие

№	35
Исходные данные	Таблица "Заказы" №заказа | Стоимость | №служащего 1 | 2 | 1 2 | 4 | 1 3 | 12 | 1 4 | 25 | 2 5 | 3 | 3 6 | 15 | 2 Таблица "Служащие" №служащего | ФИО | план 1 | Иванов И.И. | 30 2 | Петров П.П. | 40 3 | Михайлов М.М. | 50
Пояснение	Задача - вывести упорядоченный по полю ФИО список служащих, оформивших хотя бы один заказ на сумму, превышающую 10% от их индивидуального плана

SQL	<pre>SELECT №служащего, ФИО, план
	FROM Служащие
	WHERE (1.1 * план) < ANY (
	    SELECT стоимость FROM Заказы WHERE
	        (№служащего = Служащие.№служащего)
	)
	ORDER BY ФИО;</pre>

Результат	№служащего | ФИО | план 1 | Иванов И.И. | 30 2 | Петров П.П. | 40

При использовании во внешнем запросе ключевого слова **ALL** для приема нескольких значений, возвращаемых внутренним запросом, значение поля внешнего запроса поочерёдно сравнивается с каждым значением из возвращаемого множества. Если каждое из сравнений даёт результат "ИСТИНА", то и в целом проверка **ALL** дает результат "ИСТИНА".

№	36
Пояснение	Аналогично запросу 35, но требуется получить список служащих, которые оформляли только крупные заказы. В таком случае нужно слово ANY заменить словом "ALL"

SQL	`SELECT №служащего, ФИО, план`
	`FROM Служащие`
	`WHERE (0.1 * план) < ALL (`
	`  SELECT стоимость FROM Заказы WHERE`
	`  (№служащего = Служащие.№служащего)`
	`)`
	`ORDER BY ФИО;`

Результат	2 Петров П.П. 40

№	36 а
Пояснение	Запрос 36 даст правильный результат, если каждый служащий оформил хотя бы один заказ, т. е. если в таблице Заказы присутствует номер каждого служащего из таблицы Служащие. Если служащий не оформил ни одного заказа, сведения о нем будут включены в результат запроса 36. Данный запрос исправляет это

SQL	`SELECT №служащего, ФИО, план`
	`FROM Служащие`
	`WHERE (0.1 * план) < ALL (`
	`  SELECT стоимость FROM Заказы WHERE`
	`  (№служащего = Служащие.№служащего)`
	`) AND 0 <> (`
	`  SELECT COUNT(№служащего) FROM Заказы WHERE`
	`  (№служащего = Служащие.№служащего)`
	`)`
	`ORDER BY ФИО;`

Вложенные запросы, в которых внутренний запрос возвращает "ДА" или "НЕТ"

Запросы с проверками **ANY** и **ALL** всегда могут быть преобразованы в запросы с проверками на существование. Для реализации проверки на существование используются ключевые слова **EXISTS** и **NOT EXISTS**. Предикат **EXISTS** предназначен для приема внутреннего запроса значения "ИСТИНА" и "ЛОЖЬ". Если внутренний запрос обнаружил хотя бы одну строку, удовлетворяющую заданному в нём условию поиска, то предикат

EXISTS примет значение "ИСТИНА".
NOT EXISTS примет значение "ЛОЖЬ", если внутренний запрос не обнаружил ни одной строки
Только для предикатов EXISTS и NOT EXISTS во вложенных запросах допускается использование в списке выбора знака «*», означающего вывод всех полей таблицы.
Операции сравнения при замене ANY и ALL на EXISTS и NOT EXISTS переносятся внутрь условия поиска, относящегося к вложенному запросу.

№	37
Пояснение	Преобразование запроса 35

SQL	`SELECT №служащего, ФИО, план`
	`FROM Служащие`
	`WHERE EXISTS (`
	`  SELECT *`
	`  FROM Заказы`
	`  WHERE №служащего = Служащие.№служащего`
	`  AND (0.1 * Служащие.план) < стоимость`
	`)`
	`ORDER BY ФИО;`

№	38
Пояснение	Преобразование запроса 36 а

SQL	`SELECT №служащего, ФИО, план`
	`FROM Служащие`
	`WHERE EXISTS(`
	`  SELECT * FROM Заказы WHERE №служащего = Служащие.`
	`№служащего`
	`) AND NOT EXISTS (`
	`  SELECT * FROM Заказы WHERE`
	`№служащего = Служащие.№служащего`
	`  AND (0.1 * Служащие.план) > стоимость`
	`)`
	`ORDER BY ФИО;`

Используя NOT EXISTS, можно с помощью вложенного запроса реализовать запрос, который даст тот же результат, что и запрос 12? То есть найти преподавателей, которые ничего не преподают

№	39
Пояснение	Преобразование запроса 12

SQL	`SELECT FIO, CATEG`
	`FROM Prep`
	`WHERE NOT EXISTS(`
	`  SELECT *`
	`  FROM DIS_GR`
	`  WHERE Prep.FIO = Dis_Gr.PR`
	`);`

Запросы более с чем двумя уровнями вложенность

№	40
Пояснение	Задача - получить перечень студентов с указанием отводимых на них по учебному плану часов, которые сдавали студенты группы 12

SQL	`SELECT Название, Количество_часов`
	`FROM Предметы`
	`WHERE Код IN(`
	`  SELECT Код_дисциплины`
	`  FROM Экзамены`
	`  WHERE №зачётки IN(`
	`  SELECT №зачётки`
	`  FROM Студенты`
	`  WHERE Группа="12"`
	`  )`
	`)`

№	41
Пояснение	Описание запроса 40 в одноуровневом виде

SQL	`SELECT DISTINCT Предметы.Название, Предметы.Количество_часов`
	`FROM Предметы, Экзамены, Студенты`
	`WHERE Экзамены. Код_дисциплины = Предметы. Код`
	`AND Экзамены.№зачетки = Студенты. №зачетки`
	`AND Студенты. Группа = "12”;`

Любой вложенный запрос можно представить в виде одноуровневого запроса к нескольким связанным таблицам, но не все одноуровневые запросы можно представить в виде вложенных структур.

IX. DELETE, UPDATE, SELECT INTO

Wednesday, November 21, 2018 12:19

Команда "DELETE"

Синтаксис:

```
DELETE [список полей]
FROM {{<имя таблицы> [ AS <псевдоним> ] |
       <имя запроса-выборка> [ AS <псевдоним> ]},…|
       <таблица-объединение>}
[WHERE <условие отбора>]
```

№	42
Пояснение	Удалить все строки из Dis_Gr
SQL	`DELETE *` `FROM Dis_Gr`

№	43
Пояснение	Удалить все строки из Dis_Gr, относящиеся к ФКТИ

SQL	`DELETE *`
	`FROM Dis_Gr`
	`WHERE FAK = 'ФКТИ'`

Если в команде DELETE в качестве источника данных используется запрос, он не должен содержать оператор **UNION**, итоговых функций, слов **DISTINCT, GROUP BY, HAVING**
Запрос, являющийся источником для запроса на удаление, не может также внутри себя запрос, использующий ту же базовую таблицу, что и запрос на удаление.

№	44
Пояснение	Удалить из DIS_Gr строки, в которых содержится информация о предметах, на которые отведено более 48 часов

SQL	`DELETE Dis_Gr.*`
	`FROM Dis_Gr`
	`INNER JOIN Предметы`
	`ON Dis_Gr.Dis = Предметы.Код`
	`WHERE Предметы.Количество_часов > 48`

Если в качестве источника данных запроса указаны две таблицы и более, то при связи 1:n можно удалять строки таблицы со стороны "n", а при связи 1:1 - из любой, но одной таблицы.

Команда UPDATE

Предназначена для внесения одинаковых изменений в строки таблицы, удовлетворяющие заданному услвию

```
UPDATE { {<имя таблицы> [AS <псевдоним>] |
       <имя запроса-выборки [AS <псевдоним>]}, … |
       <таблица-объединение>}
SET {<имя поля> = {<выражение> | Null }},…
[WHERE <условие отбора>]
```

№	45
Пояснение	Уменьшить количество часов, которые отводятся на каждый предмет, на 20%
SQL	`UPDATE Предметы` `SET Предметы.Количество_часов = [Количество_часов]*0.8`

Если в качестве источника данных запроса указаные две таблицы и более, то при связи 1:n можно удалять строки таблицы со стороны "n", а при связи 1:1 - из любой, но одной таблицы.

№	46
Пояснение	Увеличить нагрузку преподавателей, работающий на ФКТИ, на 10%. Потребуется информация из двух таблиц - Prep и Dis_Gr, связанных друг с другом отношением 1:n. Prep - главная, в ней нужно изменить значение слова NAGR. Выполнить операцию непосредственно с помощью UPDATE нельзя, поэтому нужно создать запрос-выборку с полями FIO и NAGR и условие Dis_Gr.FAK = "ФКТИ", а затем использовать его в запросе на изменение

SQL	Запрос FOR_UPD:
	`SELECT Prep.FIO, Prep.NAGR`
	`FROM Prep INNER JOIN Dis_Gr`
	`ON Prep.FIO = Dis_Gr.PR`
	`WHERE Dis_Gr.FAK = 'ФКТИ'`
	
	Запрос 46:
	`UPDATE FOR_UPD`
	`SET NAGR = [NAGR] * 1.1`

№	46 а
Пояснение	То же, что и 46, но не требущее присовения имени вспомогательному запросу

SQL	`UPDATE(`
	`  SELECT Prep.FIO, PREP.NAGR`
	`  FROM Prep INNER JOIN Dis_Gr`
	`  ON Prep.FIO = Dis_Gr.PO`
	`  WHERE Dis_Gr.FAK = 'ФКТИ'`
	`)`

```
SET NAGR = [NAGR] * 1.1
```

Если условие отбора содержит подчинённый запрос, в нём не должно быть ссылки на обновляемую таблицу (запрос) или на базовую таблицу запроса на обновление.
В предложении SET имя поля не может встречаться более одного раза.
Значения, присваиваемые полям, должны быть совместимы с их свойствами. В случае любого нарушения запрос не выполняется, выдается предупреждающее сообщение

Команда INSERT INTO

Предназначена для вставки в таблицу или запрос одной или нескольких строк.
Синтаксис:
INSERT INTO <имя таблицы> [(<имя поля>, ...]
{ **VALUES** (<литерал>,…) | <инструкция выбора> }
Если имена полей отсутствуют, то добавляемые значения должны соответствовать порядку полей в определении таблицы. Если список имёт полей есть, ему должен соответствовать список значений, указанный после **VALUES**, или список полей после **SELECT** в инструкции выбора. В список имен обязательно должны попасть поля, для которых свойство "Обязательное поле" установлено в значение "ДА", а значение по умолчанию не задано.
При использовании в запросе **INSERT INTO** предложения **VALUES** вставляется одна строка, при использовании инструкции выбора - столько строк, сколько возвращает эта инструкция.
Вставляемые данные, как и в команде **UPDATE**, должны удовлетворять ограничениям на значения полей, ограничениям на значения для таблицы и условием ссылочной целостности.

№	47
SQL	`INSERT INTO Предметы ( Код, Название, [Количество часов] )` `VALUES ("ОЭ", "Основы экономики", "40")`

№	48
Пояснение	Трансляция запроса 47 на диалект Access
SQL	`INSERT INTO Предметы ( Код, Название, [Количество часов] )` `SELECT "ОЭ" As Expr1, "Основы экономики" AS Expr2, 40 As Expr3`

№	49
Пояснение	Запрос на добавление с инструкцией выбора, который позволяет добавить в одну таблицу данные из другой

	таблицы

SQL	`INSERT INTO Contract_Out`
	`SELECT Prep.*`
	`FROM Prep`
	`WHERE ((Date()-[Contract])/365)>5;`

Команды SELECT … INTO

Позволяет на основе одних таблиц создавать другие, как в своей, так и в другой БД. При создании новой таблицы копируются не все, а только основные свойства полей исходной таблицы, такие как тип и размер. Обычно используются для:

- Архивации данных
- Создания резервных копий таблиц
- Создание копий для экспорта данных в другую БД
- Создания копий для составления отчётов, отображающих данные за определённый период времени

Синтаксис:

SELECT <имя поля> [,…] **INTO** <имя новой таблицы> [**IN** <путь к внешней БД>] **FROM** <источних данных> [**WHERE** и т.д.]

Источник данных - это список таблиц и/или запросов, из которыз отбирают записи

№	50

SQL	`SELECT Prep.* INTO Contract_Out1`
	`FROM Prep`
	`WHERE ((Date()-[Contract])/365)>5`

Ошибки при выполнении запросов-действий

- **Дублирование первичного ключа**
  Возникает при попытке добавить или изменить запись
- **Ошибки преобразования**
  Возникают при вставке записей в существующую таблицу или при их обновлении, если тип данных источника и приёмника не совпадают, и данные источника не могут быть преобразованы к типу данных приемника
- **Заблокированные записи**
  Ошибка возникает при многопользовательском режиме работы в случае выполнения операции обновления или удаления записей, если нужные записи в данный момент используются другим пользователем
- **Нарушение условий на значение**
  Вставляемые или обновляемые записи могут нарушить условия на значения, заданные для поля обновляемой таблицы или для всей

таблицы.

Проблема (не ошибка) может возникнуть при усечении данных, добавляемые в текстовыми или MEMO-поля, при преобразовании числовых значений. Access проводит усечение без предупреждения, потому, выполняя запрос-действие, надо быть уверенным в том, что поля, принимающие данные, имеют достаточный размер.

X. Команды определения данных и управления доступом

Wednesday, November 28, 2018 11:49

Команды определения данных

Позволяют создавать или изменять структуру БД. Используются для создания или удаления таблиц и индексов, а также для изменения структуры таблиц в текущей БД

Команда	Описание
CREATE TABLE	Создание таблицы
CREATE INDEX	Создание индекса для поля или группы полей
CREATE VIEW	Создание представления
CREATE PROCEDURE	Создание процедура SQL
ALTER TABLE	Добавление, удаление поля, изменение типа поля, добавление или удаление индексов
DROP TABLE	Удаление таблицы
DROP INDEX	Удаление индекса
DROP VIEW	Удаление представления
DROP PROCEDURE	Удаление процедуры

Команда CREATE TABLE

Предназначена для описания вновь создаваемой таблицы, её полей и индексов

CREATE [**TEMPORARY**] **TABLE** <имя таблицы> (<поле>, …, [<составной индекс>, ...]

TEMPORARY означает, что создается временная таблица, которая будет автоматически удалена после завершения сеанса.

Конструкция <поле> имеет следующий вид:

<имя поля> <тип> [<размер>] [**NOT NULL**] [<описание простого индекса>]

Команда позволяет задать такие свойства поля, как "тип", "размер", "обязательное поле", "индексированное поле". Размер поля в знаках задается только для текстовых и двоичных полей. Ключевое слово **NOT NULL** позволяет задать обязательное поле.

В Access **CREATE** не позволяет задать значение по умолчанию или условие на значение. Также таблица должна содержать хотя бы одно поле.

Типы данных в Access

Тип данных Access	Тип данных SQL	Примечание
Текстовый До 255	TEXT, CHAR, VARCHAR	Размер поля указывается в круглых скобках. Если размер не указан, то по умолчанию он равен 255
Memo До 64 000	LONGTEXT	
Байт От 0 до 255	BYTE	
Целое От -32768 до 32767	SHORT, SMALLINT	
Длинное целое -2147483648 до -2147483648	LONG, INTEGER, INT	
Одинарное с плавающей точкой От $-3{,}4 \times 10^{31}$ до $3{,}4 \times 10^{31}$	SINGLE, REAL	
Двойное с плавающей точкой От $-1{,}797 \times 10^{101}$ до $1{,}797 \times 10^{101}$	DOUBLE, FLOAT, Number, Numeric	
Дата/время	DATETIME	
Денежный	CURRENCY	
Счётчик	COUNTER	
Логический	BIT	
Двоичный	BINARY	Размер поля указывается в круглых скобках. Если размер не указан, то по умолчанию он равен 510
Объект OLE	LONGBINARY	

Описание простого индекса

CONSTRAINT <имя индекса> [**PRIMARY KEY** | **UNIQUE** | **NOT NULL** | **REFERENCES** <имя внешней таблицы> [(<имя поля внешний таблицы>)][**ON UPDATE** {**CASCADE** | **SET NULL**}][**ON DELETE** {**CASCADE** | **SET NULL**}]]

- **PRIMARY KEY** - первичный ключ
- **UNIQUE** - уникальный индекс

- **NOT NULL** - значение индекса не может быть Null
- **REFERENCES** - определение связи таблицы с другой таблицей БД. Имя поле связи, если оно является ключом, можно опустить
- **ON UPDATE CASCADE** - в случае изменения значения поля связи в родительской внешней таблицы соответствующее каскадное обновление будет произведено в подчиненной таблице
- **ON DELETE CASCADE** - в случае удаления строки из родительской (внешней) таблицы все строки подчиненной таблицы, значения поля связи в которых совпадает с соответствующим значением поля удаленной строки, также будут удалены
- **SET NULL** - в случае изменения или удаления строк родительской таблицы соответствующим внешним ключам в подчиненной таблице, структура которой описывается, будут автоматически присвоены значения NULL

Описание составного индекса

CONSTRAINT <имя индекса> [**PRIMARY KEY** | **UNIQUE** | **NOT NULL**] (<имя поля>,…) | **FOREIGN KEY** [**NO INDEX**] (<имя поля>,…) **REFERENCES** <имя внешней таблицы> [(<имя поля внешней таблицы>,…)] [**ON UPDATE** {**CASCADE** | **SET NULL**}] [**ON DELETE** {**CASCADE** | **SET NULL**}]]

Для создания внешнего ключа используется предложение **FOREIGN KEY**. Перечисляются все поля описываемой таблицы, содержащие ссылки на поля внешней таблицы. Имя внешней таблицы и имена полей указываются после слова REFERENCES. Имена полей внешней таблицы должны быть заданы в том же порядке, что и ссылки на них. Если поля являются ключевыми для внешней таблицы, их можно не указывать.

№	51
Описание	Описание составного, а не простого индекса, хоть он и состоит из одного поля.

SQL	`CREATE TABLE Друзья(`
	`  [КОД] INTEGER NOT NULL,`
	`  [Фамилия] TEXT(20) NOT NULL`
	`  [Имя] TEXT (15)`
	`  [День рождения] Date`
	`  [Телефон] TEXT`
	`  [Примечания] memo,`
	`  CONSTRAINT [Индекс1] PRIMARY KEY ([Код])`
	`);`

Команда CREATE INDEX

Если кроме первичного ключа требуются другие индексы, их можно создать с помощью инструкции CREATE INDEX

CREATE [**UNIQUE**] **INDEX** <имя индекса> **ON** <имя таблицы> (<имя поля> [**ASC** | **DESC**], …) [**WITH** {**PRIMARY** | **DISALLOW NULL** | **INGORE NULL**}]

Необязательное предложение WITH позволяет задать следующие условия на значения:

- С помощью слова **PRIMARY** назначить индексированное поле ключевым. Ключ по умолчанию является уникальные индексом, поэтому **UNIQUE** после **CREATE** можно опустить. Если в таблице уже определён ключ, то нельзя использовать слово **PRIMARY**
- С помощью параметры **DISALLOW NULL** можно запретить значение Null в индексированных полях
- С помощью параметра **IGNORE NULL** запретить включение в индекс записей, имеющих значения Null в индексированных полях.

№	52
Описание	Для таблицы "Друзья" создать составной индекс с именем NewIndex по полям "Фамилия" и "Имя". По умолчанию для обоих полей используется упорядоченность по умолчанию. Такой индекс допускает попадание в таблицу двух людей с одинаковыми именами и фамилиями
SQL	`CREATE INDEX NewIndex` `ON Друзья([Фамилия], [Имя])`

№	53
SQL	`CREATE UNIQUE INDEX NewIndex` `ON Друзья ([Фамилия] DESC, [Имя] DESC);`

Команда CREATE VIEW

Создает новое представление (именованный запрос)

Синтаксис:

CREATE VIEW <имя представления> **AS** <команда **SELECT**>

Команда SELECT должна быть простой.

Допускается:

- Задание условия отбора
- Использование группировки

Не допускается:

- Сортировка
- Наличие параметров

Если запрос, определенный с помощью инструкции SELECT, является обновляемым, то и созданное представление является обновляемым.

№	54

SQL	`CREATE VIEW Запрос54`
	`AS SELECT DIS, GR`
	`FROM Dis_Gr`
	`WHERE PR ="Сидоров С.С.";`

№	55
SQL	CREATE VIEW Запрос55 AS SELECT CATEG, AVG (NAGR) AS [средняя нагрузка] FROM Prep GROUP BY CATEG;

Команда CREATE PROCEDURE

Команда создания **хранимой процедуры** - процедуры, которые используются в клиент-серверных приложения. Они хранятся на сервере в откомпилированном виде, что ускоряет доступ к данным, позволяют централизовать бизнес-логику приложения.
Фактически эта команда позволяет создать именованный запрос с параметрами
CREATE PROCEDURE <имя процедуры> **AS** <инструкция SQL>

Инструкция SQL - это одна из команд языка, например, SELECT, UPDATE и т.д.
Т.е. Команда позволяет создавать не только именованные запросы на выборку данных, но и именованные запросы других типов.

№	56
SQL	CREATE PROCEDURE Запрос56 AS SELECT DIS, GR, PR FROM Dis_Gr WHERE PR = [ФИО преподавателя?];

№	57
SQL	CREATE PROCEDURE Запрос57 AS UPDATE Prep SET Prep.NAGR = [NAGR]*1.1 WHERE Prep.CATEG = "ассистент";

№	58
SQL	Create PROCEDURE Запрос58 as DROP VIEW Запрос57;

Команда ALTER TABLE

Изменяет структуру существующей таблицы, позволяет:

- Добавить в таблицу определение нового поля
- Удалить поле из таблицы
- Изменить для какого-либо поля его тип и размер, запретить использование Null в качестве значения поля
- Добавить или удалить индекс таблицы

ALTER TABLE <имя таблицы> {**ADD COLUMN** <поле>,… | **ALTER COLUMN** <поле> | **ADD** <составной индекс> | **DROP COLUMN** <имя

поля>,… | **DROP CONSTRAINT** <имя индекса>}
Можно добавить или удалить несколько полей одной командой. Но если требуется одно поле добавить, а другое удалить, необходимо последовательно выполнить две команды **ALTER TABLE**. Нельзя добавить или удалить одновременно несколько индексом, а также изменить описание сразу нескольких полей

№	59
SQL	`ALTER TABLE Друзья` `ADD COLUMN B text not null, C text;`

№	60
SQL	`ALTER TABLE Друзья` `ALTER COLUMN C text (30) not null;`

№	61
SQL	`ALTER TABLE Друзья` `Drop COLUMN C;`

№	62
SQL	`ALTER TABLE Друзья` `ADD CONSTRAINT primarykey primary key ([Код]);`

Команда DROP
Позволяет удалить:

- Индекс таблицы
- Таблицу из базы данных
- Представление или хранимую процедуру

DROP {**TABLE** <имя таблицы> | **INDEX** <имя индекса> **ON** <имя таблицы> | **PROCEDURE** <имя процедуры> | **VIEW** <имя представления>}

Если после ключевого слова TABLE указать имя представления, а не таблицы, ошибки не будет. Поскольку процедура, созданная командой CREATE PROCEDURE, хранится в БД как представление, то для удаления и процедуры и представления можно использовать любую из команд: DROP VIEW или DROP PROCEDURE

№	63
SQL	`Drop Index NewIndex ON Друзья`

№	64
SQL	`Drop Table Друзья;`

№	65
SQL	`Drop Table Запрос54`

№	66
SQL	`Drop VIEW Запрос54;`

№	67
SQL	`Drop PROCEDURE Запрос54`

Команды управления доступом к данным

Это команда GRANT, предоставляющая конкретные привилегии существующему пользователю или группе пользователей, и команда REVOKE, эти привилегии отменяющая

Команда GRANT

```
GRANT {<привилегия> [, …]} ON {TABLE <имя таблицы> |
OBJECT <имя объекта> | CONTAINER <имя контейнера> } TO
{<имя правообладателя> [, …]}
```

Команда REVOKE

```
REVOKE {<привилегия> [, …]} ON {TABLE <имя таблицы> |
OBJECT <имя объекта>  | CONTAINER <имя контейнера> } FROM
{<имя правообладателя> [, …]}
```

Объект БД – это все то, что не является таблицей или представлением, например, форма, отчет, макрос. Большинство объектов БД - контейнеры. Допустимыми именами контейнеров являются, например, Tables, Forms, Reports.

Список привилегий

Привилегии	Возможности
SELECT	Читать строки
DELETE	Удалять строки
INSERT	Добавлять строки
UPDATE	Обновлять строки
DROP	Удалять объект

SELECTSECURITY	Просматривать опции защиты объекта
UPDATESECURITY	Изменять опции защиты объекта
DBPASSWORD	Изменять пароль БД
UPDATEINDENTITY	Обновлять учётные записи
CREATE	Создавать новые объекты
SELECTSCHEMA	Читать структуру объекта
SCHEMA	Изменять структуру объекта
UPDATEOWNER	Менять владельца объекта

Привилегии не являются независимыми друг от друга. Например, разрешение обновлять строки таблицы автоматически дает разрешение на чтение строк и чтение макета этой таблицы.

Правообладатель

Правообладатель - это пользователь или группа. Пользователи объединяются в группы для удобства администрирования БД.
Учетные записи пользователей и групп, а также пароли пользователей и данные принадлежности пользователей к группам хранятся в файле рабочей группы - зашифрованной БД. В многопользовательской среде это файл располагается на файл-сервере и имеет имя по умолчанию SYSTEM.MDW. Файл рабочей группы по умолчанию содержит учётные записи двух групп - Admins и Users и одного пользователя - Admin.
Члены **группы Admins** имеют самые широкие права доступа ко всем объектам БД и возможность управлять учётными записями пользователей и групп. Однако членство пользователя в группе Admins может быть отменено другими членами это группы.
Пользователь Admin, в отличие от группы Admins, никакими особыми привилегиями не обладает, автоматически включается в группу Users, а не в группу Admins. По умолчанию группа Users получает права на полный доступ ко всем создаваемым объектам.
Сведения о привилегиях хранятся в каждой БД. Итоговый набор привилегий пользователя - это комбинация его собственных привилегий и привилегий групп, в которые он входит, по каждому объекту пользователь получает максимально высокие привилегии из перечня возможных.

Пользователи

Создание пользователя

CREATE USER <имя пользователя> <пароль> <код учётной записи> [,<имя пользователя> <пароль> <код учётной записи>,…]

№	68

SQL	`CREATE USER Anna avk 123whatR`

Пользователь может изменить свой пароль, но код учётной записи (PID - Personal Identifier) может менять только администратор.
Пользователь может быть владельцем БД и объектом БД
Пользователь может быть владельцем БД и объектов БД, имеет набор привилегий, личный идентификатор и пароль, может войти в систему, используя свое имя.

Удаление пользователя
Используется для удаления учётной записи одного или нескольких пользователей и для исключения пользователей из группы. Если в команде используется ключевой слово FROM, то каждый перечисленный в ней пользователь будет исключен из группы.
DROP USER <имя пользователя> [,…] [**FROM** <имя группы>]

№	72
SQL	`DROP USER Anna`

Учетная запись Admin не может быть удалена, но пользователь Admin может быть удален из группы Users, если в ней останется хотя бы один член.

Изменение пользователя
ALTER USER <имя пользователя> **PASSWORD** <новый пароль> <старый пароль>

№	74
SQL	`ALTER USER Anna PASSWORD KAV avk`

Группы пользователей
Создание группы
CREATE GROUP <имя группы> <код учётной записи> [, <имя группы> <код учётной записи>,...]

№	69
SQL	`CREATE GROUP Programmers crazy8s`

Пользователь и группа не должны иметь одинаковые имена. В основном группы имеют те же возможности, что и пользователи, но не могут владеть БД. Нельзя войти в систему, используя учётную запись группы.

Для добавления пользователей к существующей группе предназначена команда ADD USER

ADD USER <имя пользователя> [, …] **TO** <имя группы>

№	70
SQL	`AAD user Anna TO Users;`

№	71
SQL	`DROP USER Anna FROM Users`

Удаление группы

Удаляет одну или несколько существующих групп. Это действие, однако, не затрагивает пользователей – членов удаленных групп, они остаются пользователями, хотя перестают быть членами удаленной группы.

DROP GROUP <имя группы>[, …]

№	73
SQL	`DROP GROUP Programmers`

Примеры использования GRANT и REVOKE

№	75
Пояснение	Дает разрешение пользователю с именем учетной записи Anna читать данные из таблицы «Друзья», пополнять таблицу новыми данными, удалять строки таблицы.
SQL	`GRANT DELETE, INSERT ON TABLE Друзья TO Anna`

№	76
Пояснение	Лишает пользователя Anna возможности добавления новых строк в таблицу «Друзья»:
SQL	`REVOKE INSERT ON TABLE Друзья FROM Anna;`

№	77
Пояснение	Демонстрирует использование инструкции GRANT с ключевым словом CONTAINER для установки разрешения на чтение структуры вновь создаваемых таблиц для группы Programmers:
SQL	`GRANT SELECT SCHEMA ON CONTAINER TABLES TO Programmers;`