МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МИНСКИЙ ГОСУДАРСТВЕННЫЙ ВЫСШИЙ

РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

М.В.ВАСИЛЕВСКАЯ, Е.А. ЛАЗИЦКАС

**СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ**

Лабораторный практикум

для студентов специальностей 1-08 01 01

«Профессиональное обучение»

В двух частях

ЧАСТЬ 2

МИНСК 2011

**Предисловие**

Современный специалист в области разработки СУБД и клиент-серверных приложений должен уметь пользоваться различными технологиями получения данных из СУБД, а также различными методами и средствами разработки клиент-серверных приложений различного уровня сложности.

Целью дисциплины «Системы управления базами данных» является изучение основ реляционной модели баз данных, принципам их создания и обработки данных, дляполучение необходимой информации.

Лабораторный практикум по данной дисциплине предполагает выполнение 24 лаб раб.

Данный практикум является продолжением первой части.

В первой части содержаться задания по разделам «Разработка БД в среде MSAccess», «Администрирование БД Interbase».

Вторая часть практикума включает задания по разделам «Создание БД MySQL», «Создание БД Oracle», «Создание клиент-серверных приложений», «Работа с транзакциями».

Описание каждой лабораторной работы содержит теоретические сведения, необходимые для её выполнения, порядок выполнение работы, задания и контрольные вопросы

При защите лабораторной работы необходимо предоставить отчет выполненной работы, продемонстрировать результат выполнения работы и ответить на контрольные вопросы

**Указания по выполнению лабораторных работ**

Перед выполнением лабораторной работы студенты должны ознакомиться с предлагаемыми теоретическими сведениями, необходимыми для успешного выполнения конкретной работы. Следует внимательно ознакомиться с заданием, обдумать действия по выполнению работы, изучить при необходимости дополнительный теоретический материал, а затем приступить к выполнению задания.

Результаты выполнения всех лабораторных работ необходимо сохранить на магнитном носителе, так как они будут использоваться для выполнения последующих работ.

Отчет по лабораторной работе включает:

- тему лабораторной работы;

- цели работы;

- пошаговое описание, выполненных заданий;

- ответы на контрольные вопросы по лабораторной работе;

- выводы по итогам лабораторной работы.

К выполнению следующей работы студенты допускаются только после сдачи отчета по предыдущей лабораторной работе.

**Лабораторная работа 1**

**УСТАНОВКА SQL-СЕРВЕРА, ПОДКЛЮЧЕНИЕ КЛИЕНТСКОЙ ПРОГРАММЫ**

Цель работы: формирование умений установки на персональный компьютер и конфигурировать сервер баз данных MySQLи закрепление знаний, умений и навыков создания объектов БД.

**1.1. Краткие теоретические сведения**

MySQL – это система управления базами данных (СУБД). База данных (БД) представляет собой структурированную совокупность данных. СУБД применяется для записи, выборки и обработки данных, хранящихся в компьютерной БД. В реляционной БД информация хранится в отдельных таблицах, благодаря чему увеличиваются скорость и гибкость работы СУБД.

**1.2. Описание команд и рекомендации к выполнению работы**

Для начала установки сервера MySQL необходимо запустить инсталлятор и следовать инструкциям, появляющимся в окне инсталлятора.

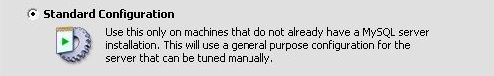
Рекомендуется выбрать типичный (*Typical*) вид инсталляции.



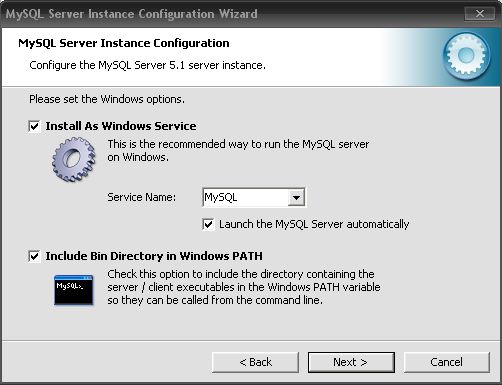
Далее следуем инструкциям инсталлятора, выбираем пункт ConfiguretheMeSQLServernow, для более детальной настройки сервера.



Соглашаемся со стандартной установкой:



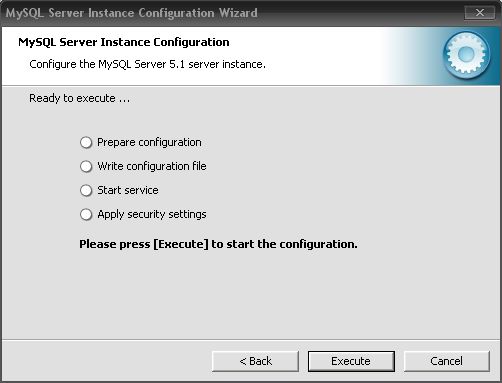
Если отметить “InstallAsWindowsService”, сервер будет запускаться в виде сервиса, что является рекомендуемым способом запуска. Ниже, в ниспадающем списке, задается имя сервиса. Далее, поставьте галочку рядом с “LaunchtheMySQLServerautomatically”, чтобы он запускался автоматически. Также поставьте галочку рядом с “IncludeBinDirectoryinWindowsPATH” – это позволит установить видимость каталога “bin” для командной строки.





Установите пароль главного пользователя root. Не оставляйте это поле пустым, это убережет вас от возможных неприятностей в дальнейшем с безопасностью баз данных.

В данном окне обратите внимание на строку “Writeconfigurationfile”, которая указывает на месторасположение конфигурационного файла MySQL – my.ini.



Сервер MySQL устанавливается в каталог C:\ProgramFiles\ MySQL\MySQLServer 5.1. Все настройки программы хранятся в файле my.ini в каталоге программы. Что бы указать параметры настройки сервера, необходимо изменить директивы файлы my.ini.

Откройте файл настроек и найдите в разделе [mysqld] параметры basedir (каталог с установленным сервером MySQL) и datadir (каталог для хранения БД):

[mysqld]

basedir="C:/Program Files/MySQL/MySQL Server 5.1"

datadir="C:/Program Files/MySQL/MySQL Server 5.1/data"

Изменяя параметр datadir можно указать каталог для хранения БД.

Для соединения с сервером MySQL используется клиентская программа mysql.exe. Для проверки работоспособности сервера MySQL необходимо выполнить несколько команд в терминале Windows (рис. 1). Для запуска терминала нужно нажать «Пуск🡪 Выполнить…», в появившемся окне набрать команду cmd и нажать клавишу Enter.

Для отображения списка существующих баз данных используется команда mysqlshow –uroot -pMyPassworw, где root – привилегированный пользователь, MyPassword – пароль, введенный при инсталляции сервера.

С помощью ключа –u указывается имя пользователя (username), под которым необходимо подключиться. Если имя пользователя не указано, по умолчанию будет использоваться имя, под которым был выполнен вход в операционную систему.

Ключ –p сообщает серверу о том, что вы хотите соединиться с использованием пароля. Включать пароль в эту строку не обязательно – сервер MySQL запросит его самостоятельно.

Информацию о состоянии и версии сервера можно получить при помощи команды mysqladmin –uroot –pMyPasswordversionstatusproc.

Команда mysql –uroot –p запускает клиентскую программу для работы с сервером БД. Клиентская программа устанавливает сетевое соединение с сервером БД и позволяет вводить инструкции SQL в интерактивном режиме для получения необходимой информации из БД.

*В этой и всех последующих лабораторных работах все команды необходимо вводить в окне клиентской программы.*

При успешном входе в терминале отобразится примерно следующая информация:

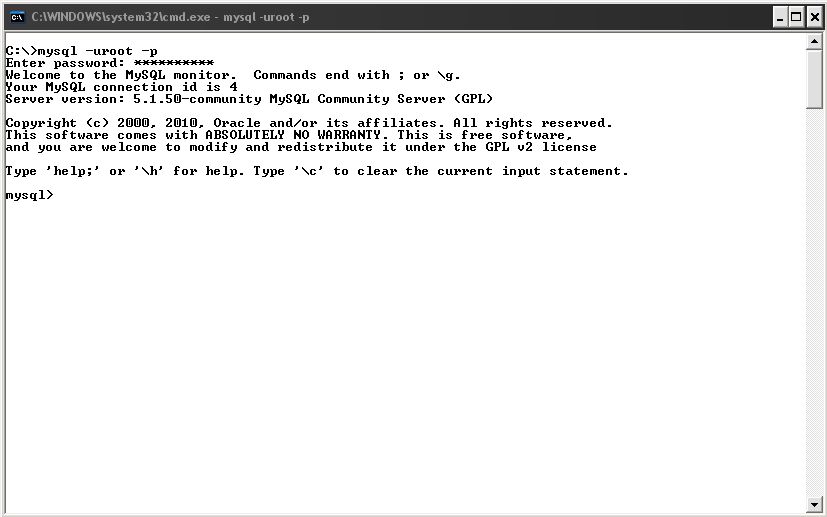


Рис. 1. Вид окна терминала с результатом выполнения команд   
для проверки работы СУБД MySQL

Для отображения списка существующих баз данных используется команда showdatabases; Как вы увидите дальше, примеры команд завершаются точкой с запятой (;), которая сообщает MySQL о том, что команду необходимо выполнить. Если точку с запятой не поставить, ничего не произойдет.

Указать базу данных, с которой будет производиться работа, можно при помощи команды use:

use*dbname;*

После этой команды MySQL должен вывести следующую строку:

Database*changed*

Можно и не вводить команду use, но тогда база данных должна быть указана во время входа в систему:

mysql –Ddbname –uusername -pMyPassword

Создание базы данных представляет наиболее простую задачу. Введите в командной строке MySQL:

createdatabase*dbname;*

Ответ должен выглядеть подобно показанному ниже:

Query OK, 1 row affected (0.06 sec)

Это значит, что все действия были выполнены правильно. Если подобного ответа не последовало, убедитесь, что в конце строки присутствует точка с запятой. Удаление БД осуществляется командой

Dropdatabase**dbname;**

Выход из программы осуществляется командой exit.

Пример типичной сессии работы в клиентской программе mysql.exe представлен на рис. 2.

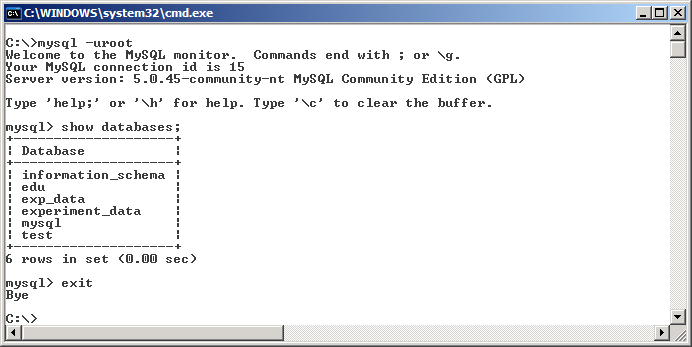


Рис. 2. Вид окна терминала с примером типичной сессии   
работы в клиентской программе mysql.exe

**1.3. Создание таблиц базы данных**

Следующий этап настройки базы данных связан с созданием таблиц.

В СУБД MySQL существует несколько типов таблиц. Основные из них:[MyISAM](http://ru.wikipedia.org/wiki/MyISAM) для быстрого чтения,  [InnoDB](http://ru.wikipedia.org/wiki/InnoDB) для транзакций и ссылочной целостности.

Это делается с помощью SQL-команды

CREATETABLEtablename(columns)

Columns – разделяемый запятыми список столбцов в таблице. Каждый столбец должен иметь имя, за которым следует тип данных.

create table student

( idStudent int unsigned not null auto\_increment primary key,

name char(50) not null,

address char(100) not null,

city char(30) not null,

date date not null

)ENGINE=InnoDB;

Каждая таблица создается с помощью отдельного оператора createtable . Определение столбца содержит его имя, за которым следует тип данных. В определениях некоторых столбцов присутствуют и другие спецификаторы. Например составной ключ можно создать путем написания строки primarykey(fiel1, field2 .. fieldN) в конструкции createtable.

Просмотр таблицы базы данных осуществляется с помощью команды

describe*dbname;*

**2. Практическая часть**

**Порядок выполнения работы**

1) Установить сервер баз данных MySQL на компьютер.

2) Выполнить первичную настройку программы.

3) Проверить работоспособность сервера MySQL.

4) Запустить программу-клиент mysql.exe и подключиться к серверу.

5) Вывести и записать список существующих БД.

6) В зависимости от варианта создать требуемые таблицы.

7) Найти на диске в директории баз данных MySQL каталог, содержащий созданную БД.

**3. Контрольные вопросы**

1) Пояснить термин «реляционная база данных».

2) Где хранятся настройки программы СУБД MySQL?

3) Как можно проверить работоспособность СУБД MySQL?

4) Где хранятся данные СУБД MySQL?

**ЛАБОРАТОРНАЯ РАБОТА 2**

**УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ.КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ БД**

Цель работы:формирование умений и навыков управления доступом к данным; формирование умений копирования и восстановления БД.

1. **Краткие теоретические сведения**

**1.1. Управление доступом к данным**

Система MySQL может содержать много пользователей. Пользователь root из соображений безопасности должен использоваться только для административных целей. Каждый пользователь, которому необходимо работать в системе, должен получить учетную запись и пароль.

Система привилегий MySQL идентифицирует пользователя по имени и сетевому адресу, с которого осуществляется подключение. При поступлении каждого запроса сервер MySQL проверяет, достаточно ли у пользователя привилегий для его выполнения. Например, при считывании строк из таблицы или удалении таблиц из БД сервер в первом случае проверяет, имеется ли у пользователя для этой таблицы привилегия SELECT, а во втором – имеется ли у него для этой базы привилегия DROP.

При определении прав пользователей сервер использует данные таблиц user, db и host из БД mysql. Каждая из этих таблиц включает в себя поля контекста и привилегий.

Поля контекста определяют область действия каждой из записей в таблицах, т. е. контекст, к которому имеет отношение та или иная запись, например, БД или отдельная таблица, для которой устанавливается привилегия.

Поля привилегий определяют доступные действия над данными, заданными контекстом.

**1.2 Создание пользователей и задание им прав**

Создание (добавление) и удаление учетных записей пользователей производится командой CREATEUSER. Синтаксис команды CREATE USER следующий:

CREATE USER <имя пользователя> [IDENTIFIED BY <пароль>];

Удаление учетной записи пользователя производится с помощью команды DROPUSER:

DROP USER <имя пользователя>;

Создание (добавление) и удаление учетных записей пользователей, а также назначение им привилегий осуществляется командами GRANT и REVOKE соответственно.

Основные привилегии СУБД MySQL

|  |  |
| --- | --- |
| Название привилегии | Допустимое действие |
| SELECT | Считывание данных из таблиц |
| INSERT | Вставка данных в таблицы |
| UPDATE | Изменение записей в таблицах |
| DELETE | Удаление записей из таблиц |
| INDEX | Создание или уничтожение индексов |
| ALTER | Модификация структуры таблиц |
| CREATE | Создание таблиц и баз данных |
| GRANT | Предоставление другим пользователям привилегий, которые имеет пользователь |
| DROP  FILE | Удаление баз данных или таблиц  Чтение и запись файлов на сервере с помощью команд LOAD INFILE и SELECT OUTFILE |

* + 1. Синтаксис команд GRANT и REVOKE:

GRANT<тип привилегии> [(<столбцы>)] ОN<таблица>

TO<имя пользователя> [IDENTIFIEDBY<пароль>];

REVOKE<тип привилегии> [(<столбцы>)] ON<таблица>FROM<имя пользователя>;

Название таблицы при назначении привилегий пользователю может задаваться в формате <название базы данных>.<таблица>. Если указать в качестве названия БД или таблицы символ \*, то это будет означать, что привилегия относится ко всем БД или таблицам.

Имена новых пользователей можно добавить, применяя команду GRANT. Учетная запись пользователя admin с паролем somepass, имеющего полный набор привилегий, с правом локального подключения к серверу создается командой:

GRANT ALL PRIVILEGES ON \*.\* TO admin@localhost IDENTIFIED BY 'somepass';

Учетная запись пользователя user с паролем userpass, имеющего привилегии выборки, модификации и вставки данных во всех БД, с правом локального подключения к серверу создается командой:

mysql> GRANT SELECT, UPDATE, INSERT ON \*.\* TO user@localhost IDENTIFIED BY 'userpass';

Чтобы обновить настройки привилегий сервера без перезагрузки применяется команда:

FLUSHPRIVILEGES;

Для проверки работы системы привилегий необходимо подключиться к СУБД под именем созданного пользователя:

c:\>mysql –uadmin –p

Параметр –p означает необходимость ввода пароля при подключении к серверу.

**1.3 Копирование и восстановление БД**

Утилита **mysqldump** позволяет получить дамп содержимого базы данных или совокупности баз для создания резервной копии или пересылки данных на другой SQL-сервер (не обязательно MySQL-сервер). Дамп будет содержать набор команд SQL для создания и/или заполнения таблиц.

Так же **mysqldump** имеет возможность развертывания баз данных из созданного sql-файла.

Пример использования mysqldump для создания дампа базы данных «database» при помощи перенаправления потока в файл «database.sql»:

mysqldump -uroot -hlocalhost -p database > database.sql

**-u** или --**user=...** — имя пользователя

**-h** или **--host=...** — удаленный хост (для локального хоста можно опустить этот параметр)

**-p** или **--password** — запросить пароль

**database** — имя базы данных

**database.sql** — файл для дампа

Для того чтобы сделать дамп несколько баз данных, необходимо использовать параметр **--databases** (или сокращенно **-B**), пример:

mysqldump -uroot -hlocalhost -p -B database1 database2 database3 > databases.sql

А для того чтобы сделать дамп всех баз данных, необходимо использовать параметр **--all-databases** (или сокращенно **-A**), пример:

mysqldump -uroot -hlocalhost -p -A > all-databases.sql

Для того, чтобы развернуть дамп, перенаправляем поток в обратную сторону и развертываем базу данных:

mysql -uroot -hlocalhost -p database < database.sql

Иличерез mysql-console:

mysql> usedatabase;

mysql> source database.sql

**2. Практическаячасть**

**Порядок выполнения работы**

1) Создать несколько пользователей. Делегировать пользователям следующие права:

- права системного администратора с возможностью передавать эти права другим пользователям;

- права системного администратора без права передавать эти права другим пользователям;

- права на добавление и выборку записей из всех таблиц;

- права на удаление, добавление и выборку данных из нескольких таблиц, с указанием полей, которые разрешено изменять.

2) Научиться пользоваться командой, которая ликвидирует права.

3) Сохранить резервную копию БД на съемном носителе для дальнейшего использования.

**3.Контрольные вопросы**

1) Каким образом можно ограничить доступ пользователям к БД?

2) Какие виды ограничений бывают?

3) Каким образом можно ликвидировать права пользователей?

**Лабораторная работа 3**

**Индексы и обзоры.**

Цель работы: получить практические навыки работы с индексами, оптимизации работы запросов. Научиться создавать обзоры.

**1.1. Краткие теоретические сведения**

Индексы применяются для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице начиная с первой записи, пока не будут найдены соответствующие строки. Чем больше таблица, тем больше накладные расходы. Если же таблица содержит индекс по рассматриваемым столбцам, то MySQL может быстро определить позицию для поиска в середине файла данных без просмотра всех данных. Для таблицы, содержащей 1000 строк, это будет как минимум в 100 раз быстрее по сравнению с последовательным перебором всех записей. Однако в случае, когда необходим доступ почти ко всем 1000 строкам, быстрее будет последовательное чтение, так как при этом не требуется операций поиска по диску.

Все индексы MySQL (PRIMARY, UNIQUE, и INDEX) хранятся в виде B-деревьев. Строки автоматически сжимаются с удалением пробелов в префиксах и оконечных пробелов.

Индексы используются для того, чтобы:

* Быстро найти строки, соответствующие выражению WHERE.
* Извлечь строки из других таблиц при выполнении объединений.
* Производить сортировку или группирование в таблице.

Предположим, что вызывается следующий оператор SELECT:

mysql>SELECT \* FROMtbl\_nameWHEREcol1=val1 ANDcol2=val2;

Если по столбцам col1 и col2 существует составной индекс, то соответствующие строки могут выбираться напрямую. В случае, когда по столбцам col1 и col2 существуют раздельные индексы, оптимизатор пытается найти наиболее ограничивающий индекс путем определения, какой индекс найдет меньше строк, и использует данный индекс для выборки этих строк.

MySQL применяет индексы также для сравнений LIKE, если аргумент в выражении LIKE представляет собой постоянную строку, не начинающуюся с символа-шаблона. Например, следующиекоманды SELECT используютиндексы:

mysql> SELECT \* FROM tbl\_name WHERE key\_col LIKE "Patrick%";

mysql> SELECT \* FROM tbl\_name WHERE key\_col LIKE "Pat%\_ck%";

В первой команде рассматриваются только строки с "Patrick" <= key\_col < "Patricl", а во второй - только строки с "Pat" <= key\_col < "Pau".

Следующие команды SELECT не будут использовать индексы:

mysql> SELECT \* FROM tbl\_name WHERE key\_col LIKE "%Patrick%";

mysql> SELECT \* FROM tbl\_name WHERE key\_col LIKE other\_col;

В первой команде величина LIKE начинается с шаблонного символа. Во второй команде величина LIKE не является константой.

**1.2. Создание индекса**

Общий вид создания индекса:

CREATE [UNIQUE|FULLTEXT] INDEX index\_name

ON tbl\_name (col\_name[(length)],... )

Обычно все индексы создаются в таблице во время создания самой таблицы командой CREATE TABLE.

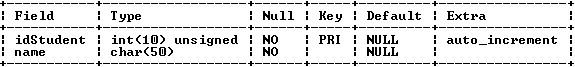
CREATE INDEX дает возможность добавить индексы к существующим таблицам.

Также индекс можно создать следующей командой:

ALTER TABLE tbl\_name ADD INDEX (field)

где field – наименование поля.

Рассмотрим пример. Пусть имеется таблица Student с полями idStudent (PK) и name.



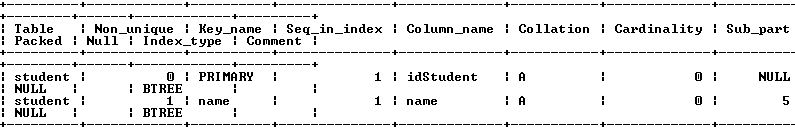
Изописание видно, что в таблице уже имеется индекс типа PRIMARY, созданный автоматически при создании таблицы. Предположим, что необходим частый поиск по имени студента, для чего следует создать индекс:

ALTER TABLE student ADD INDEX (name(5));

В данном примере 5 – длина индекса. Здесь не следует пренебрегать данным параметром во избежание экономии памяти, т.к. чем больше длина индекса, тем больше места занимает таблица на жестком диске. Тем более меньшая длина индекса гораздо повысит скорость операции INSERT.

Проверяем созданный индекс:

SHOW INDEX FROM student;



После этого поиск, сортировка по имени студента будут происходить гораздо быстрее.

В СУБД MySQL удаление индексов производится операторами DROP INDEX или ALTER TABLE. Удаление первичных ключей (индексов PRIMARY KEY) осуществляется только с помощью оператора ALTER TABLE. Операторы удаления индексов имеют следующий синтаксис:

DROPINDEX<index\_name>ON<bl\_name>;

ALTER TABLE <tbl\_name> DROP INDEX <index\_name>;

Оператор для удаления индекса PRIMARY KEY имеет синтаксис:

ALTER TABLE <tbl\_name> DROP PRIMARY KEY;

Если такой индекс с именем PRIMARY KEY создан не был, а таблица имеет один или несколько индексов UNIQUE, будет удален первый из них.

Удаление столбцов из таблицы влияет на индексы. Удаляя столбец из таблицы, вы тем самым удаляете этот столбец из индекса. Удаляя все индексируемые столбцы из таблицы, вы удаляете весь индекс.

**1.3. Оператор EXPLAIN**

EXPLAIN может в точности рассказать вам, что происходит, когда вы выполняете запрос. Эта информация позволит вам обнаружить медленные запросы и сократить время, затрачиваемое на обработку запроса, что впоследствии может значительно ускорить работу вашего приложения.

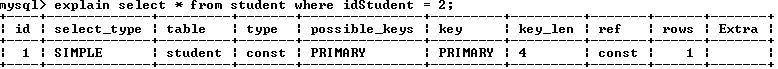
Оператор EXPLAIN можно применять двумя способами:

1. EXPLAIN tbl\_name; (идентично describe tbl\_name;)
2. Применение оператора EXPLAIN перед SELECT запросами.

Рассмотрим второй способ. Простейший пример использования:

EXPLAIN SELECT \* FROM student WHERE idStudent = 2;

В этом примере производится выборка данных студента на основе его идентификатора (idStudent). Вот то, что мы имеем в результате выполнения запроса EXPLAIN:



Id: Идентификатор (ID) таблицы в запросе. EXPLAIN создает по одной записи для каждой таблицы в запросе.

Select\_type: simple. Возможныезначения: SIMPLE, PRIMARY, UNION, DEPENDENT UNION, SUBSELECT, и DERIVED. Это тип запроса.

* SIMPLE – обычныйтипзапросаSELECT.
* PRIMARY – внешний (первый) запрос, в котором используются подзапросы и соединения.
* UNION – второй или последний запрос в соединении
* DEPENDENTUNION - второй или последний запрос в соединении, зависящий от первичного запроса
* SUBQUERY – внутренний подзапрос.
* DEPENDENTSUBQUERY – внутренний подзапрос, зависящий от первичного запроса
* DERIVED – подзапрос, использованный в выражении FROM.

Table. Имя таблицы, из которой MySQL читает данные.

Type. Тип объединения, которое использует MySQL. Возможные значения: eq\_ref, ref, range, index, или all. (const – таблица в запросе считывается только однажды)

Possible\_keys. Список индексов (или NULL, если индексов нет), которые MySQL может использовать для выборки рядов в таблице.

Key. Название индекса, который использует MySQL (после проверки всех возможных индексов).

Key\_len. Размер ключа в байтах.

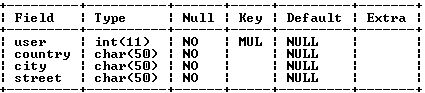
Ref. Колонки или значения, которые используются для сравнения с ключем.

Rows. Количество рядов, которые MySQL необходимо проверить, для обработки запроса(!)

Extra. Дополнительная информация о запросе.

Этот пример достаточно прост. Мы производим поиск по первичному ключу (idStudent) и может быть только одна запись, которая подойдет нашим условиям (переменная rows равна 1).

Рассмотрим более расширенный пример. Создадим еще одну таблицу address, содержащую место проживания студентов.



EXPLAIN SELECT \*

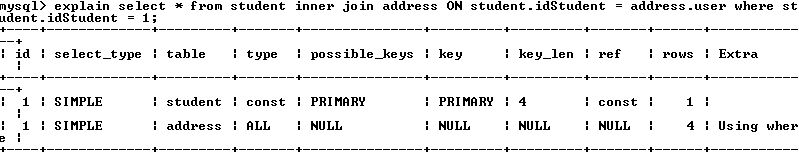
FROM student

INNER JOIN ADDRESS

ON student.idStudent = address.user

WHERE student.idStudent = 1;

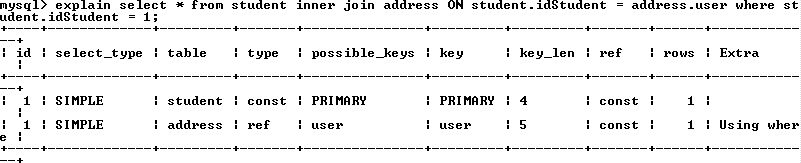
Наблюдаем результат, и видим, что во второй таблице не используется индекс, и она просматривается вся при соединении. При большом количестве данных это очень критично.



Добавим индекс во вторую таблицу для поля user.

ALTER TABLE address ADD INDEX (user);

Наблюдаем результат:



Теперь поле проиндексировано и просматривается всего лишь 1 запись из присоединяемой таблицы.

**1.4. Представления (обзоры)**

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базы данных, определенного с помощью оператора SELECT, в момент обращения к представлению. Представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу.

Для создания представления используется оператор CREATE VIEW, имеющий следующий синтаксис:

CREATE  [OR REPLACE]  
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  
    VIEW view\_name [(column\_list)]  
    AS select\_statement  
    [WITH [CASCADED | LOCAL] CHECK OPTION]

где view\_name - имя создаваемого представления. select\_statement - оператор SELECT, выбирающий данные из таблиц и/или других представлений, которые будут содержаться в представлении

Оператор CREATE VIEW содержит 4 необязательные конструкции:

1. OR REPLACE - при использовании данной конструкции в случае существования представления с таким именем старое будет удалено, а новое создано.
2. ALGORITM - определяет алгоритм, используемый при обращении к представлению. UNDEFINED – автоматический выбор (по умолчанию).
3. column\_list - задает имена полей представления.,
4. WITH CHECK OPTION - при использовании данной конструкции все добавляемые или изменяемые строки будут проверяться на соответствие определению представления. В случае несоответствия данное изменение не будет выполнено. Обратите внимание, что при указании данной конструкции для необновляемого представления возникнет ошибка и представление не будет создано.

При создании представления основанного на нескольких таблицах возможна ситуация повторения представления. Например:

CREATEVIEW v ASSELECT a.id, b.id FROM a,b;

Поэтому необходимо использовать псевдонимы имен:

CREATEVIEW v ASSELECT a.id a\_id, b.id b\_id FROM a,b;

Для просмотра содержимого представления используется оператор SELECT (полностью аналогично как в случае простой таблицы).

SELECT \* FROMv;

**2. Практическая часть**

1) Составить сложные запросы для 3 или более таблиц с использованием INNERJOIN.

2) Проанализировать запросы с помощью оператора EXPLAIN.

3) Оптимизировать запросы добавлением индексов.

4) Зафиксировать результаты.

5) Создать обзоры по БД вашего варианта

**3. Контрольные вопросы**

1) Для чего необходимо обеспечивать уникальность значений ключевых полей?

2) Перечислите виды индексов.

3) Для чего используется оператор EXPLAIN?

4) Что такое обзор?

5) Почему рекомендуется создавать псевдонимы имен в обзорах?

**ЛАБОРАТОРНАЯ РАБОТА 4**

**ТРИГГЕРЫ. ХРАНИМЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ**

Цель работы: формирование умений и навыков создания и модификации триггеров, хранимы процедур и функций.

**1. Краткие теоретические сведения**

**1.1. Создание и модификация триггеров**

Поддержка триггеров включена, начиная с MySQL 5.0.2. Триггер представляет собой именованный объект базы данных, который связан с таблицей, и будет активизирован, когда произойдет определенное событие для таблицы.

Синтаксис команды создания триггера следующий:

CREATE

[DEFINER = {*<пользователь>* | CURRENT\_USER}]

TRIGGER*<имя триггера><время><событие>*

ON*<имя таблицы>*FOREACHROW*<тело триггера>*

Триггер становится связанным с *<имя таблицы>*.

Предложение DEFINER определяет имя пользователя MySQL, которое будет использовать при проверке привилегий доступа в вызове триггера. Значение по умолчанию: пользователь, который выполняет инструкцию CREATETRIGGER, что равнозначно DEFINER = CURRENT\_USER. *<пользователь>*указывается в формате '*user\_name*'@'*host\_name*' (как в команде GRANT).

*<время>* задает время действия. Это может быть BEFORE или AFTER, чтобы указать, что триггер активизируется до или после модификации каждой строки.*<событие>* указывает вид инструкции, которая активизирует триггер. *<событие>*может быть одним из следующего:

INSERT: триггер активизируется, когда новая строка вставляется в таблицу. Например, через команды INSERT, LOAD DATA или REPLACE.

UPDATE: триггер активизируется, когда строка изменяется. Например, через инструкцию UPDATE.

DELETE: триггер активизируется, когда строка удаляется из таблицы. Например, через инструкции DELETE и REPLACE. Однако, инструкции DROP TABLE и TRUNCATE относительно таблицы *НЕ* активизируют триггер, потому что не используют DELETE.

Не может быть двух триггеров для данной таблицы, которые имеют те же самые время действия и событие. Например, не может быть двух триггеров BEFORE UPDATE для таблицы. Номожетбыть BEFORE UPDATE и BEFORE INSERT или BEFORE UPDATE и AFTER UPDATE.

*<тело триггера>* задает инструкцию, которая будет выполнена, когда триггер активизируется. Если необходимо выполнить много инструкций, используется операторная конструкция BEGIN ... END.

Обращаться к столбцам в подчиненной таблице можно используя псевдонимы OLD и NEW. OLD.*<имя столбца>*обращается к столбцу существующей строки прежде, чем она модифицируется или удалится. NEW.*<имя столбца>* обращается к столбцу новой строки, которая будет вставлена, или же к существующей строке после того, как она модифицируется.

Примерсозданиятриггера:

CREATE TABLE test1(a1 INT);

CREATE TABLE test2(a2 INT);

CREATE TABLE test3(a3 INT NOT NULL AUTO\_INCREMENT PRIMARY KEY);

CREATE TABLE test4(a4 INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

b4 INT DEFAULT 0);

DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1

FOR EACH ROW BEGIN

INSERT INTO test2 SET a2 = NEW.a1;

DELETE FROM test3 WHERE a3 = NEW.a1;

UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;

END;

|

DELIMITER ;

Синтаксис удаления триггера:

DROPTRIGGER [*<база данных>*.]*<имя триггера>*

Имя базы данных необязательно. Если оно не задано, триггер удаляется из заданной по умолчанию базы данных. Использование этой инструкции требует привилегии SUPER.

**Замечание**: в настоящее время триггеры не активизируются каскадными действиями внешнего ключа.

**1.2 Создание и модификация хранимых процедур и функций**

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. Хранимые функции являются разновидностью хранимых процедур, основное отличие состоит в том, что хранимые функции могут возвращать скалярное значение и вызываются из оператора точно так же, как и любые другие функции (то есть, через указание имени функции). Процедура вызывается с помощью оператора CALL и может только передавать значения обратно, используя выходные переменные.

Синтаксис создания хранимой процедуры и функции соответственно:

CREATE

[DEFINER = {*<пользователь>* | CURRENT\_USER}]

ROCEDURE <имяхранимойпроцедуры>([[IN | OUT | INOUT] *<имяпараметра><тип>*[,...]])

[<характеристика>...]

<телопроцедуры>

CREATE FUNCTION

[DEFINER = {*<пользователь>* | CURRENT\_USER}]

<имя хранимой процедуры>([*<имя параметра><тип>* [,...]]) [RETURNS <тип>]

[<характеристика> ...]

<тело процедуры>

*<тип>* - любой допустимый тип данных MySQL.

<характеристика>:  
LANGUAGE SQL

| [NOT] DETERMINISTIC

| {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}

| SQL SECURITY {DEFINER | INVOKER}

| COMMENT 'строка'

Характеристики DETERMINISTIC и NOT DETERMINISTIC указывают, производит ли функция всегда тот же самый результат для входных данных. Характеристики CONTAINS SQL, NO SQL, READS SQL DATA и MODIFIES SQL DATA обеспечивают информацию относительно того, читает ли функция или записывает данные. NO SQL или READS SQL DATA указывают, что функция не изменяет данные. Характеристика SQL SECURITY может применяться для определения, должна ли процедура выполняться с использованием привилегий пользователя, создающего эту процедуру, или привилегий пользователя, ее вызывающего. Конструкция COMMENT является расширением MySQL и может использоваться для описания хранимой процедуры. Такая информация отображается операторами SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION.

По умолчанию характеристика имеет следующие значение:

NOT DETERMINISTIC

CONTAINS SQL

SQL SECURITY DEFINER

COMMENT ''

<тело процедуры>состоит из допустимой инструкций хранимого кода SQL. Это может быть простая инструкция типа SELECT или INSERT, либо это может быть составная инструкция, использующая BEGIN и END. Составные инструкции могут содержать объявления, циклы и другие инструкции управляющей структуры.

Вызов процедуры производится командой CALL:

CALL <имя хранимой процедуры>( [<параметр> [,...]])

Инструкции ALTER PROCEDURE и ALTER FUNCTION используются, чтобы изменить характеристики сохраненной процедуры или функции.

ALTER {PROCEDURE | FUNCTION} <имя хранимой процедуры> [<характеристика>...]

Изменить аргументы или тело функции и процедуры нельзя. По этому используйте конструкцию:

DROP {PROCEDURE | FUNCTION} [IFEXISTS] <имя хранимой процедуры>

Эта инструкция используется, чтобы удалить сохраненную процедуру или функцию. То есть определенная подпрограмма будет удалена с сервера.

Предложение IFEXISTS является расширением MySQL. Это предотвращает ошибку, если процедура или функция не существует.

**1.3Синтаксис хранимого кода**

Для использования множественных инструкций необходимо, чтобы была возможность посылать строки запросов, содержащие разделитель операторов ;. Добиться этого можно путем применения команды DELIMITER в командной строке клиента mysql. Замена завершающего запрос разделителя ; (например, на разделитель $$) позволяет использовать ; в теле процедуры.

Хранимый код может включать множественные инструкции, благодаря составному оператору BEGIN…END, который и определяет блок инструкций.

[<*метка начала*>:] BEGIN

[<*блок инструкций*>]

END [<*метка конца*>]

Значения <*метка начала*> и <*метка конца*>, если оба заданы, должны быть одинаковыми. Метка позволяет не путать операторы, если они вложены.

Конструкция DECLARE используется для объявления внутренних переменных, создания курсоров и конструкций исключений.

DECLARE*<имя переменной>*[,...] type [DEFAULT*<значение>*]

Эта инструкция используется, чтобы объявить внутренние переменные, переменные, которые определяются для процедур, функций, триггеров. Чтобы задать значение по умолчанию для переменной, надо указать DEFAULT. Значение может быть определено как выражение, оно не обязательно должно быть константой. Если DEFAULT отсутствует, начальное значение NULL, type, здесь, подобен стандартным типам данных.

Пример:

DECLAREaINTDEFAULT 5;

Курсоры – это указатели на ресурс выборки. Курсоры существуют только в хранимой процедуре или функции. Курсоры довольно медленные, поэтому использовать их стоит по необходимости. Создаются курсоры при помощи оператора DECLARE:

DECLARE *<имякурсора>* CURSOR FOR *<select выражение>*

Эта инструкция объявляет курсор. Каждый курсор должен иметь уникальное имя. *<select выражение>*– SELECT, не может иметь оператор INTO.

Инструкция OPEN открывает предварительно объявленный курсор:

OPEN *<имя курсора>*

Инструкция FETCH выбирает следующую строку (если строка существует), используя определенный открытый курсор, и продвигает указатель курсора:

FETCH*<имя курсора>*INTO*<имя переменной>* [,*<имя переменной>*] ...

Если больше нет доступных строк, происходит условие No Data. Чтобы обнаружить это условие, можно установить обработчик для этого:

CLOSE *<имя курсора>*

CLOSE Закрывает предварительно открытый курсор. Если курсор не закрыт, он все равно закроется в конце составной инструкции (обычно по наступлению END), в которой он был объявлен.

Следует обратить внимание: сначала объявляются переменные, потом курсоры:

CREATE PROCEDURE curdemo()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE a CHAR(16);

DECLARE b,c INT;

DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;

DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;

OPEN cur1;

OPEN cur2;

*<блокинструкций>*

CLOSE cur1;

CLOSE cur2;

END

Констукция SELECT … INTO так же позволяет «заполнить» переменную, прямо из таблицы:

SELECT<имя столбца>[,...]

INTO*<имя переменной>* [,...] *<sql выражение>*

Этот синтаксис SELECT сохраняет выбранные столбцы непосредственно в переменные. Следовательно, количество выбранных столбцов слева от INTO должно быть равно количеству переменных справа от INTO.

Пример:

SELECT id,data INTO x,y FROM test.t1 WHERE id\_tm = 5;

Названия переменных не чувствительны к регистру. Важно: имена переменных SQL не должны совпадать с именами столбцов.

В конструкции управления потоком входят IF, CASE, LOOP, WHILE, REPLACE ITERATE и LEAVE. Многие из этих конструкций могут содержат другие инструкции. Такие конструкции могут быть вложены. Например, IF мог бы содержать цикл, который непосредственно содержит WHILE, который в свою очередь включает в себя оператор CASE.

ИнструкцияIF:

IF*<условие>*THEN*<блок операторов>*

[ELSEIF*<условие>*THEN*<блок операторов>*] ...

[ELSE *<блок операторов>*]

END IF

Инструкция CASE:

CASE*<выражение>*WHEN*<значение>*

THEN*<блок операторов>*

[WHEN*<значение>*THEN*<блок операторов>*] ...

[ELSE *<блок операторов>*]

END CASE

или

CASEWHEN*<условие>*THEN*<блок операторов>*

[WHEN*<условие>*THEN*<блок операторов>*] ...

[ELSE *<блок операторов>*]

END CASE

В первом варианте обрабатывается *<блок операторов>*, если *<выражение>*=*<значение>*. Во втором – результат для первого указанного условия *<условие>*, если оно истинно. Если соответствующая величина результата не определена, то обрабатывается *<блок операторов>*, указанное после оператора ELSE. Есличасть ELSE ввыраженииотсутствует, возвращается NULL.

Пример:

SELECT CASE 1 WHEN 1 THEN "one" WHEN 2 THEN "two" ELSE "more" END;

SELECT CASE WHEN 1>0 THEN "true" ELSE "false" END;

SELECT CASE BINARY "B" WHEN "a" THEN 1 WHEN "b" THEN 2 END;

ИнструкцияWHILE:

[<*метка начала*>:]

WHILE*<условие>*DO*<блок операторов>*

END WHILE

[<*метка конца*>]

Операторный список *<блок операторов>* внутри инструкции WHILE будет повторятся до тех пор пока *<условие>* равно true.

Инструкция LEAVE:

LEAVE label

Эта инструкция используется, чтобы из выйти любой указаной конструкции управления потоком данных. Может использоваться внутри BEGIN … END или же конструкций цикла (LOOP, REPEAT, WHILE). Можно сравнить c инструкцией break из С++.

Инструкция REPEAT:

[*<метка начала>*:]

REPEAT*<блок операторов>*

UNTIL *<условие>*

END REPEAT

[*<метка конца>*]

Пример с использование различных конструкций:

CREATE PROCEDURE curdemo()

BEGIN

DECLARE done INT DEFAULT 0; # переменная, определяющая условие выполнения нашего цикла

DECLARE a CHAR(16);

DECLARE b,c INT;

DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1; # создаёмкурсоры

DECLARE cur2 CURSOR FOR SELECT id FROM test.t2;

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1; # статус, когдакурсористощатся

OPEN cur1; # открываемкурсор

OPEN cur2;

REPEAT

FETCH cur1 INTO a, b; # забираем циклом данные из курсоров, пока не кончатся данные

FETCH cur2 INTO c;

IF NOT done THEN # пока done 0

IF b < c THEN INSERT INTO test.t3 VALUES (a,b); # записываемвтаблицунаименьшееиз b и c

ELSE INSERT INTO test.t3 VALUES (a,c);

END IF;

END IF;

UNTIL done END REPEAT;

CLOSE cur1;

CLOSE cur2;

END

1. **Практическая часть**

**Порядок выполнения работы**

1) В зависимости от варианта создать несколько триггеров для различных таблиц (возможные ситуации включения триггеров предусмотреть по собственному усмотрению).

2) Научиться использовать команду удаления триггеров.

3) В зависимости от варианта создать несколько процедур для различных таблиц (возможные ситуации использования процедур предусмотреть по собственному усмотрению).

**3. Контрольные вопросы**

1) С какими объектами БД связываются триггеры?

2) Какие виды инструкций активизируют триггер?

3) Каким образом можно удалить триггер?

4) В чем отличие хранимой процедуры от функции

5) Какие конструкции управления потоками данных вы знаете?

**ЛАБОРАТОРНАЯ РАБОТА № 5**

**СУБД InterBase и технология IBX**

*Цель работы:* получение практических навыков создание БД Interbase и использования компонентов технологии IBX. Формирование навыков и умений настройки настройки BDE и закрепление знаний, умений и навыков создания объектов БД.

1. **Краткие теоретические сведения**

СУБД InterBase является клиент-серверной. Для разработки приложений баз данных на основе данной СУБД интегрированные среды программирования BorlandDelphi и BorlandC++ Builder оснащены всеми необходимыми компонентами, заключенные под общим названиям технология IBX. Однако, для того, чтобы они были доступны на компьютере разработчика необходимо наличие клиентской библиотеки.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вкладка InterBase | | | | | |
| Название | | Основные свойства | | | Комментарии |
| IBDatabase | |  | | | Отвечает за связь с базой данных |
|  | | DatabaseName | | | Имя базы данных и путь к ней |
|  | | Params | | | Параметры подключения к базе: имя поль­зователя и пароль |
|  | | LoginPromt | | | Логическое поле. Отвечает, будет ли ото­бражаться окно ввода имени пользователя и пароля каждый раз, когда происходит соединение с базой данных |
|  | | Connected | | | Логическое свойство. Если содержит True, то связь с базой данных установлена, в противном случае — нет |
| IBTransaction | |  | | | Отвечает за работу всех транзакций для связанной с данным компонентом базой данных |
|  | | DefaultDatabase | | | Содержит имя компонента IBDatabase, транзакциями которого будет осуществляться управление |
| IBQuery | |  | | | Позволяет отправлять запросы к базе данных и получать результат их выполнения |
|  | | DatabaseName | | | Содержит имя компонента IBDatabase. Соответственно все запросы будут отправляться базе данных, прописанной в компоненте IBDatabase |
|  | | SQL | | | SQL-запрос для работы с базой |
|  | | Active | | | Логическое свойство. Когда установлено в True, SQL-запрос выполняется и результат передается в клиентское приложение (туда, откуда он был послан) |
| Вкладка DataAccess | | | | | |
| Название | Основные свойства | | | Комментарии | |
| DataSource |  | | | Служит посредником между компонентами визуализации данных (таких как DBGrid и DbEdit*)*и источниками данных (таких как IВQuery). Без данного компонента увидеть данные на экране не удастся | |
|  | DataSet | | | Содержит имя компонента источника данных (например, IВQuery) | |
| Вкладка DataControl | | | | | |
| Название | Основные свойства | | Комментарии | | |
| DBGrid |  | | Компонент, отображающий содержимое таблицы или результата запроса в виде двухмерной таблицы, разделенной сеткой | | |
|  | DataSource | | Имя компонента-посредника DataSource. | | |
| DBNavigator |  | | Набор кнопок для работы с данными, содержит кнопки перехода между записями таблицы, а также кнопки для подтверждения либо отказа от изменений | | |
|  | DataSource | | Имя компонента-посредника типа DataSource. Указывается для того, чтобы привязать кнопки к определенному набору данных | | |
| Вкладка Standart | | | | | |
| Название | Основные свойства | | Комментарии | | |
| MainMenu |  | | Данный компонент организует главное меню для формы, на которой он расположен | | |
|  | Items | | В свойстве хранятся пункты и подпункты меню | | |
| Button |  | | Обычная кнопка | | |
|  | Caption | | Определяет текст кнопки | | |
|  | Font | | Определяет параметры шрифта текста кнопки | | |

**2. Практическая часть**

**Порядок выполнения работы**

1)Спроектируйте базу данных, состоящую из нескольких таблиц таким

образом, чтобы в многотабличной системе были таблицы со связью 1:1, 1:N

2)Отредактируете несколько записей в базе данных.

3)Отсортируйте данные в базе данных по какому-либо признаку или полю,

записав результаты сортировки.

4)Составьте вторичный индекс из одного, двух или более полей.

**3. Контрольные вопросы**

1)Каким образом, используя компоненты IBX визуализировать содержимое таблиц

2)Как отключить ото­бражения окна ввода имени пользователя и пароля каждый раз, когда происходит соединение с базой данных

3)Какие компоненты осуществляют навигацию по записям БД.

**Лабораторная работа № 6**

**ADO-технология доступа к БД. Файл-серверные СУБД**

Цель работы:познакомиться с ADO-технологией доступа к БД, изучить подключение к файл-серверным СУБД из Delphi через ADO-соединение.

1. **Краткие теоретические сведения**

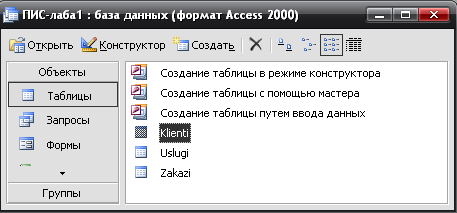
На основе технологии ADO в Delphi созданы соответствующие компоненты-наборы TADOTable, TADOQuery, TADOStoredProc , повторяющие в функциональном отношении компоненты TTable, TQuery, TstoredProc, но не требующие развертывания и настройки на клиентской машине BDE.

Основным достоинством ADO является ее естественная ориентация на создание «облегченного» клиента. На машине сервера данных (это может быть файловый сервер в рамках файл/серверной технологии или машина с сервером данных - в технологии клиент/сервер) устанавливается так называемый провайдер данных - некоторая надстройка над специальной технологией OLE DB, «понимающая» запросы объектов ADO и «умеющая» переводить эти запросы в нужные действия с данными. Взаимодействие компонентов ADO и провайдера осуществляется на основе универсальной для Windows технологии ActiveX, причем провайдер реализуется как СОМ-сервер, а ADO-компоненты - как COM-клиенты. На машине сервера создается и размещается источник данных. В случае файл/серверных систем отдельные таблицы типа dBASE, FoxPro, Paradox и т.п. должны управляться соответствующим ODBC-драйвером,а в роли провайдера используется Microsoft OLE DB Provider for ODBC drivers. Если по каким-либо причинам не найден нужный драйвер файл/серверные таблицы можно перенести в формат MS Access. На их основе создается единый файл, содержащий все необходимые таблицы, индексы, хранимые процедуры и прочие элементы БД. Такой файл управляется машиной баз данных Microsoft Jet 4.0 Database Engine, а в роли провайдера используется Microsoft Jet 4.0 OLE DB Provider. Если используется промышленный сервер данных Oracle или MS SQL Server, данные не нуждаются в какой-либо предварительной подготовке, а в роли провайдера используется соответственно Microsoft OLE DB Provider for Oracle или Microsoft OLE DB Provider for SQL Server. Нетрудно обнаружить и явный недостаток такой технологии: ADO не может использоваться, если для соответствующей структуры данных (в частности, для БД многих популярных серверов - InterBase, Informix, DB2 и пр. не создан нужный провайдер или ODBC драйвер (однако уже существует ODBC драйвер для InterBase – прим. ред.)).

На машине клиента располагаются связные компоненты TADOConnection и компоненты-наборы данных TADOTable, TADOQuery, TADOStoredProc, а также не показанные на рисунке компоненты-наборы TADODataSet и командные компоненты TADOCommand. Каждый из этих компонентов может связываться с провайдером данных либо с помощью связного компонента TADOConnection, либо минуя его и используя собственное свойство ConnectionString. Таким образом, TADOConnection играет роль концентратора соединений с источником данных компонентов-наборов и в этом смысле подобен компоненту TDatabase в традиционной архитектуре с BDE.Компоненты-наборы TADODataSet в функциональном плане повторяют свойства уже известных из предыдущих версий Delphi компонентов TClientDataSet технологии MIDAS. Командные компоненты TADOCommand предназначены для реализации запросов на языке управления данными DDL (Data Definition Language), т.е. для реализации SQL-запросов, которые не возвращают данные (запросы типа CREATE, DROP, UPDATE и т.п.). Специальный компонент RDSConnection (не показан на рисунке) создан для упрощения связи с MS Internet Explorer и предназначен для разработки интранет-приложений. Компоненты-наборы с помощью хорошо известных по предыдущим версиям Delphi компонентов-источников TDataSource и визуализирующих компонентов TDBGrid, TDBMemo, TDBEdit и т.п. обеспечивают необходимый интерфейс с пользователем программы.

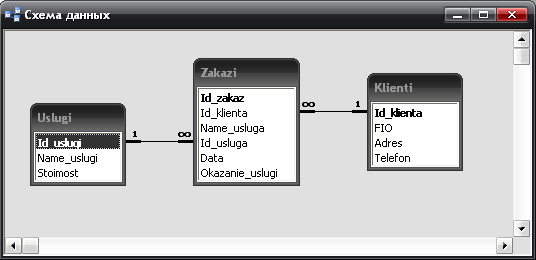
**Этап создания БД**

1. В соответствии с вариантом «Ателье обуви. Оформление заказов» создаем базу данных в MicrosoftAccess (Рис.1).

****

**Рис.1.** Создание таблиц

1. Установливаем связи (Рис.2).

****

**Рис.2.** Установление связей между таблицами

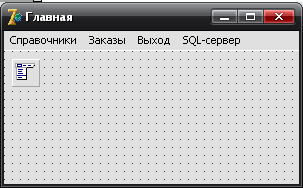
1. Вносим данные в таблицы (Рис.3).



**Рис.3.** Заполнение таблиц

**Этап подключения к БД**

1. Создаем меню из нескольких форм. Для этого размещаем на форме приложения компонент MainMenu и отредактируем его (Рис.4).

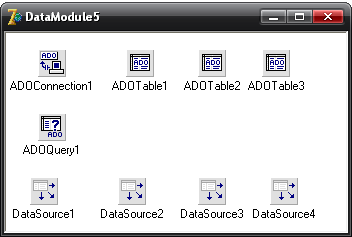


**Рис.4.** Создание меню на главной форме

1. Организовываем доступ к БД через ADOConnection, ADOTable, DataSource. Используя объект DataModule.

На DataModule (File – New – DataModule) разместим следующие компоненты (Рис.5):

* 1 компонент AdoConnection – связывает Delphi с БД
* 3 компонента AdoTable – связывает Delphi с таблицами БД
* 1 компонент AdoQuery –SQL-запрос
* 4 компонента DataSource – связь между набором данных (AdoTable или AdoQuery) и визуальными компонентами (на вкладке DataAccess)



**Рис.5.** Объект DataModule

1. Настраиваем компоненты:

AdoConnection:

* свойство ConnectionString – открывается окно, в нем по нажатию кнопки Build выбираем MicrosoftJet 4.0vOLEDBProvider, потом по кнопке Next – выбираем нашу БД
* свойство Login Promt: False
* свойство Connected: True

Настраиваемтаблицы (AdoTable1, AdoTable2, AdoTable3, AdoTable4):

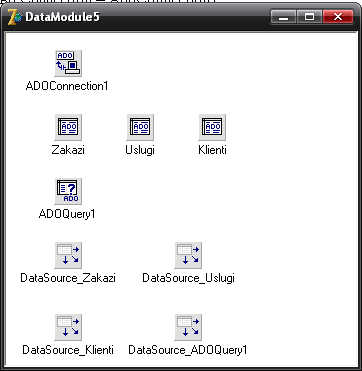
* свойство Connection = AdoConnection1,
* свойство TableName –указать название таблицы, с которыми установлена связь;
* свойство Name – имя по таблице БД.
* свойство Active –True.

Настроиваем запрос AdoQuery:

* свойство Connection = AdoConnection1.

DataSource1, DataSource2, DataSource3, DataSource4, DataSource5:

* свойство DataSet – выбираем соответствующую таблицу;
* свойство Name – имя по таблице БД.



**Рис.6.** Настройка компонентов

**Этап отображения и редактирования данных**

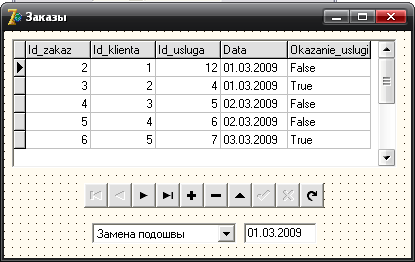
1. Отображаем данные в таблицах, использовав компонент DBGrid, свойство DataSource - DataModule4.DataSourceKnigi и DataModule4.DataSourceAvtori соответственно. Предварительно прописав в разделе implementation «UsesUnit5» (на Unit4 располагается DataModule).
2. На форме Заказы вставляем объекты DBEdit и DBLookUpComboBox (вкладка Data Controls).

Настроиваем DBEdit :

* Edit1: Свойство DataSource–DataModule5.DataSource\_Zakazi
* Свойство Data field – Data

Настроиваем DBLookUpCombo Box (позволяет выйти на подстановочную таблицу, взять из нее данные; также может использоваться для организации списка из полей таблицы).

* DataSource –DataModule5.DataSource\_Zakazi
* Свойство Datafield – Id\_usluga
* ListField – Name\_uslugi
* ListSource – DataModule5.DataSource\_Uslugi
* Key Field – Id\_uslugi

****

**Рис.7.** Объекты DBEdit и DBLookUpComboBox

**Этап фильтрации и сортировки данных**

1. На форме Заказы добавляем фильтрацию по клиентам и услугам. Для этого вставляем компонент CroupBox (Standart), создаем два компонента RadioButton, 2 компонента DBLookUpComboBox, вспомогательный компонент Edit1, кнопку Фильтр.

ЗапрограммируемкнопкуФильтр:

procedure TForm4.Button1Click(Sender: TObject);

begin

if RadioButton1.Checked=True then

begin

Edit1.Text:= DBLookUpComboBox2.Text;

DataModule5.Zakazi.Filter:='Name\_uslugi ='+Edit1.Text;

end;

if RadioButton2.Checked=True then

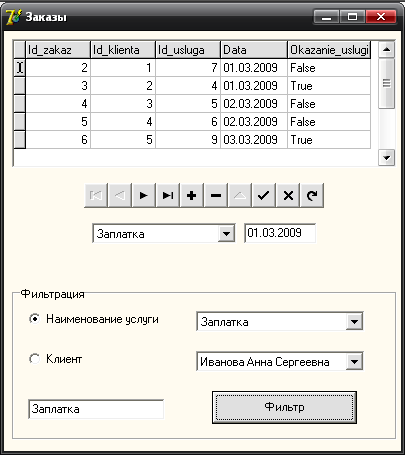
begin

Edit1.Text:= DBLookUpComboBox3.Text;

DataModule5.Zakazi.Filter:='FIO ='+Edit1.Text;

end;

end;



**Рис.8.** Фильтрация

2. На форму Услуги добавиляем объект CroupBox с радиокнопками «По наименованию» и «По стоимости»:

Запрограммируем радиокнопку «По наименованию»:

procedureTForm2.RadioButton1Click(Sender: TObject);

begin

DataModule5.Uslugi.IndexFieldNames:='Name\_uslugi';

end;

Запрограммируем радиокнопку «По стоимости»:

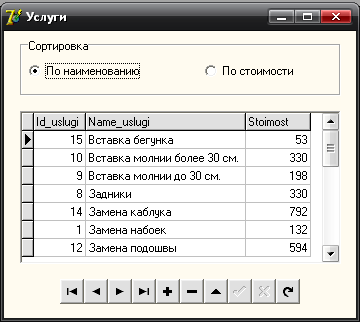
procedure TForm2.RadioButton1Click(Sender: TObject);

begin

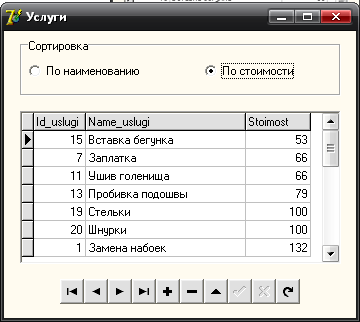
DataModule5.Uslugi.IndexFieldNames:='Stoimost';

end;

Проверяем работоспособность (Рис.9,Рис.10).



**Рис.9.** Сортировка по наименованию



**Рис.9.** Сортировка по стоимости

**Этап работы с SQL-запросами**

1. Построить SQL-запросы:
   * На выборку:

SELECT Id\_uslugi, Name\_uslugi, Stoimost

FROM Uslugi

WHERE (((Uslugi.Stoimost)>100))

ORDER BY Name\_uslugi;

SELECT Id\_zakaz, Id\_klienta, Id\_usluga, Data, Okazanie\_uslugi

FROM Zakazi

WHERE (((Zakazi.Id\_zakaz)>2))

ORDER BY Id\_zakaz;

* + На обновление:

UPDATE Услуги SET Name\_Uslugi = "Стельки"

WHERE Id\_uslugi=19;

UPDATEKlientiSETFIO = "АношкинаАлександраВладимировна"

WHERE Id\_klienta=1;

* + На добавление:

INSERT INTO Услуги( Name\_uslugi, Stoimost )

VALUES ("Шнурки", 200);

INSERT INTO Klienti ( FIO, Adres,Telefon )

VALUES ("Потапова Антонина Сергеевна", "Ленинский 12-96",347091);

* + На удаление:

DELETE \*

FROM Uslugi

WHERE Id\_Uslugi=18;

DELETE \*

FROM Klienti

WHERE Id\_klienta=7.

Построить запрос по трем таблицам с использованием связи между таблицами и сортировки:

SELECTZakazi.Id\_zakaz, Klienti.FIO, Uslugi.Name\_uslugi, Zakazi.Okazanie\_uslugi

FROM Uslugi INNER JOIN (Klienti INNER JOIN Zakazi ON Klienti.Id\_klienta = Zakazi.Id\_klienta) ON Uslugi.Id\_uslugi = Zakazi.Id\_usluga

ORDERBYKlienti.FIO.

Скопировать текст SQL-запроса в буфер обмена для дальнейшей вставки в Delphi. Зайти в Delphi и настроить компонент ADOQuery. В

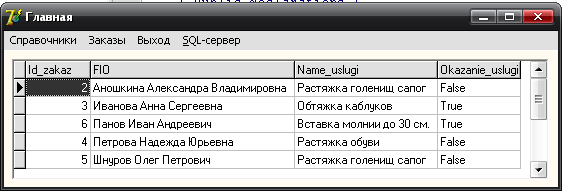
ыбрать на DataModule5 компонент AdoQuery1.

Настроить свойства:

* + свойство Connection = AdoConnection1 (было настроено раньше),
  + свойство SQL – в редакторе SQL текста вставила текст SQL-запроса из буфера обмена,
  + свойство Active –True.

На главную форму вставить DBGrid. Свойство DataSource настроить на компонент DataSourceQuery.

Запустить программу и проверить ее работоспособность (Рис.10).



**Рис.10.**SQL-запрос

Создать подключение базы данных из текущей папки приложения. Для этого:

1. Отключить свойство Connected компонента AdoConnection.

2. Поместить файл базы данных в папку приложения

3. При настройке компонента AdoConnection:

* свойство ConnectionString – открывается окно строки соединения, в нем по нажатию кнопки Build выбираем MicrosoftJet 4.0vOLEDBProvider, потом по кнопке Next – выбираем нашу БД и вместо полного пути с указанием диска и каталогов оставляем только название файла с базой данных.

1. **Практические задания**

**Порядок выполнения работы**

1) создать БД согласно индивидуального задания в среде MSAccess

2) с использованием ADO компонентов подключить БД в среде Delphi

3) создать полнофункциональное приложение с использованием ADO компонентов

**3. Контрольные вопросы**

1) Какие компоненты ADO используются для подключения БД

2) Как осуществляется настройка соединения база данных с ADO.

**ЛАБОРАТОРНАЯ РАБОТА № 7**

**Работа с Firebird иInterbase в Delphi. ADO компоненты**

*Цель работы:* получить практические навыки работы с ADO компонентами.Получение практических навыков создание БД Interbase. Формирование навыков и умений настройки настройки компонентов ADO и закрепление знаний, умений и навыков создания объектов БД.

1. **Краткие теоретические сведения:**

В Delphicуществует несколько способов работы с Interbase и [Firebird](http://www.ibprovider.com/rus/documentation/firebird.html).

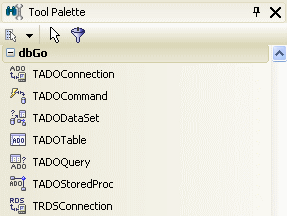
3 способа организации доступа к данным через IBProvider:

* dbGo (ADO Express) компоненты, работающие через библиотеку ADO.
* прямой доступ к COM-интерфейсам ADO, минуя компоненты dbGo.
* прямой доступ к COM-интерфейсам OLEDB при помощи стороннихVCL-компонент (OLEDBDirect/OLEDBExpress).

*Компоненты dbGo*

dbGo - это VCL-компоненты от Borland, позволяющие работать с библиотекой ADO из Delphi и C++ Builder привычным для этих средств разработки способом. До Delphi 6 компоненты назывались *ADOExpress*.

Внешний вид панели компонентов *dbGo* в *Delphi* (RAD Studio 2007):



Состав dbGo-компонентов:

-Компонент *TADOConnection* - подключение к базе данных.

-Компонент *TADODataSet* - базовый набор данных. TADODataSet позволяет загружать данные таблиц, хранимых процедур и произвольных SQL запросов и передавать изменения обратно в БД. Важным условием является то, что выражение указанное в *CommandText*, обязательно должно возвращать набор данных. *TADODataSet* соответствует объекту *ADODB.Recordset*.

- Компонент *TADOTable* является упрощенными вариантом TADODataSet. Он позволяет загружать в память целые таблицы.

- Компоненты *TADOQuery* и *TADOStoredProc* предназначены для выполнения произвольных SQL-запросов и хранимых процедур.

- Компонент *TADOCommand* соответствуют объеку *ADODB.Command*. Фактически объединяет в себе возможности трех компонентов: TADOTable, TADOQuery, TADOStoredProc.

1. **Практические задания**

**Порядок выполнения работы**

Согласно своей предметной область выполните следующие:

1) Подключите БД с помощью ADO компонентов

2)Создайте простейший редактор таблиц на основе *TADODataSet*.

3)Используя компоненты*TDBNavigator*.,*TDBGrid*отобразить содержимое таблиц и навигацию по записям.

4)Для редактирования деталей использовать контейнер*TDBCtrlGrid* в который поместить *TDBLabel*, *TDBText*, *TDBComboBox*.

**Контрольные вопросы:**

1) Приведите схему доступа к данным с применением ADO.

2) Какие компоненты Delphi используются для организации доступа к данным по технологии ADO?

3) Как задаются параметры соединения при разработке в Delphi приложения, использующего технологию ADO?

**ЛАБОРАТОРНАЯ РАБОТА №8**

**РАЗРАБОТКА ПРИЛОЖЕНИЯ В СРЕДЕ DELPHI**

Цель работы: получение практических навыков разработки клиент-серверных приложений в среде Delphi. Формирование навыков и знаний работы с визуальными и невизульными компонентами Delphi. Получить навыки работы с двухзвенной артитектурой.

**1.Краткие теоретические сведения**

Для взаимодействия клиентской программы с данными нужно использовать утилиту, которая обеспечивает клиентскую программу необходимыми данными. Если для создания клиентского приложения выбрана среда Delphi, то в качестве такой утилиты можно использовать BDE.

Утилита BDEAdministrator не является частью приложения. В зависимости от СУБД она может располагаться на машине клиента либо сервера.

Как правило, клиентское приложение использует BDEAdministrator для непосредственного обращения к данным. Кроме того, между приложением и данной утилитой имеется ряд компонентов, которые упрощают разработку программы.

Невизуальные компоненты осуществляют непосредственную работу с BDE, три из которых (Table, Query, StoredProc) служат источниками данных. Визуальные компоненты отображают поставляемые ими данные и помогают создавать удобный интерфейс пользователя. Между источниками данных и визуальными компонентами обязательно располагаются промежуточные компоненты DataSourse, открывающие либо закрывающиеся потоки данных, которыми обмениваются источники данных с визуальными компонентами.

При создании клиентских приложений, которые используют двухзвенную архитектуру, необходимо придерживаться следующих рекомендаций:

На стороне сервера:

-установить сервер БД;

- создать файл БД, который размещается на сервере (определить место размещения БД на диске и задать название БД);

На стороне клиента:

- с помощью утилиты BDEAdministrator задать псевдоним и установить соответствующие настройки для обращения к удаленной БД;

- реализовать клиентcкое приложение, например, средствами Delphi, используя соответствующие компоненты.

**2 Практические задания**

**Порядок выполнения работы**

1. Подготовить клиентское приложение, средствами которого необходимо получать информацию следующего плана:

- осуществлять выборку данных по параметрам заданным пользователем;

- осуществлять выборку данных по определенным признакам (например определенный промежуток времени);

- определить основные максимальные и минимальные финансовые или численные показатели;

- осуществлять выборку данных по определенным признакам (количество признаков ≥ 3)

- определять среднее значение основных показателей;

1. **Контрольные вопросы**

1) Назовите способы обращения к полю записи.

2) Как ограничить диапазон просматриваемых записей?

3)В каких случаях целесообразно отключать визуальные компоненты от источника данных?

4) Для чего используются процедуры Edit, Post, Insert, Append, Delete?

5) Назовите способы обращения к полю записи.

**ЛАБОРАТОРНАЯ РАБОТА №9**

**Проектирование приложений для работы с БД**

Цель работы:получить практические навыки проектирования приложения. Формирование умений проектирования модели приложений и создания приложений.

* 1. **Краткие теоретические сведения**

Одной из самых существенных возможностей пакета Delphi является разработка приложений для работы с БД. Здесь под приложением мы будемпонимать экранную форму, созданную для удобной работы с информацией,хранящейся в таблицах БД.

**Виды форм БД и их создание**

Формы БД различаются по способу представления информации.

Существуют следующие виды форм:

**Список-форма** Она похожа на таблицу, где записи расположены одна под другой. Следует отметить, что для создания такой формы используется компонента DBGrid. Как правило, все поля записи в этой форме размещаются в одну строку. С этой формой очень удобно работать, когда записи короткие

**Бланк-форма**Эта форма похожа на анкету, где все поля одной записи расположены на одном листе. Такая форма используется при отображении

больших записей из БД, например анкетной информации о человеке, и позволяет создать очень информативное представление данных за счет дополнительных элементов.

**Форма с подформой**Представляет собой форму, в которой отображаются два или более связанных между собой наборов данных. Эта форма используется для вывода данных из связанных таблиц. В главной форме отображаются записи из одной таблицы, а в подчиненной - записи из другой таблицы, связанные с текущей записью в главной форме. Этот вариант представления данных удобен, когда одной записи в первом

наборе данных соответствует несколько записей во втором.

* 1. **Практические задания**

**Порядок выполнения работы**

1)Используя исходную БД, спроектируйте экранную форму (приложение),включив в нее все поля и оформив некоторые из них следующим образом:

– комбинированная строка ввода значений поля БД (DBComboBox).

*-*  необходимо вывести в формате, например: 15 Апрель 1998 г.

– набор варианта значения поля БД (DBRadioGroup).

- необходимо вывести в формате, например: 1990 год.

– выключатель для полей БД логического типа (DBCheckBox).

2)Создайте и разместите на форме вычисляемое(ые) поле,

**3. Контрольные вопросы**

1)Компоненты полей. Что это такое?

2) Перечислите основные компоненты полей.

3) Какие поля называют вычисляемыми?

4)Как создать вычисляемое поле?

5)Почему в некоторых случаях вычисляемое поле автоматически не отображается в компоненте DBGrid?

**ЛАБОРАТОРНАЯ РАБОТА №10**

**Проектирование отчетов**

Цель работы:получить практические навыки разработки отчетов. Формирование умений и навыков работ ы с компонентом QuickReport 2.0. Получение практических навыков построения отчетов.

1. **Краткие теоретические сведения**

Отчет – это документ, который распечатывается на принтере, чтобы затем на его основе принимать решения. В Delphi отчет представляет собой специально оформленную информацию из БД и предназначен для наглядного ее отображения в виде особой формы. Его можно просмотреть на экране, а затем распечатать на принтере. Подобная форма содержит специальные компоненты, которые определяют данные и внешний вид отчета при печати. В пакете Delphi 3.0 имеется генератор отчетов **QuickReport 2.0** , который позволяет просматривать, сохранять в файл, читать из файла и печатать отчет.

Отчеты, так же как и формы, могут быть разных типов:

**Список-**выводит информацию набором строк, каждая из которых представляет одну запись.

В **бланке** одна запись размещается на одной странице.

**Отчет с подотчетом** позволяет отобразить информацию из одной таблицы как дополнение к данным из другой

**Отчеты печати почтовых этикеток** позволяет разместить на листе в отчете несколько небольших прямоугольных областей, которые, затем могут быть наклеены на отправляемые по почте вещи.

Стадия подготовки отчета. На этой стадии выбираются компоненты отчета и их параметры.

Стадия создания отчета. На этой стадии из выбиранных компонентов формируется отчет. Выполняется она с помощью генератора отчетов.

Стадия просмотра отчета. На этой стадии можно просмотреть ви полученного отчета на экране и при необходимости внести в него изменения, вернувшись на стадию подготовки отчета.

Стадия печати отчета. На этой стадии осуществляестся вывод отчета на принтер

Структура стандартного отчета QuickReport

Page Footer- Окончание страницы. Полоса появляется в нижней части каждой страницы. Предназначена для отображения служебной информации

Page Header - Заголовок страницы. Полоса размещается в верхней части каждой страницы отчета. Обычно здесь размещается служебная информация об отчете

Title - Заголовок отчета. Полоса располагается после заголовка только на первой странице отчета. Содержит название отчета

Column Header - Заголовок столбца. Может размещаться в произвольных местах отчета, предназначен для выведения общей информации перед началом печати строк данных (шапка таблицы, описание нижеследующих данных и т. д.). В отчете можно размещать несколько таких полос

Detail - Полоса данных. Содержит информацию из заданных оформлением отчета полей набора данных. Таких полос в отчете может быть несколько

Summary - Итоговая полоса. Располагается сразу после полосы данных. Предназначена для размещения общей информации, суммирующей данные (сумма значений колонок или среднее значение и т. д.)

Таблица 1.- Свойства и методы компонента TQuickRep

|  |  |  |
| --- | --- | --- |
| Объявление | Тип | Описание |
| Свойства | | |
| Property AllDataSets: TList; | Pu | Список всех наборов данных, используемых в отчете |
| property Available: boolean; | Ro, Pu | Определяет, готов ли отчет к использованию. При значении False отчет готовится к печати или просмотру |
| property BandList: TList; | Ro | Список имен всех полос отчета |
| property Bands: TquickRepBands | Pb | Определяет набор полос отчета |
| property DataSet: TDataSet; | Pb | Связанный с отчетом набор данных |
| property Description: TStrings; | Pb | Текстовое описание отчета |
| property ExportFilter: TQRExportFilter; | Pu | Определяет компонент TQRExportFilter, используемый для экспорта отчета в файл |
| property Exporting: boolean; | Ro | Возвращает True, если свойство ExportFilter не пустое. Работает только при Available = False |
| property Options: TquickRoportOptions; | Pb | Определяет параметры отчета |
| property Page: TQRPage; | Pb | Определяет параметры страницы |
| property Printer: TQRPrinter; | Pu | Дублирует свойство QRPrinter |
| property PrinterSettings: TQRPrinterSettings; | Pb | Определяет параметры печати отчета |
| property PrintlfEmpty: boolean; | Pb | При значении True разрешает печатать пустой отчет |
| property RecordCount: integers- | Pu | Возвращает число записей в связанном с отчетом наборе данных. Для запроса SQL возвращает 1 |
| property RecordNumber: integers; | Pu | Возвращает номер печатаемой строки |
| property ReportTitle: strings- | Pb | Краткое текстовое описание отчета. Используется для идентификации отчета при поиске в очереди на печать принтера |
| property ShowProgress: boolean; | Pb | При значении True во время печати отчета отображается индикатор состояния процесса |
| property Units: TQRUnits | Pb | Определяет единицу измерения отчета |
| property Zoom: integer; | Pb | Задает масштаб отображения отчета |
| Методы | | |
| procedure ExportToFilter (Afilter:TQRExportFilter); | Pu | Сохраняет отчет в формате, который определяется экземпляром класса фильтра, переданного в параметре |
| procedure NewColumn; | Pu | Осуществляет переход на следующую колонку или страницу (если отчет простой) |
| procedure NewPage; | Pu | Осуществляет переход на следующую страницу отчета |
| procedure Prepare; | Pu | Создает отчет |
| procedure Preview; | Pu | Осуществляет предварительный просмотр отчета |
| procedure Print; | Pu | Осуществляет печать отчета |
| procedure PrinterSetup; | Pu | Открывает стандартный диалог настройки принтера |
| procedure PrintBackground; | Pu | Осуществляет фоновую печать отчета в отдельном потоке |
| procedure Re set PageFooter-Size ; | Pu | Восстанавливает стандартные размеры окончания страницы |
| Методы-обработчики событий | | |
| property AfLerPreview: TQRAfterPreviewEvent; | Pb | Вызывается после закрытия окна предварительного просмотра отчета |
| property AfterPrint: TQRAfterPrintEvent ; | Pb | Вызывается после завершения печати отчета |
| property BeforePrint: TQRDeforePrintEvent; | Pb | Вызывается перед началом печати или просмотра отчета |
| property OnEndPage: TQREndPageEvent; | Pb | Вызывается при окончании создания очередной страницы отчета |
| property OnNeedData: procedure (Sender: Tobject; MoreData: boolean) | Pb | Вызывается при переносе данных из набора данных в отчет при его подготовке. Параметр MoreData при значении False прерывает этот процесс |
| property OnPreview: procedure (Sender: TObject) | Pb | Используется при необходимости заменить стандартный предварительный просмотр на другой |
| property OnStartPage: procedure (Sender: Tobject); | Pb | Вызывается при начале создания очередной страницы отчета |

**2 . Практические задания**

**Порядок выполнения работы**

1. Используя приложение, созданное в лабораторной работе спроектируйте и создайте:

- сводный отчет

- вывод таблиц с использованием группировки

- составление отчета с вычисляемым полем

- составление отчета с использованием агрегатных функций

**Контрольные вопросы:**

1) Какие компоненты используются при подготовке отчета

2) Изобразите схему, поясняющую состав простого отчета и его взаимосвязи с набором данных

3) Как осуществляется создание вычисляемых полей в отчете

**ЛАБОРАТОРНАЯ РАБОТА №11**

**Использование запросов при проектировании**

**приложений**

Цель работы: получить практические навыки разработки и подключению запросов. Формирование навыков и умений работы с компонентами *TQuery, TStoredProc.*

1. **Краткие теоретические сведения**

Использование одной компоненты Table позволяет работать только с одной таблицей. Поэтому, когда необходимо работать с несколькими связанными таблицами, нужно создавать соответствующее число компонент Table. В этом случае целесообразно взять компонент Query класса TQuery являющегося потомком класса TDBDataSet. Эта компонента позволяет определить набор данных на основе нескольких таблиц с помощью SQL запроса. Её также удобно применять, когда таблица БД черезмерно велика, в этом случае с помощью запроса можно ограничить набор раассматриваемых данных.

Можно создать оператор DelphiSQL при помощи компоненты Query следующим образом:

- поместить на форму объекты Query, DataSource и связать их вместе;

- с помощью свойства *SQL* ввести указание SQL;

- установить свойство *Acive* в значение True

У класса *TQuery* отметим следующие наиболее важные свойства, которые он добавляет к наследуемым от классов *TDataSet, TBDEDataSet и TDBDataSet:*

Local – определяет расположение таблиц (true – локальные таблицы, false – таблицы на SQL-сервере); свойство только для чтения.

RequestLive – свойство логического типа, определяет возможность изменять набор данных, полученный в результате выполнения запроса. Эта возможность имеется, если свойство имеет значение True и у запросов, которые основаны на одной таблице и не используют сортировку и агрегатные функции. У остальных запросов результат доступен только для чтения независимо от значения данного свойства. В этих случаях можно посмотреть свойство CanModify для того чтобы увидеть, успешен ли запрос, если же нужно откорректировать таблицу с помощью запроса SQL? То следует использовать команду SQLUpdate/

SQL – свойство типа Tstrings, определяет текст SQL-запроса, который используется при выполнении методов Open или ExecSQL.

Unidirectional – определяет направление перемещения курсора по набору данных, полученному в результате выполнения запроса.

UpdateMode – свойтсво типа TUpdateMode, определяет способ обновления записей из промежуточного буфера.

Следующие свойства компоненты TQuery используются в динамическихSQL-запросах. Приведем некоторые из них:

DataSource – свойство типа TDataSource , определяет источник данных, значения полей которого используются как параметры для динамического запроса.

Param[Index] – свойство типа TParam, задает список элементов типа TParams, которые определяют параметры в динамическом запросе. С помощью этого свойства в редакторе значений параметров задаются начальные величины параметров. Чтобы перейти к редактору значений параметров запроса, нужно активизировать мышью значение этого свойства.

Класс TQuery добавляет к наследуемым от классов TDataSet, TBDDataSet, TDBDataSet следующие методы (некоторые из них):

ExecSQL- процедура выполняет SQL – запрос

Prepare – процедура посылает запрос в BDE для проверки синтаксиса и оптимизации. Рекомендуется выполнить для динамических запросов.

События, которые обрабатываются компонентой TQuery, полностью наследуются от класса TDataSet.

При постановке переменных связи в динамический запрос программным методом с помощью свойства Params выполняются обычно следующие шаги:

- необходимо убедиться в том, что таблица закрыта;

- подготавливается объект TQuery с помощью выдачи команды Prepare;

- свойству Params присваиваются конкретные значения;

- открывается запрос.

1. **Практические задания**

Порядок выполнения работ

1. Используя приложение, созданное ранее, спроектируйте и создайте запросы следующего вида:

- параметрические запросы по одному или нескольким параметрам с использованием группировки

- запрос с использованием агрегатных функций

- запрос на совпадение значений в нескольких таблицах

- вывод значений полей по id

- запрос на вывод значений за период

1. **Контрольные вопросы**

1) Какие операции можно использовать при формировании условия в предложении WHERE?

2) Поясните использование операций сравнения, BETWEEN, IN, LIKE.

3) Как изменить порядок выводимых строк в результирующем наборе?

4) Что в запросах вычисляют функции AVG и COUNT?

5) Для чего используется группировка в запросах?

**ЛАБОРАТОРНАЯ РАБОТА №12**

**ПРОЕКТИРОВАНИЕ МЕНЮ**

Цель работы:Получить практические навыки разработки меню. Формирование навыков и умений работы с компонентами TMainMenu, построения меню.

1. **Краткие теоретические сведения**

Элемент меню может представлять собой либо подменю, либо команду, либо разделительную линию. К какой категории относится конкретный элемент, зависит от его характеристик. Несмотря на это, любой элемент меню принадлежит к одному и тому же классу – TmenuItem и каждый может рассматриваться как самостоятельно меню.

Для того чтобы элемент меню представлял собой подменю, необходимо, чтобы свойство Items содержало соответствующее компоненты этого элемент меню является либо командой, либо разделительной линией.

Для того, чтобы элемент меню представлял собой разделительную линию, следует его свойству Caption задать значение «-».

Элементы меню могут объединяться в группы. Информация о принадлежности элементов к группам может использоваться при слиянии меню нескольких форм и при использовании взаимосвязанных элементов меню. При этом группы элементов меню должны идти в определенном порядке – по возрастанию их номеров.

Характеристики элемента меню задают его внешний вид и место в структуре меню. Среди его свойств отметим следующие:

Break – свойство, определяющее наличие разбиения меню на столбцы на текущем элементе;

Caption – свойство типа string, содержащее текст элемента меню.

Checked – свойство логического типа, определяющее, является ли элемент меню отмеченным

Default – свойство логического типа, если оно имеет значение True, текст элемента меню выделяется полужирным шрифтом, а двойное нажатие старшего элемента приводит к появления у текущего элемента события OnClick. По умолчанию имеет нулевое значение False.

Enabled – свойство логического типа; если оно имеет значение True, элемент меню реагирует на событие от мыши и клавиатуры.

GroupIndex – свойство целого типа, содержащее номер группы, к которой принадлежит элемент меню. По умолчанию меню нулевое значение.

Items[Index] – свойство-массив типа TMenuItem (индекс Index – целого типа), задающее младшие элементы меню по отношению к текущему элементу. Число элементов определяется свойством Count. Нумерация начинается с нуля и используется только для чтения.

MenuIndex – свойство целого типа, содержащее индекс текущей компоненты в списке items старшей компоненты

RadioItem – свойство логического типа, если оно имеет значение True, элемент меню выполняет роль переключателя, то есть помеченным может быть только один элемент одной группы.

ShortCut- свойство целого типа, определяющее код клавиши быстрого управления, при нажатии которых активизируется элемент управления меню.

Visible – свойство логического типа, если оно имеет значение True, элемент меню отображается на экране, если False – нет.

В классе TMenuItem определено одно событие – OnClick.

Класс имеет ряд методов, в основном связанных с корректировкой структур.

Add(Item) – процедура, добавляющая элемент Item в список младших элементов.

Delete(Index) – процедура, удаляющая из списка младших элементов элемент с индексом Index. При этом элемент из динамической памяти не удаляется и может быть повторно использован.

1. **Практические задания**

**Порядок выполнения работы**

1) Используя приложение, созданное ранее, спроектируйте, создайте главное меню, предоставляющее следующие возможности (состав и структуру меню и его подменю определить самостоятельно).

- завершение работы

- печать отчета, созданного в лабораторной работе

- поиск по таблицам

- вывод результатов запросов

- вывод отсортированных записей по параметру

1. **Контрольные вопросы**

1) Для чего и как используется свойство Filter компонента Table?

2) Какие используются методы для поиска записи по индексированному (не индексированному) полю?

3) Что такое закладка?

4) Как найти нужную запись? Перечислите разные способы.

**ЛАБОРАТОРНАЯ РАБОТА №7**

**Работа в ORACLE. Создание сервера приложений**

Цель работы:научиться создавать, редактировать и удалять таблицы и их элементы средствами ORACLEDatabaseExpressEdition. Создавать сервера приложений.

1. **Краткие теоретические сведения**

Пакет OracleDatabase 10gExpressEdition (OracleDatabaseXE) является свободно распространяемой версией СУБД Oracle. Работа с СУБД выполняется с помощью интуитивно понятного WEB-интерфейса браузера. С помощью этого интерфейса можно выполнять все основные операции по созданию таблиц баз данных, установлению связей между таблицами, вводу данных, созданию запросов, отчетов, администрированию пользователей.

Рассмотрим основные правила работы с этим пакетом.

1. Запускаем дистрибутив пакета - файл **OracleXE.exe** (размер 216 мб). Указываем место для установки и соглашаемся с остальными опциями. На одной из страниц установки потребуется указать пароль, с которым будем осуществляться доступ к серверу СУБД. Укажем в качестве пароля слово **oracle**.
2. После установки в меню **Пуск\Программы** появится раздел **OracleDatabase 10g ExpressEdition**, содержащий подразделы

Get Help - помощь

Backup Database – резервирование БД

Get Started – вызовсправкипо Oracle Express

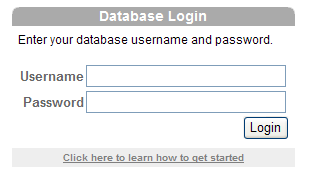
Go To Database Home Page – домашняя страница

Run SQL Command Line – работа с БД с помощью команд SQL

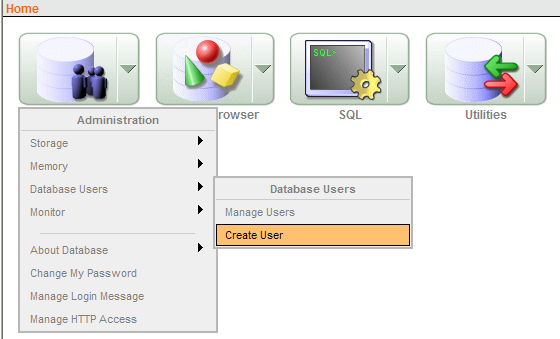
Start Database – запуск сервера

Stop Database – остановка сервера

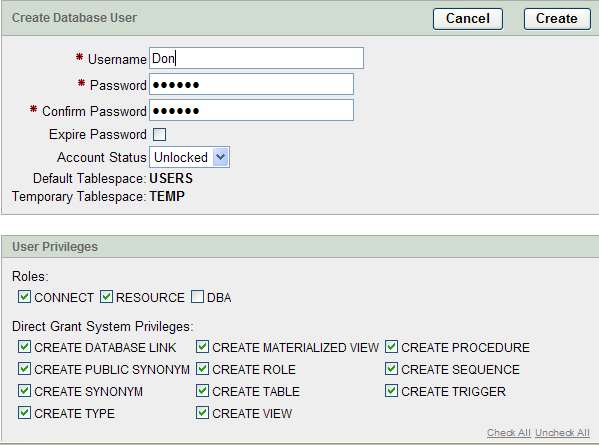
1. Идем к домашней странице **GoTo Database Home Page**. При этом открывается окно,



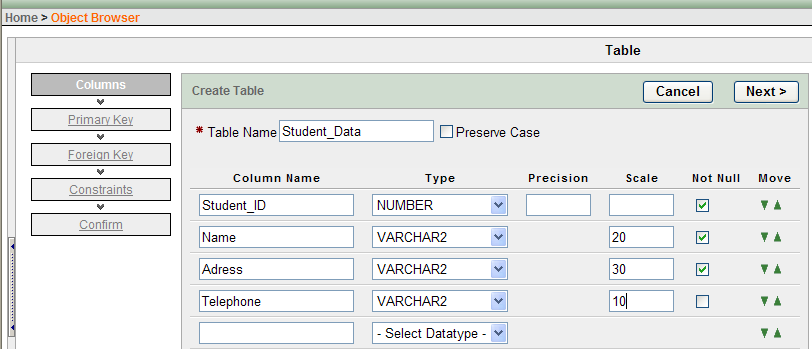
1. Первоначально на сервере создан один пользователь **system** с паролем, указанный при установке (**oracle**), поэтому первоначально надо зайти с этими логином и паролем. Для проверки Имени пользователя кликните по ссылке: «**Click here to learm how to get started**». В пункте 2 будет указан Username, который надо ввести.
2. Далеенажмемнакнопку**Administration**, выберемраздел**Database Users → Create Users:**



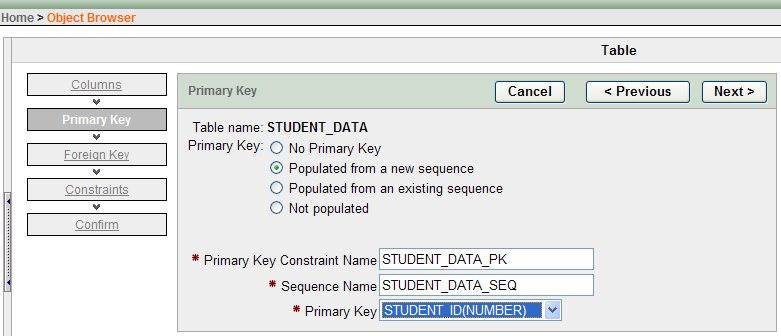
1. Создадим нового пользователя с вашим именем и паролем, например:



1. После заполнения формы нажимаем на кнопку **Create** (Создать). Далее заканчиваем сеанс, нажав ссылку **Logout** в правом верхнем углу экрана.
2. Выбираем:. Далее нажимаем **Create** →**Table.**
3. Открывается меню для создания таблицы. Заполним поля (**Columns**) таблицы:



1. Далее нажимаем **Next**. Открывается форма для создания Ключа (**Primary Key**): Выбираем **Populated from a new sequence**, задаем ключевое поле (в нашем случае **Student\_ID(Number**):



1. Нажимаем кнопку **Next.** Открывается форма для задания внешнего ключа **(Foreing Key).** Если внешний ключ не задается, нажимаем**Далее**.
2. Открывается форма для создания Ограничений (**Constraints)**. При отсутствии ограничений нажимаем **Finish.**
3. Следующая форма сообщает о том, что пользователем создана таблица. Нажимаем кнопку **Create.**
4. Открывается диалоговое окно построителя таблицы**: STUDENT\_DATA.** В левой части приводится список всех созданных таблиц. В правой части: мастер создания и изменения таблицы:

- Добавить столбец,

- Изменить столбец,

- Переименовать столбец,

- Удалить столбец,

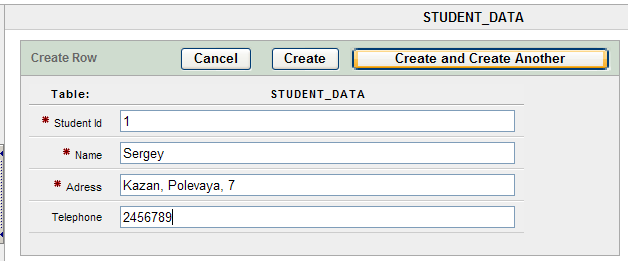
- Переименовать таблицу,

- Копировать таблицу,

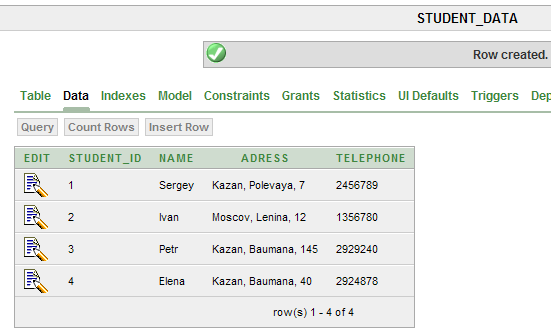
- Удалить таблицу.

Чтобы заполнить таблицу, выбираем вкладку **Data,** кнопку **Insert Row.**

В появившуюся форму заносим данные:



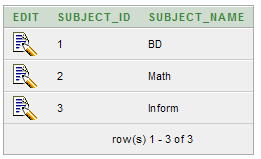
Далее нажимаем **Create**, затем кнопку **Insert Row.** Заполняем данные на следующего студента. В результате заполнения полей таблицы появляется список всех студентов:



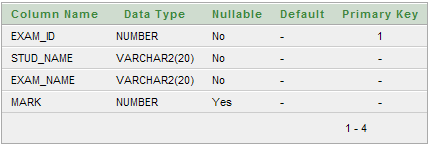
1. Создать таблицу **SUBJECT:**



Внести в таблицу следующие данные:



1. Создать таблицу **EXAMS:**



Занести в таблицу данные.

**2.Практические задания**

**Порядок выполнения работы**

1) создать пользователя

2) согласно теме индивидуального задания создать БД

3) заполнить таблицы тестовыми данными

**3. Контрольные вопросы**

1)Возможностиweb –интерфейсаOracleDatabase 10gExpressEdition

2)Перечислите некоторые ключевые возможности Oracle Database 10g

3) Какова архитектура сервера базы данных Oracle

Лабораторная работа № 13

Создание пользовательских приложений

Цель работы:Получение практических навыков проектирования и разработки приложений Oracle. Формирования навыков работы в среде OracleDatabase 10gExpressEdition

1. **Краткие теоретические сведения**

Пользовательские приложения в Oracle представляют собой отчеты и формы, выполненные как Web-страницы и позволяющие получать информацию из Базы данных.

Рассмотрим пример создания приложения формы отчета по таблицам, входящим в учебную базу данных пользователя с идентификатором HR.

*Разблокировка пробной учетной записи пользователя*

Чтобы создать свое приложение, Вы должны войти как пользователь базы данных. OracleDatabaseXE поставляется с экспериментальным пользователем базы данных, именуемым **HR**. Этот пользователь владеет несколькими таблицами базы данных в пробной схеме, которая может быть использована при создании приложений для вымышленного подразделения HumanResources. Однако, из соображений безопасности, учетная запись этого пользователя заблокирована. Вы должны разблокировать эту учетную запись, прежде чем сможете создать свое пробное приложение.

Чтобы разблокировать пробную учетную запись пользователя:

1. Убедитесь, что Вы все еще подключены как администратор базы данных, как это описано в предыдущем разделе.
2. Щелкните на иконке Administration, а затем щелкните DatabaseUsers.
3. Щелкните на иконку схемы HR, чтобы отобразить пользовательскую информацию для **HR.**



1. В ManageDatabaseUser введите следующие настройки:
   * Password и ConfirmPassword: Введите**hr** в качестве пароля.
   * Account Status: Выберите Unlocked.
   * Roles: Убедитесь, что активированы как CONNECT, так и RESOURCE.
2. Щелкните Alter User.

Теперь все готово для создания приложения.

*Подключение к пробной учетной записи*

Для подключения к пробной учетной записи:

1. Закончите работу с учетной записью администратора базы данных, нажав Logout в верхнем правом углу домашней страницы базы данных.
2. Нажмите Login.
3. В окне подключения введите **hr** в качестве имени пользователя и пароля.
4. Нажмите Login.

Появится домашняя страница базы данных.

*Создание простого приложения*

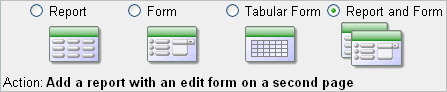
Создание приложения это самый простой способ просматривать и редактировать данные в Вашей базе данных. Вы создадите это приложение на основе таблицы **EMPLOYEES**, являющейся частью схемы **HR**.

Чтобы создать приложение, основанное на таблице **EMPLOYEES**:

1. На домашней странице базы данных щелкните иконку ApplicationBuilder.
2. Нажмите кнопку Create.
3. На открывшейся странице выберите CreateApplication и нажмите Next.
4. В поле CreateApplication введите следующие параметры:
   1. В поле Name введите **MyApp.**
   2. Остальные поля оставьте без изменений.
   3. Нажмите Next.

Далее, добавьте страницы к вашему приложению.

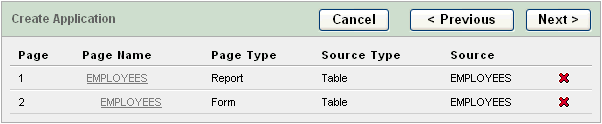
1. Под разделом Add Page:
   1. В опции Select Page Type выберите Report and Form.



Обратите внимание, что в поле Action отображается тип страницы, которую вы добавляете.

* 1. В поле Table or View выберите EMPLOYEES.
  2. Нажмите кнопку Add Page.

Две новые страницы отобразятся вверху страницы в разделе CreateApplication.



* 1. Click Next.

1. Опцию Tabs оставьте без изменений (One Level of Tabs) и нажмите Next.
2. Опцию SharedComponents оставьте без изменений и нажмите Next.

Эта опция позволит вам импортировать общие компоненты из других приложений. Общие компоненты - это стандартные элементы, которые могут быть отображены или применены на любой странице приложения.

1. Параметры полей Authentication Scheme, Language и User Language Preference Derived From оставьте без изменений и нажмите Next.
2. В опциях **User** Interface выберите Theme 2 и нажмите Next.

Темы это наборы шаблонов, которые можно использовать для задания расположения элементов и определения внешнего вида всего приложения.

1. Подтвердите сделанные изменения. Чтобы вернуться на предыдущую страницу мастера, нажмите Previous. Чтобы принять изменения, нажмите Create.

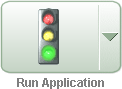
После того как вы нажмёте Create, вверху страницы появится следующее сообщение:

Application created successfully.

*Запускновогоприложения*

Чтобы запустить Ваше приложение:

1. Щелкните иконку Run Application.



1. На странице авторизации, введите **hr** как в поле UserName, так и в поле Password.

Появится Ваше приложение, показывая таблицу **EMPLOYEES**.

1. Исследуйте Ваше приложение. При желании, Вы можете делать запросы к таблице**EMPLOYEES**. Для управления приложением, используйте инструментальное меню разработчика в нижней части страницы.

Инструментальное меню разработчика

Инструментальное меню разработчика дает возможность оперативно отредактировать текущую страницу, создать новую страницу, элемент управления или компонент, посмотреть состояние сессии, а также включить/отключить режим отладки или ссылки редактирования.

1. Для выхода из приложения и возврата в ApplicationBuilder, щелкните EditPage 1 в инструментальном меню разработчика.
2. Для возврата на домашнюю страницу базы данных выберите пункт Home в верхней части страницы.

Описание gs_bread_myapp.gif следует

1. **Практические задания**

**Порядок выполнения работы**

- создать на основе индивидуального задания приложения в среде OracleDatabase 10gExpressEdition

## **3. Контрольные вопросы**

## 1)Каковы основные средства администрирования доступа

## 2)Как осуществляется подключение к СУБД

## 3)Как осуществляется ограничение доступа к отдельным частям таблицы

## 4) Понятие пользователи и схемы

## **Лабораторная работа** № 14

## Работа с базой данных с использованием утилиты SQLplus.

Цель работы: получение практических навыков использования утилиты SQLPLUS. Формирование навыков получения доступа к БД OracleExpress через локальные и глобальные сети.

1. **Краткие теоретические сведения**

Утилита SQLplus является клиентским приложением, которое может осуществить доступ к базе данных Oracle Express через локальную или глобальную сеть. Ее можно установить на компьютер клиента без установки сервера Oracle Express. При установке сервера она устанавливается автоматически.

Для работы с этой утилитой, добавим нового пользователя с именем **Stud**. Для этого выполним команды:

1. Запустите домашнюю страницу **Oracle Express**, выполняя команду **Пуск\Все программы\ OracleDatabase 10g ExpressEdition\GoTo Database Home Page.**
2. В появившемся приглашении введите логин system и пароль oracle. В первой вкладке администрирования выберите раздел DateBase Users\CreateUsers и добавьте нового пользователя с именем **test**, пароль – **test**.
3. Выполните подсоединение к базе данных пользователя **test** с помощью **SQLplus**, для этого:
4. Откройте меню **Пуск** системы **Windows** и нажмите **«Выполнить».** В появившемся окне введите команду **cmd** для запуска командного интерпретатора **DOS**.
5. Наберите команду **SQLplus**, клавиша <Enter>.
6. Введите имя пользователя **test,** затем пароль **test**. Должно появиться приглашение **SQL>**
7. Создайте таблицу:

* **Orders (Заказы)** с полями: **№ заказа, ФИО покупателя, Дата**.

Create table Orders (

ID Int not null primary key,

FIO varchar2 (20) not null,

Data date not null); клавиша<Enter>.

При работе в командной строке DOS предыдущие команды можно выбрать с помощью кнопок стрелочек вверх-вниз.

* Добавьте новый столбец **Название товара**: **(Name\_Tovar)**

Alter table Orders add (Name\_Tovar varchar2 (40) not null); клавиша<Enter>.

* Добавьтестроку данных.

Insert into Orders values (1, ‘Ivanov’,’25.10.2008’,’Bred’);

* Добавьте еще несколько строк данных.
  + Выполните просмотр данных из таблицы **Orders**

SELECT \* FROM Orders; клавиша<Enter>.

* Выполните просмотр количества записей:

SELECT count(\*) FROM Orders; клавиша<Enter>.

1. Выполните команду отсоединения:

Disconnect клавиша<Enter>.

Снова откройте домашнюю страницу, войдя под именем **test**, пароль – **test** и посмотрите созданную таблицу через раздел **ObjectBrower**.

1. **Практические задания**

**Порядок выполнения работы**

1)осуществите подсоединение к базе, созданной ранее

2) создайте средствамиSQLplus новую таблицу

3)заполните таблицу данными

4) выполните просмотр данных

5) отсоединетесь от БД

1. **Контрольные вопросы**
   * 1. Наиболее популярные команды SQL\*Plus
     2. Использование SQL\*Plus для форматированной выдачи
     3. Совместное использование команд SPOOL, SAVE и START

**ЛАБОРАТОРНАЯ РАБОТА №15**

**Управление транзакциями**

Цель работы: Освоение способов управления транзакциями в средах СУБД DB2 и Oracle. Формирование навыков и умений управления транзакциями,

1. **Краткие теоретические сведения**

Уровни изоляции

Потерянные изменения. Транзакция Т1 читает данные. Транзакция Т2 читает те же данные. Транзакция T1 на основании прочитанного значения вычисляет новое значение данных, записывает его в базу данных и завершается. Транзакция T2 на основании прочитанного значения вычисляет новое значение данных, записывает его в базу данных и завершается. В результате значение, записанное транзакцией Т2, "затрет" значение, записанное транзакцией Т1.

Грязное чтение. Транзакция Т1 изменяет некоторые данные, но еще не завершается. Транзакция Т2 читает эти же данные (с изменениями, внесенными транзакцией Т1) и принимает на их основе какие-то решения. Транзакция Т1 выполняет откат. В результате решение, принятое транзакцией Т2 основано на неверных данных.

Неповторяющееся чтение. Транзакция Т1 в ходе своего выполнения несколько раз читает одни и те же данные. Транзакция Т2 в интервалах между чтениями транзакцией Т1 изменяет эти данные и фиксируется. В результате оказывается, что чтения одних и тех же данных в транзакции Т1 дает разные результаты.

Фантом. Транзакция Т1 в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Транзакция Т2 в интервалах между выборками транзакции Т1 добавляет или удаляет строки или изменяет столбцы некоторых строк, используемых в критерии выборки, и фиксируется. В результате оказывается, что одни и те же выборки в транзакции Т1 выбирают разные множество строк.

Инструментальные средства DB2

Для выполнения сценариев проверки изолированности следует запустить два сеанса интерактивного SQL. Можно запустить в два сеанса DB2 Command Center, однако, Command Center - приложение довольно ресурсоемкое, и его выполнение в двух экземплярах на компьютере недостаточной мощности может оказаться затруднительным, поэтому мы предлагаем во втором сеансе запустить DB2 Command Line Processor. Command Line Processor - приложение для интерактивного выполнения операторов SQL из командной строки.

CONNECT RESET

Инструментальные средства Oracle

По умолчанию в среде SQL\*Plus режим AUTOCOMMIT выключен.

**Подготовка к работе**

Подготовить SQL-скрипты для выполнения проверок изолированности транзакций по образцу сценариев, описанных в справке. Ваши скрипты должны работать с таблицами

**3. Практические задания**

1)Запустить Oracle SQL\*Plus. Запустить второй экземпляр Oracle SQL\*Plus в отдельном окне. В обоих окнах соединиться с локальной базой данных.

2) Установить в обоих сеансах уровень изоляции READ COMMITTED. Выполнить сценарии проверки:

* потерянных изменений,
* грязного чтения,
* незавершенного чтения,
* фантома.

Записать протокол выполнения сценариев.

3)Установить в обоих сеансах уровень изоляции SERIALIZABLE. Выполнить сценарии проверки:

* незавершенного чтения,
* фантома.

Записать протокол выполнения сценариев.

4)Закончить работу с Oracle.

5)Запустить DB2 Command Center и соединиться с базой данных DB2EDU. В другом окне запустить второй экземпляр DB2 Command Center или DB2 Command Line Processor (см. справку).

6)Отключить в обоих сеансах режим AUTOCOMMIT.

7)Установить в обоих сеансах уровень изоляции UNCOMMITTED READ . Выполнить сценарии проверки:

* потерянных изменений,
* грязного чтения.

Записать протокол выполнения сценариев. Разорвать соединение с базой данных.

8) Установить в обоих сеансах уровень изоляции CURSOR STABILITY. Выполнить сценарии проверки:

* грязного чтения.
* незавершенного чтения.

Записать протокол выполнения сценариев. Разорвать соединение с базой данных.

9)Установить в обоих сеансах уровень изоляции REPEATEABLE READ. Выполнить сценария проверки фантома. Записать протокол выполнения сценария. Разорвать соединение с базой данных.

10)Закончить работу с DB2.

**3. Контрольные задания**

1) Понятие транзакции

2) Перечислите уровни изоляции транзакций

3) Перечислите возможные состояния транзакций

**Лабораторная работа 16**

**Функции PHP для работы с MySQL. Подключение к базе данных MySQL.**

Цель работы: получить практические навыки работы с PHP функциями подключения к серверу MySQL.

**1. Краткие теоретические сведения**

Для подключения к базе данных используется команда **mysql\_connect.** Синтаксискомнды:

resource**mysql\_connect** ([string server [, string username [, string password [, bool new\_link [, int client\_flags]]]]] )

Параметры:

Hostname – имя хоста, на котором находится база данных.

Username – имя пользователя.

Password – пароль пользователя.

Возвращает указатель на соединение с MySQL в случае успешного выполнения, или **FALSE** при неудаче.

**mysql\_connect()** устанавливает соединение с сервером MySQL. Следующие значения по умолчанию установлены для отсутствующих параметров: server = 'localhost:3306', username = имя пользователя владельца процесса сервера и password = пустой пароль.

Если второй вызов функции произошёл с теми же аргументами **mysql\_connect()**, новое соединение не будет установлено. Вместо этого функция вернёт ссылку на уже установленное соединение. Параметр new\_link может заставить функцию **mysql\_connect()** открыть ещё одно соединение, даже если соединение с аналогичными параметрами уже открыто.

Соединение с сервером будет закрыто при завершении исполнения скрипта, если до этого оно не будет закрыто с помощью функции [**mysql\_close()**](http://php.su/functions/?mysql-close).

Для работы с созданной БД необходимо выбрать ее командой **mysql\_select\_db**:

bool**mysql\_select\_db** ( string database\_name [, resource link\_identifier] )

Параметры:

database\_name – имябазыданных.

link\_identifier – ID соединения, которое получено в функции mysql\_connect. (параметр необязательный, если он не указывается, то используется ID от последнего вызова mysql\_connect). Возвращает true в случае успешного завершения или false в случае возникновения ошибки.

mysql**\_select\_db()** выбирает для работы указанную базу данных на сервере, на который ссылается переданный указатель. Если параметр указателя опущен, используется последнее открытое соединение. Если нет ни одного открытого соединения, функция попытается соединиться с сервером аналогично функции [**mysql\_connect()**](http://php.su/functions/?mysql-connect), вызванной без параметров.

Функция **mysql\_close** закрывает соединение с MySQL:

bool**mysql\_close** ( [resource link\_identifier] )

Возвращает **true** в случае успешного завершения или **false** в случае возникновения ошибки.

**mysql\_close()** закрывает соедиение с базой данных MySQL, на которое указывает переданный указатель. Если параметр link\_identifier не указан, закрывается последнее открытое (текущее) соединение.

Использование **mysql\_close()** не необходимо для непостоянных соединений. Они автоматически закрываются в конце скрипта.

Для отображения ошибок используется функция **mysql\_error()**.

string**mysql\_error** ( [resource link\_identifier] )

Возвращает строку, содержащую текст ошибки выполнения последней функции MySQL, или *''* (пустая строка) если операция выполнена успешно. Если в функцию не передан параметр ссылки на соединение, последнее открытое соединение будет использовано.

Пример:

<html>

<body>

<?php

$connect = mysql\_connect("localhost", "root");

if(!$connect) {

print ‘Не удалось подключиться к БД.<br/>’;

print mysql\_error();

exit;

}

$db = mysql\_select\_db("mydb",$db);

if(!$db) {

print ‘Не удалось выбрать БД.<br/>’;

print mysql\_error();

exit;

}

mysql\_close($connect);

?>

</body>

</html>

1. **Практическая часть**

**Порядок выполнения работы**

1) Создать простейший сценарий подключения к базе данных.

**3. Контрольные вопросы**

1) Для чего нужна функция mysql\_connect()?

2) Для чего нужна функция mysql\_select\_db()?

3) Что произойдет, если в функцию mysql\_close() не передать параметр?

**Лабораторная работа 17**

**ВЫПОЛНЕНИЕ ЗАПРОСОВ MYSQL. ВЫБОРКА НАБОРОВ ДАННЫХ.**

Цель работы:закрепление навыков выполнения запросов MySQL и выборки наборов данных.

**1. Краткие теоретические сведения**

**1.1. Выполнение запросов MySQL**

Все запросы к текущей базе данных отправляются функцией mysql\_query(). Этой функции нужно передать один параметр - текст запроса. Текст запроса может содержать пробельные символы и символы новой строки (\n). Текст должен быть составлен по правилам синтаксиса SQL. Пример запроса:

$q = mysql\_query("SELECT \* FROM mytable");

Приведенный запрос должен вернуть содержимое таблицы mytable. Результат запроса присваивается переменной $q. Результат - это набор данных, который после выполнения запроса нужно обработать определенным образом.

**1.2. Обработка результатов запроса**

Если запрос, выполненный с помощью функции **mysql\_query()** успешно выполнился, то в результате клиент получит набор записей, который может быть обработан следующими функциями PHP:

* mysql\_result() - получить необходимый элемент из набора записей;
* mysql\_fetch\_array() - занести запись в массив;
* mysql\_fetch\_row() - занести запись в массив;
* mysql\_fetch\_assoc() - занести запись в ассоциативный массив;
* mysql\_fetch\_object() - занести запись в объект.

Также можно определить количество содержащихся записей и полей в результате запроса. Функция mysql\_num\_rows() позволяет узнать, сколько записей содержит результат запроса:

$q = mysql\_query("SELECT \* FROM mytable");  
echo "Втаблице mytable ".mysql\_num\_rows($q)." записей";

Запись состоит из полей (колонок). С помощью функции mysql\_num\_fields() можно узнать, сколько полей содержит каждая запись результата:

$q = mysql\_query("SELECT \* FROM mytable");  
echo "Втаблице mytable ".mysql\_num\_fields($q)." полей ";

У нас также есть возможность узнать значение каждого поля. Это можно сделать с помощью следующей функции:

mysql\_result (int $result, int $row, mixed $field);

Параметр функции $row задает номер записи, а параметр $field - имя или порядковый номер поля.

Предположим, SQL-запрос вернул следующий набор данных:

Email Name Last\_Name

--------------------------------------------

ivanov@mail.ru Ivan Ivanov

petrov@mail.ruPetrPetrov

Вывести это в браузер можно следующим образом:

$rows = mysql\_num\_rows($q);

$fields = mysql\_num\_fields($q);

echo "<pre>";

for ($c=0; $c<$rows; $c++) {

for ($cc=0; $cc<$fields; $cc++) {

echo mysql\_result($q, $c, $cc)."\t";

echo "\n";

}

}

echo "</pre>";

Следует отметить, что функция mysql\_result() универсальна: зная количество записей и количество полей, можно "обойти" весь результат, но в тоже время, скорость работы данной функции достаточно низка. Поэтому, для обработки больших наборов записей рекомендуется использовать функции mysql\_fecth\_row(), mysql\_fecth\_array(), и.т.д.

Функция mysql\_fecth\_row(int $res) получает сразу всю строку, соответствующую текущей записи результата $res. Каждый следующий вызов функции перемещает указатель запроса на следующую позицию (как при работе с файлами) и получает следующую запись. Если более нет записей, то функция возвращает FALSE. Пример использования данной функции:

$q = mysql\_query("SELECT \* FROM mytable WHERE month=\"$db\_m\" AND day=\"$db\_d\");

for ($c=0; $c<mysql\_num\_rows($q); $c++) {

$f = mysql\_fetch\_row($q);

echo $f;

}

Использовать функцию mysql\_fecth\_row() не всегда удобно, так как значения всех полей одной записи находятся все в одной строке. Удобнее использовать функцию mysql\_fetch\_array(), которая возвращает ассоциативный массив, ключами которого будут имена полей.

Функция mysql\_fecth\_array(int $res [, int $result\_type]) возвращает не ассоциативный массив, а массив, заданный необязательным параметром $result\_type, который может принимать следующие значения:

* MYSQL\_ASSOC - возвращает ассоциативный массив;
* MYSQL\_NUM -возвращает массив с числовыми индексами, как в функции mysql\_fecth\_row();
* MYSQL\_BOTH - возвращает массив с двойными индексами, то есть вы можете работать с ним, как с ассоциативным массивом и как со списком (MYSQL\_BOTH - это значение по умолчанию для параметра $result\_type.

В PHP есть функция, возвращающая ассоциативный массив с одним индексом: mysql\_fetch\_assoc(int $res);

Фактически, данная функция является синонимом для mysql\_fetch\_array($res, MYSQL\_ASSOC);

Пример использования функции mysql\_fecth\_array():

$q = mysql\_query("SELECT \* FROM mytable WHERE month=\"$db\_m\" AND day=\"$db\_d\");

for ($c=0; $c<mysql\_num\_rows($q); $c++) {

$f = mysql\_fetch\_array($q);

echo "$f[email] $f[name] $f[month] $f[day] <br/>";

}

Как видно, использовать функцию mysql\_fetch\_array() намного удобнее, чем mysql\_fetch\_row().

**1.3. Функции получения информации о результатах SQL-запросов**

PHP предоставляет еще несколько полезных функций, которые позволяют узнать информацию о результах SQL-запросов.

* Функция mysql\_field\_name(int $result, int $offset) возвращает имя поля, находящегося в результате $result с номером $offset (нумерация начинается с 0). Другими словами, функция возвращает имя поля с номером $offset.
* Функция mysql\_field\_type(int $result, int $offset) возвращает тип поля с номером $offset в результате $result (номер задается относительно результата, а не таблицы);
* Функция mysql\_field\_flags(int $result, int $offset) возвращает пречисленные через пробел флаги (модификаторы), которые имеются у поля с номером $offset. Все поддерживаемые MySQL флаги:

|  |  |
| --- | --- |
| **Флаг** | **Описание** |
| not\_Null | Поле не может содержать неопределенного значения (NULL), то есть поле должно быть явно инициализировано |
| Primary\_Key | Поле будет первичным ключом - идентификатором записи, по которому можно однозначно идентифицировать запись; |
| auto\_increment | При вставке новой записи значение этого поля будет автоматически увеличено на единицу, потому в таблице никогда не будет двух записей с одинаковым значением этого поля; |
| Unique\_Key | Поле должно содержать уникальное значение; |
| Multiple\_Key | Индекс |
| Blob | Поле может содержать бинарный блок данных |
| Unsigned | Поле содержит беззнаковые числа |
| Zerofill | Вместо пробелов используются символы с кодом \0 |
| Binary | Поле содержит двоичные данные |
| enum | Поле может содержать один элемент из нескольких возможных (элемент перечисления) |
| timestamp | В поле автоматически заносится текущая дата и время при его модификации |

Функция mysql\_field\_flags() возвращает флаги в виде строки, в которой флаги разделяются пробелами.

**2. Практическая часть**

**Порядок выполнения работы**

1) Создать скрипт, который выбирает и выводит все данные из одной таблицы с помощью функции mysql\_fetch\_row().

2) Создать скрипт, который выбирает и выводит данные из нескольких таблиц с помощью функции mysql\_fetch\_array().

3) Создать скрипт, который получает информацию о полях одной таблицы.

**3. Контрольные вопросы**

1) С помощью какой функции отправляются запросы к выбранной базе данных?

2) Какие функции используются для обработки результатов запроса?

3) В чем отличие функций mysql\_fetch\_array() и mysql\_fetch\_assoc()?

4) Какие функции используются для подключения информации о результатах запросов?

**Лабораторная работа 18**

**Создание БД с помощью языка PHP. Типы данных MySQL. PearDB**

Цель работы: формирование навыков работы с функциями PHP для MySQL.

**1. Типы данных MySQL**

* 1. **Числовые типы**

Если для целочисленных типов определен атрибут auto\_increment, столбец должен иметь индекс PRIMARY KEY или unique. Вставка значения null в столбец auto\_\_increment приводит к получению значения, а единицу большего максимального текущего значения столбца. Определение атрибута UNSIGNED запрещает ввод для числовых типов отрицательных значений.

При определении атрибута zerofill значения числовых типов дополняются нулями для достижения установленной ширины отображения столбца.

**TINYINT[(М)]**

**Описание:** очень малое целое.

**Допустимыеатрибуты:** auto\_increment, unsigned, zerofill.

**Диапазон:** от -128 до 127 (от -27 до 27-1) или от 0 до 255 (от 0 до 28-1), если UNSIGNED.

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0, если NOT NULL.

**Объем:** 1 байт.

**SMALLINT[(М)]**

**Описание:** малое целое.

**Допустимыеатрибуты:** auto\_increment, unsigned, zerofill.

**Диапазон:** от -32768 до 32767 (от -215 до 215-1) или от 0 до 65535 (от 0 до 216-1), если UNSIGNED.

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0, если NOT NULL.

**Объем:** 2 байта.

**MEDIUMINT[(М)]**

**Описание:** целое среднего размера.

**Допустимыеатрибуты:** auto\_increment, unsigned, zerofill.

**Диапазон:** от -8388608 до 8388607 (от -223 до 231-1) или от 0 до 16777215 (от 0 до 224-1), если unsigned.

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0, если NOT NULL.

**Объем:** 3 байта.

**INT[(М)]**

**Описание:** целое нормального размера.

**Допустимыеатрибуты:**AUTO\_INCREMENT, UNSIGNED, ZEROFILL.

**Диапазон:** от -2147483648 до 2147483647 (от -2м до 231-1) или от 0 до 4294967295 (от 0 до 2"-1), если UNSIGNED.

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0, если NOT NULL.

**Объем:** 4 байта.

**Синоним:** integer [(М)].

**BIGINT [(M)]**

**Описание:** большое целое.

**Допустимые атрибуты:** auto\_increment, unsigned, zerofill.

**Диапазон:** от -9223372036854775808 до 9223372036854775807 (от -2м до 263-1) или от 0 до 18446744073709551615 (от 0 до 264-1), если UNSIGNED.

**Значение по умолчанию:**null, если столбец может иметь значение NULL, 0, если NOTNULL.

**Объем:** 8 байт.

**FLOAT [(M,D)]**

**Описание:** малое число с плавающей запятой; одинарная точность (менее точный тип, чем DOUBLE).

**Допустимые атрибуты:**zerofill.

**Диапазон:** минимальные ненулевые значения — ±1,175494351Е-38; максимальные ненулевые значения — ±3,402823466Е+38.

**Значение по умолчанию:**null, если столбец может иметь значение

NULL, 0, если NOTNULL.

**Объем:** 4 байта.

**Синоним:** в версиях, предшествующих MySQL 3.23, floatD) — синоним для FLOAT со значениями М и D по умолчанию.

**Замечание:** начиная с версии MySQL 3.23, FLOATD) — действительный тип числа с плавающей запятой (значения сохраняются с максимальной точностью, разрешенной аппаратными средствами и не округляются до определенного числа знаков после запятой).

**DOUBLE [(M,D)]**

**Описание:** большое число с плавающей запятой; двойная точность (более точный тип, чем float).

**Допустимые атрибуты:**ZEROFILL.

**Диапазон:** минимальные ненулевые значения — ±2,2250738585072014Е-308; максимальные ненулевые значения — ±1,7976931348623157Е+308.

**Значение по умолчанию:**null, если столбец может иметь значение NULL, 0, если NOTNULL.

**Объем:** 8 байт.

**1.1.1. Строковые типы**

Наиболее часто используются следующие:

**VARCHAR(M)**

**Описание:** строка переменной длины от 0 до М символов. Аргумент М должен быть целочисленным значением от 1 до 255 (в версиях до MySQL 3.23) и от 0 до 255 (в версиях MySQL 3.23 и выше). Замыкающие пробелы при считывании строки удаляются. Если длина строки превышает М символов, лишние символы при записи удаляются.

**Допустимые атрибуты:** binary.

**Допустимая длина:** от 0 до М байт.

**Значение по умолчанию:** null, если столбец может иметь значение NULL, " ", если NOT NULL.

**Объем:** длина значения плюс 1 байт для записи длины.

**Сравнение:** регистр символов не учитывается (учитывается, если BINARY).

**TEXT**

**Описание:** значение TEXT нормального размера.

**Допустимые атрибуты:** только глобальные атрибуты.

**Допустимая длина:** от 0 до 65535 (от 0 до 21б-1) байт.

**Значение по умолчанию:** null, если столбец может иметь значение NULL, " ", если NOT NULL.

**Объем:** длина значения плюс 2 байта для записи длины.

**Сравнение:** регистр символов не учитывается.

**ENUM("valuel","value2",...)**

**Описание:** перечень; значениям столбцов может присваиваться одно из значений списка.

**Допустимые атрибуты:** только глобальные атрибуты.

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, первое значение списка, если not null.

**Объем:** 1 байт для перечня с числом членов от 1 до 255, 2 байта для перечня с числом членов от 256 до 65535.

**Сравнение:** регистр символов не учитывается (учитывается в версиях до MySQL 3.22.1).

**1.1.2. Типы даты и времени**

**DATE**

**Описание:** дата, представленная в формате "YYYY-MM-DD".

**Допустимые атрибуты:** только глобальные атрибуты.

**Диапазон:** от 1000-01-01" до "9999-12-31".

**Нулевое значение:** 0000-00-00".

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0000-00-00", если NOT NULL.

**Объем:** 3 байта D байта в версиях, предшествующих MySQL 3.22).

**TIME**

**Описание:** значение времени, представленное в формате "чч.мм.сс" ("- чч:мм:сс" для отрицательных значений). Это значение отражает истекшее время, однако может устанавливаться и как суточное время.

**Допустимые атрибуты:** только глобальные атрибуты.

**Диапазон:** ОТ "-838 : 59 : 59" ДО "838-59-59".

**Нулевое значение:** 00:00:00".

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0000.00", если NOT NULL.

**Объем:** 3 байта.

**Замечание:** хотя значение 0:00:00" используется как нулевое при вставке недопустимых значений в столбец TIME, оно также является допустимым и лежащим в пределах нормального диапазона.

**DATETIME**

**Описание:** значение даты и времени (обязательны обе части), представленное в формате "YYYY-MMDD hh:mm:ss".

**Допустимые атрибуты:** только глобальные атрибуты.

**Диапазон:** от 1000-01-01 00:00:00" до "9999-12-31 23:59:59".

**Нулевое значение:** 0000-00-00 00:00:00".

**Значение по умолчанию:** NULL, если столбец может иметь значение NULL, 0000-00-00 00:00:00", если NOT NULL.

**Объем:** 8 байт.

**1.2. Функции MySQL**

Рассмотрим наиболее употребляемые и полезные функции PHP для MySQL.

int**mysql\_affected\_rows** ( [resource link\_identifier] )

Возвращает количество рядов, затронутых последним INSERT, UPDATE, DELETE запросом к серверу, на который ссылается указатель link\_identifier. Если ресурс не указан, функция использует последнее, успешное соединение, выполненное с помощью функции [mysql\_connect()](http://php.su/functions/?mysql-connect). При использовании транзакций **mysql\_affected\_rows()** надо вызывать после INSERT, UPDATE, DELETE запроса, но не после подтверждения.

При использовании UPDATE, MySQL не обновит колонки, уже содержащие новое значение. Вследствие этого, функция **mysql\_affected\_rows()** не всегда возвращает количество рядов, подошедших по условия, только количество рядов, обновлённых запросом.

Пример:

mysql\_query(‘UPDATE myTable SET school=123, city= ‘Минск’ WHERE id=4’);

if (mysql\_affected\_rows() > 0) {

print ‘Данныеуспешноизменены’;

}

int**mysql\_insert\_id** ( [resource link\_identifier] )

Возвращает ID, сгенерированный колонкой с AUTO\_INCREMENT последним запросом INSERT к серверу, на который ссылается переданный функции указатель *link\_identifier*. Если параметр *link\_identifier* не указан, используется последнее открытое соединение. Mysql\_insert\_id() возвращает 0, если последний запрос не работал с AUTO\_INCREMENT полями. Если вам надо сохранить значение, убедитесь, что mysql\_insert\_id() вызывается сразу после запроса.

Пример:

mysql\_query(“INSERT INTO mytable (product) values (‘kossu’)”);  
print “Last inserted record has id “ . mysql\_insert\_id();

string**mysql\_escape\_string** ( string unescaped\_string )

Функция экранирует все спец-символы в *unescaped\_string*, вследствие чего, её можно безопасно использовать в [mysql\_query()](http://php.su/functions/?mysql-query). Mysql\_escape\_string() не экраинрует “*%”* и “*\_”*. Функция идентична [mysql\_real\_escape\_string()](http://php.su/functions/?mysql-real-escape-string), исключая то, что [mysql\_real\_escape\_string()](http://php.su/functions/?mysql-real-escape-string) принимает параметром ещё и указатель на соединение и экранирует в зависимости от кодировки. Mysql\_escape\_string() не делает этого и результат работы не зависит от кодировки, в который вы работаете с БД.

Пример:

$item = “Zak’s Laptop”;  
$escaped\_item = mysql\_escape\_string($item);  
print “Escaped string: “ . $escaped\_item;

Результат:

Zak\’s Laptop

**1.2.1 Функциинепосредственно MySQL:**

**IFNULL(expr1,expr2)**

Еслиexpr1 неравноNULL, тофункцияIFNULL() возвращаетзначениеexpr1, впротивномслучае - expr2. В зависимости от контекста функция IFNULL() может возвращать либо числовое, либо строковое значение:

mysql> SELECT IFNULL(1,0);

-> 1

mysql> SELECT IFNULL(NULL,10);

-> 10

mysql> SELECT IFNULL(1/0,10);

-> 10

mysql> SELECT IFNULL(1/0,'yes');

-> 'yes'

Данная функция полезна при использовании LEFTJOIN. Допустим имеется таблица Students и таблица Hobbies, содержащая хобби студентов. Предполагается, что не каждый студент имеет хобби. Задача узнать хобби каждого студента, а если его нет – вывести строку ‘отсутствует’.

SELECT s.name, IFNULL(h.hobby, ‘отсутствует’) FROM students s

LEFT JOIN hobbies h ON s.idStudent = h.idStudent;

**IF(expr1,expr2,expr3)**

Если expr1 равно значению ИСТИНА (expr1 <> 0 и expr1 <> NULL), то функция IF() возвращает expr2, в противном случае - expr3. В зависимости от контекста функция IF() может возвращать либо числовое, либо строковое значение:

mysql> SELECT IF(1>2,2,3);

-> 3

mysql> SELECT IF(1<2,'yes','no');

-> 'yes'

mysql> SELECT IF(STRCMP('test','test1'),'no','yes');

-> 'no'

**LOCATE(substr, str)**

Возвращает позицию первого вхождения подстроки substr в строку str. Если подстрока substr в строке str отсутствует, возвращается 0:

mysql>SELECTLOCATE('bar', 'foobarbar');

-> 4

mysql> SELECT LOCATE('xbar', 'foobar');

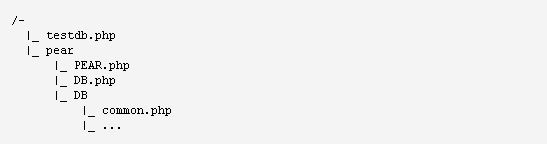
-> 0

**1.3. Написание сценариев PHP с использованием PEAR::DB**

**PEAR** ([PHP](http://ru.wikipedia.org/wiki/PHP)ExtensionandApplicationRepository) — это библиотека классов [PHP](http://ru.wikipedia.org/wiki/PHP) с открытым исходным кодом.

Установка:

1. Распакуйте архив PEAR.tgz в какую-нибудь директорию.
2. Для дальнейшей работы из понадобится только PEAR.php.
3. Создайте в структуре вашего сайта директорию pear/ и скопируйте PEAR.php в нее.
4. Распакуйте пакет DB.
5. Скопируйте файл DB.php и поддиректорию DB/ в директорию pear/ вашего сайта.
6. Откройте файл DB.php в редакторе, найдите строчку **require\_once 'PEAR.php';** и сотрите ее. Сохраните файл.
7. Откройте файл mysql.php в подкатегории DB категории PEAR, найдите строчку **require\_once 'DB/common.php';** и замените на **require\_once 'common.php';.** Сохраните файл.
8. В корневом каталоге создайте файл testdb.php. Структура должна выглядеть примерно так:



1. Добавьте в файл testdb.php следущий код:

<?php

header("Content-type: text/html; charset=utf-8");

require\_once 'PEAR/PEAR.php';

require\_once 'PEAR/DB.php';

$myDB =& DB::factory('mysql'); // подсоединяемся к mysql

$myDB->connect(DB::parseDSN('mysql://root:12345@localhost/testsmarty')); //подсоединяемся к базе

$myDB->setFetchMode(DB\_FETCHMODE\_ASSOC);

// устанавливаем кодировку

$myDB->query('set names utf8');

$q = 'SELECT \* FROMpages'; // какой-нибудьзапрос

$result = $myDB->query($q);

//проверяем корректность выполнения запроса

if(!DB::isError($result) && $result->numRows() > 0) {

// получаем ассоциативный массив значений

while($row = $result ->fetchRow()) {

print('name = '.$row['name']);

}

}

?>

**2. Практическая часть**

**Порядок выполнения работы**

1) Сделать скрипт создания БД, подключения к ней и создания таблицы.

2) Научиться применять типы и функции MySQL и PHP.

3) Настроить PearDB. Подключиться к своей БД и вывести данные различных запросов.

**3. Контрольные вопросы**

1) Приведите особенности установки библиотеки PearDB?

2) Перечислите функции PHP для работы с MySQL?

**Лабораторная работа 19**

**СОЗДАНИЕ WEB-СТРАНИЧКИ. ИНТЕРАКТИВНОЕ управление ДАННЫМИ.**

Цель работы***:*** формирование умений и навыков создания web-страниц с использованием MySQL.

**1. Краткие теоретические сведения**

**1.1. Работа с формами в php**

Для получения данных, вводимых пользователем, в html используется тег <form></form>.

Атрибут action тега form определяет, по какому адресу передаются данные с формы. Атрибут methodопределяет каким способом передаются данные: в глобальном массиве POST или GET. Обратиться к содержимому обоих массивов можно с помощью глобального массива REQUEST.

**1.2 Сохранение данных**

Сохранение полученных с формы данных производится с помощью выполнения команды mysql\_query. В качестве параметра передаем SQL запрос на вставку данных в таблицу, где в качестве значений подставляем данные из массива $\_POST ($\_GET), соответствующие именам элементов формы.

Пример:

Создаем таблицу с названием "test\_table" и с шестью полями с названиями:

"name" - здесь будут храниться имена пользователей;

"message" - здесь будут храниться сообщения;

"data" - здесь будет храниться дата отправки сообщения;

"id" - идентификационный номер записи (строки), ключевое поле.

Создаем файл index.html, содержащий форму для добавления данных:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Форма с сохранением в MySQL</title>

</head>

<body>

<h3>Пример формы с сохранением данных в MySQL</h3>

<form action="save\_form.php" method="post" name="test\_form">

<table border="1" cellpadding="0" cellspacing="0">

<tr>

<td colspan="2" align="center"><strong>Отправказапроса</strong></td>

</tr>

<tr>

<td width="150">Имя :</td>

<td><input type="text" name="test\_name" maxlength="30" /></td>

</tr>

<tr>

<td width="150">Сообщение :</td>

<td><textarea name="test\_mess"></textarea></td>

</tr>

<tr>

<td colspan="2" align="center">

<inputtype="submit" value="Отправить запрос" />

<input type="reset" value="Очистить" />

</td>

</tr>

</table>

</form>

</body>

</html>

Создаем обработчик формы save\_form.php с сохранением полученных данных в MySQL:

<?

/\* Соединяемся с базой данных \*/

$hostname = "localhost"; // название/путь сервера, с MySQL

$username = "root"; // имя пользователя

$password = ""; // пароль пользователя

$dbName = "test\_base"; // название базы данных

/\* Таблица MySQL, в которой будут храниться данные \*/

$table = "test\_table";

/\* Создаем соединение \*/

mysql\_connect($hostname, $username, $password) or die ("Не возможно создатьсоединение");

/\* Выбираем базу данных. Если произойдет ошибка - вывести ее \*/

mysql\_select\_db($dbName) or die (mysql\_error());

/\* Определяем текущую дату \*/

$cdate = date("Y-m-d");

/\* Составляем запрос для вставки информации в таблицу

name...date - название конкретных полей в базе;

в $\_POST["test\_name"]... $\_POST["test\_mess"] - в этих переменных содержатся данные, полученные из формы \*/

$query = "INSERT INTO $table(name, email, theme, message, data) values('".$\_POST['test\_name']."', '".$\_POST["test\_mess"]."', '$cdate')";

/\* Выполняем запрос. Если произойдет ошибка - вывести ее. \*/

mysql\_query($query) or die(mysql\_error());

/\* Закрываем соединение \*/

mysql\_close();

/\* В случае успешного сохранения выводим сообщение и ссылку возврата \*/

?>

<div style="text-align: center; margin-top: 10px;">

<font color="green">Данные успешно сохранены!</font>

<a href="index.html">Вернутьсяназад</a></div>

**2. Практическая часть**

**Порядок выполнения работы**

1) Создать следующие страницы:

- страница вывода всех данных одной таблицы;

- страница добавления записи;

- страница редактирования записей;

2) Создать скрипт для удаления записей из таблицы.

**3. Контрольные вопросы**

1) С помощью какой команды осуществляется сохранений данных, полученных с формы?

2) Какой используется тег для получения данных, вводимых пользователем?