Троицкий авиационный технический колледж – филиал федерального государственного бюджетного образовательного учреждения высшего образования «Московский государственный технический университет гражданской авиации»

Методическое пособие
для учащихся заочного отделения

ПМ 02 Разработка и администрирование баз данных
МДК 02.02 Технология разработки и защиты баз данных
Тема 2 РАЗРАБОТКА И ЗАЩИТА УДАЛЕННЫХ БАЗ ДАННЫХ
VI семестр (часть2)

г. Троицк, 2020 г.

Учебное пособие для студентов заочного отделения специальности 09.02.03. составлено в соответствии с требованиями Государственного образовательного стандарта к минимуму содержания и уровню подготовки выпускника по специальности 09.02.03 преподавателем Черевковой О.А.

Пособие состоит из 2 частей. В первой части рассматриваются основные понятия, технологии доступа к данным, основные принципы работы с удаленными базами данных, основы проектирования серверной части приложений, во второй – визуальные средства проектирования структуры базы данных, проектирование клиентской части приложения, администрирование и эксплуатация удаленных баз данных. Каждая часть имеет ряд практических упражнений для выработки навыков и закрепления теоретического материала.

.

Тема 3.4: Хранимые процедуры и триггеры

1. Понятие хранимой процедуры. Преимущества и недостатки ХП. Виды ХП
2. Команды создания, редактирования и удаления хранимой процедуры
3. Обработка ошибок ХП
4. Понятие и назначение триггера. Виды триггеров команды создания, редактирования и удаления триггера
5. Понятие и виды каскадных воздействий

1. Понятие и назначение хранимой процедуры. Виды ХП

Определение. Хранимая процедура – это 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. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления потоком.

Преимущества:

1. Снижают нагрузку на сеть (за счет выполнения запросов на стороне сервера БД)
2. Позволяют создавать библиотеки функций на сервере баз данных
3. Повышают безопасность данных (пользователи БД не могут получить непосредственный доступ к таблицам БД)
4. Упрощают выполнение запросов к БД (не нужно знать сложный язык запросов)

Недостатки:

1. Сложность написания
2. Большая нагрузка на сервер БД

Виды процедур: ХП и хранимые функции

2. Команды создания, редактирования и удаления хранимой процедуры

Создание ХП:

CREATE PROCEDURE имя_хранимой_процедуры ([параметр[, ...]])
[характеристика ...] тело_процедуры

Создание функции:

CREATE FUNCTION имя_функции ([параметр[, ...]])
[RETURNS тип]
[характеристика ...] тело_функции

параметр:
[IN | OUT | INOUT] имя_параметра тип
тип: любой допустимый тип данных MySQL

характеристика:
LANGUAGE SQL
| [NOT] DETERMINISTIC
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'строка'

Language SQL – язык запросов - SQL
Deterministic – ХП считается "детерминированной", если она всегда возвращает один и тот же результат для одних и тех же входных аргументов, в противном случае функция является "недетерминированной"
SQL security – параметры безопасности процедуры
Definer – процедуру будет использовать создатель
Invoker – процедуру будет использовать пользователь базы данных
Comment – описание ХП

тело процедуры: допустимый оператор (операторы) SQL процедуры.

Конструкция RETURNS может быть определена только для FUNCTION. Она используется для указания типа результата функции, при этом в теле функции должен присутствовать оператор RETURN значение.

Список аргументов, заключенный в круглые скобки, должен присутствовать всегда.
Если аргументы отсутствуют, следует использовать пустой список аргументов (). Каждый аргумент по умолчанию является аргументом IN.

Вызов ХП: CALL имя_хранимой_процедуры ([параметр[, ...]]);

Вызов функции: SELECT имя_функции([параметр[, ...]]);

Пример 1: **(**ХП возвращает количество записей в таблице emp и присваивает это значение системной переменной k**)**

```
create procedure kolich_emp (out kol int)
select count(*) from emp;
```

Вызов ХП: call kolich_emp(@k); /*системная переменная указывается, когда возвращается единственное значение для дальнейшего использования в других ХП*/

Пример 2: (ХП возвращает список сотрудников**)**

```
create procedure spisok_emp()
select * from emp;
```

Вызов ХП: call spisok_emp();

Пример 3: **(**ХП возвращает суммы заработных плат сотрудников, сгруппированных по отделам**)**

```
create procedure sum_salary()
select depno, SUM(salary)
from emp
group by depno;
```

Вызов ХП: call sum_salary();

Пример 4: (ХП возвращает строку: "This is my stored procedure"**)**

```
create procedure proc
NOT DETERMINISTIC
LANGUAGE SQL
SQL SECURITY DEFINER
COMMENT "
      BEGIN
            SELECT ‘This is my stored procedure’;
      END;
```

Пример 5. (ХП возвращает список сотрудников, заработные платы которых от 10000 до 80000 рублей**)**

```
create procedure spisokzp
(IN iInput1 INT, IN iInput2 INT)
select * from  emp
where salary between iInput1 AND iInput2;
```

Вызов ХП: call spisokzp(10000,80000);

Пример 6: (Функция присоединяет к строке Hello строку, указанную при вызове функции**)**

```
create function hello (s char(20))
returns  char(50)
return concat('Hello, ',s,'!') ;
```

Вызов функции: SELECT hello('world');

Переменные в MySQL

Простые переменные

Пример 7. (Процедура возвращает информацию о пятом отделе**)**

```
CREATE PROCEDURE PROC2()
BEGIN
DECLARE iVar INT DEFAULT 0; /*объявление и инициализация переменной iVar*/
SET iVar = 5; /*присвоено новое значение переменной iVar=5*/
SELECT * FROM depart
WHERE depno=iVar;
END;

CALL PROC3();
```

Системные переменные

```
SET @VS = 5;
SELECT @VS;
```

Пример 8.(ХП считает количество записей в таблице emp и присваивает это значение системной переменной VS**)**

```
CREATE PROCEDURE PROC3()
BEGIN
SELECT count(*) INTO @VS FROM EMP;
END;
```

Вернуть значение переменной: SELECT @VS;

Разница между простыми и системными переменными в том, что системные переменные доступны извне хранимой процедуры. То есть, чтобы извлечь какие-то данные нужно пользоваться системными, а переменные которые нужны только внутри процедуры должны быть простыми.

Редактирование ХП:

```
ALTER {PROCEDURE | FUNCTION} имя_хранимой_процедуры [характеристика ...];

характеристика:
NAME новое имя
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'строка'
```

Данный оператор можно использовать для переименования хранимой процедуры или функции, а также для изменения ее характеристик. В операторе ALTER PROCEDURE или ALTER FUNCTION разрешается указывать и более одной замены.

Удаление ХП:

```
DROP {PROCEDURE | FUNCTION} имя_хранимой_процедуры;
```

3. Обработка ошибок ХП

При некоторых условиях не исключена необходимость в обработке ошибок времени выполнения ХП.

Синтаксис обработки:

```
DECLARE тип_обработчика HANDLER FOR код_ошибки[, . . .]
оператор_хранимой_процедуры
```

тип__обработчика:

```
CONTINUE
| EXIT
```

код_ошибки:

```
SQLSTATE ‘пятисимвольный код ошибки’
| SQLWARNING – любое предупреждение MySQL
| NOT FOUND – ошибка, связанная с отсутствием объекта БД
| SQLEXCEPTION – ошибки, не охваченные ключевыми словами: SQLWARNING
                  и NOT FOUND
```

С помощью данного оператора задаются обработчики, каждый из которых может отвечать за обработку одной или более ошибок. Для обработчика CONTINUE выполнение текущей операции продолжается после выполнения оператора обработчика. Для обработчика EXIT выполнение текущего составного оператора BEGIN...END завершается.

Пример 9: (ХП обрабатывает ошибку «Нарушение уникальности значения первичного ключа». При этом системной переменной @error присваивается значение ‘Ошибка’ и ХП продолжает свою работу, то есть оператор SELECT Version(), который выводит версию MySQL, будет выполнен**)**

```
CREATE PROCEDURE handlerdemo ()
BEGIN
DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @error = ‘Ошибка’;
INSERT INTO depart VALUES (1);
INSERT INTO depart VALUES (1);
SELECT Version();
END;

Select @error;
```

4. Понятие и назначение триггера. Виды триггеров команды создания и удаления триггера

Определение. Триггер представляет собой хранимую процедуру, которая активизируется до или после наступления одного из трех событий: добавления, обновления или удаления данных.

Назначение триггеров:

1. Проверка условий на корректность добавляемых данных
2. Накапливание статистики в других таблицах
3. Вычисление значений полей
4. Каскадные воздействия (обновление и удаление)

Характеристики триггеров:

- Триггер ничего не принимает и ничего не возвращает
- Триггер выполняется автоматически
- В MySQL для одной таблицы можно создать не более 6 триггеров
- В теле триггера разрешается использовать команды языка ХП, объявление локальных переменных, обработчики ошибок

Синтаксис команды создания триггера:

```
CREATE TRIGGER <имя триггера>
{BEFORE|UPDATE}
{DELETE|INSERT|UPDATE}
ON <имя таблицы> FOR EACH ROW
BEGIN
<тело триггера>;
END;
```

old – обращение к «старому» значению поля
new - обращение к «новому» значению поля
before – триггер срабатывает до возникновения события
after - триггер срабатывает после возникновения события

Пример 1. Триггер не позволяет добавить новую запись, если добавляемое значение поля «Название отдела» уже присутствует в БД.

```
CREATE TRIGGER tr1 BEFORE INSERT ON depart
 FOR EACH ROW
 BEGIN

   SELECT COUNT(name1) into @k
   FROM depart
   WHERE NAME1=NEW.NAME1;

   IF (@k<>0) THEN
     SET NEW.name1=NULL;
  END IF;
END;
```

Пример 2. Триггер срабатывает после добавления новой записи в таблицу Depart и добавляет новую запись в таблицу LOG о том, что данная запись действительно добавлена в таблицу Depart данным пользователем, на данную дату.

```
CREATE TRIGGER INS_DEPART AFTER INSERT ON depart
  FOR EACH ROW
     BEGIN
       insert into LOG
       set operaz  = 'Insert',
       row_depno= new.depno,
       user1=user(),
       data_m = NOW();
     END;
```

Команда создания таблицы LOG:

```
CREATE TABLE log
 (
   ID  INTEGER(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
   Operaz VARCHAR(255) NOT NULL,
   Vremya TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
   Row_depno DECIMAL(2,0) NOT NULL,
   User1 VARCHAR(30),
   Data_m DATE,
 );
```

Каскадные воздействия.

Определение. Каскадными воздействиями (каскадным удалением и обновлением) называются автоматически выполняемые изменения (удаление или обновление) полей связи (внешних ключей) в подчиненных таблицах при изменениях значений полей связи (первичных ключей) в главных таблицах.

Внешние ключи для работы триггеров, выполняющих каскадные воздействия, необходимо удалять!

Пример 3. Триггер, выполняющий каскадное обновление в таблице emp.

```
CREATE TRIGGER upd BEFORE UPDATE ON depart
  FOR EACH ROW
    BEGIN
       UPDATE `emp` SET depno=NEW.depno
       WHERE depno=OLD.depno;
    END;
```

Пример 4. Триггер, выполняющий каскадное удаление из таблицы emp.

```
CREATE TRIGGER del BEFORE DELETE ON depart
  FOR EACH ROW
     BEGIN
       DELETE FROM emp
       WHERE emp.depno=old.depno;
     END;
```

Пример 5. Триггер автоматически присваивает значение полю koeff

```
CREATE TRIGGER koeff BEFORE INSERT ON emp
  FOR EACH ROW
     SET NEW.koeff=NEW.salary *0.13;
```

Удаление триггеров.

```
DROP TRIGGER имя_триггера;
```

Тема 3.5: Сортировка, фильтрация и поиск данных

Методы сортировки, поиска и фильтрации данных

Методы поиска данных

Для поиска определенной записи в наборе имеются два метода: Locate и Lookup. Метод Locate позволяет найти запись с определенными значениями заданных полей.

Синтаксис метода имеет вид:

```
function Locate(const  KeyFields: String; const KeyValues: Variant;
                Options: TLocateOptions):  Boolean;
```

KeyFields - перечень полей, участвующих в поиске. Поля в строке отделяются друг от друга точкой с запятой.

KeyValues - одно или несколько значений для поиска. Если значений несколько, необходимо передать функции вариантный массив или создать его с помощью функции VarArrayOf.

Options - набор параметров поиска. Может содержать значения:

loCaseInsensitive. Поиск без учета регистра (заглавные и малые символы).

loPartialKey. Значения полей для поиска даны не полностью. Например, если в KeyValues для поиска по полю last_name дана строка 'Ст', то будут найдены поля с фамилией 'Степанов' и 'Стеценко'.

В случае успешного поиска функция Locate делает найденную запись текущей и возвращает true.

Рассмотрим теперь вторую функцию поиска - LookUp. Эта функция похожа на только что рассмотренную функцию Locate:

```
function Lookup(const KeyFields: String; const  KeyValues: Variant;
                const ResultFields: String): Variant;
```

Главное ее отличие от Locate в том, что она не делает найденную запись текущей, а возвращает значения заданных полей найденной записи в виде вариантного массива. Как и в предыдущем случае, параметры:

KeyFields - перечень полей, участвующих в поиске. Поля в строке отделяются друг от друга точкой с запятой.

KeyValues - одно или несколько значений для поиска. Если значений несколько, необходимо передать функции вариантный массив или создать его с помощью функции VarArrayOf.

Кроме того, функции передается параметр ResultFields - перечень полей, значения которых должна возвратить функция. Как и KeyFields, он представляет собой список полей, отделенных друг от друга точкой с запятой.

Методы фильтрации данных

Существует два основных способа фильтрации наборов данных:

- использование свойства Filter;
- создание обработчика OnFilterRecord.

Для работы с фильтрами используются следующие свойства компонентов - потомков TDataSet:

Filter содержит условие, которому должны удовлетворять записи набора данных. Условие может содержать операторы сравнения и логические операторы. Например, last_name='T*' and start_date>'02.02.2000'

Filtered свойство типа boolean. Если true, режим фильтрации данных включен, и к набору данных применяется условие, содержащееся в свойстве Filter.

FilterOptions задает режим фильтрации. Может включать дополнительные условия:

JoCaselnsensitive - сравнение строк в условии Filter выполняется без учета регистра, строчные и прописные буквы считаются одним и тем же;

foNoParlialCompare — сравнение строк в условии Filter выполнятся по полному тексту, а '*' воспринимается как символ. Если это условие выключено, символ '*' в строках считается любым

набором символов, например, условие last_name='T*' выполняется для всех строк, начинающихся с символа T.

Методы сортировки

Предложение ORDER BY SQL- команды SELECT

Предложение ORDER BY применяется для сортировки результирующего набора данных по одной или нескольким колонкам. Для определения порядка сортировки используются ключевые слова ASC (по возрастанию) или DESC (по убыванию). По умолчанию данные сортируются по возрастанию. Синтаксис предложения ORDER BY имеет вид:

```
ORDER BY column1 [{ASC| DESC}]
[,column2 [{ASC| DESC}] [,...]
```

Например, для сортировки списка сотрудников по фамилии и затем по имени следует использовать следующий SQL-запрос:

```
SELECT LastName, FirstName, Title
FROM Employees
ORDER BY LastName, FirstName
```

Если требуется отсортировать данные в убывающем порядке (например, необходим список продуктов в порядке убывания цен), используется ключевое слово DESC:

```
SELECT ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC
```

Тема 3.6: Транзакции. Кэш. Компоненты работы с КЭШем

1. Понятие и назначение транзакции
2. Понятие и назначение КЭШа

1. Понятие и назначение транзакции

Транзакцией называется серия последовательных изменений данных, объединенных одним непременным условием: они либо все должны завершиться успешно, либо должны быть устранены последствия любого из них.

Транзакция – это группа действий, переводящая базу данных из одного целостного состояния в другое и выполняющаяся по принципу: «либо все, либо ни одного».

Клиентская программа управляет транзакциями с помощью трех методов компонента TDatabase: StartTransaction – стартует транзакцию, Commit – подтверждение изменений, Rollback – откат изменений.

2. Понятие и назначение КЭШа

Суть кэширования изменений заключается в том, что ***в каталоге запуска программы создается локальная копия данных и все последующие изменения относятся не к реальным данным таблиц БД, а к хранящейся в буфере (кэше) их локальной копии.***

После изменения данных в КЭШе они могут быть либо перенесены в реальные таблицы БД (подтверждение изменений), либо кэш ликвидируется без запоминания изменений(откат изменений).

Любой набор можно перевести в режим кэширования изменений, если в его свойство **CachedUpdates** поместить значение **TRUE**. **Подтверждение изменений** происходит путем

обращения к методу **ApplyUpdates** набора данных, **откат** – обращением к методу **CancelUpdates**.

Преимущества кэширования изменений:

1. снижение нагрузки на сеть
2. отсутствие блокировок данных в условиях многопользовательского доступа к ним
3. реализация диалоговых окон (формы, в которых пользователь может сначала изменить данные, но потом отказаться от сделанных изменений).

Клиентская программа одновременно может кэшировать сколько угодно наборов данных.

Тема 3.7: Исключительные ситуации и ошибки

1. Понятие исключительной ситуации
2. Обработка динамических ошибок

1. Понятие исключительной ситуации

Исключительные ситуации, или исключения, связаны с возникновением ошибок в процессе выполнения приложений. Рассмотрим способы обработки исключительных ситуаций.

Виды ошибок

Ошибки, возникающие в процессе разработки и выполнения программы, могут быть:
- синтаксические
- логические
- динамические

Синтаксические ошибки вызываются нарушением синтаксиса языка и выявляются и устраняются при компиляции программы

Логические ошибки являются следствием реализации неправильного алгоритма и проявляются при выполнении программы.

Динамические ошибки возникают при выполнении программы и являются следствием неправильной работы операторов, процедур, функций или методов программы, а также операционной системы. Динамические ошибки называют также ошибками времени выполнения (Runtime errors).

Программист должен предвидеть возможность возникновения динамических ошибок и предусматривать их обработку. Для обработки динамических ошибок введено понятие исключительной ситуации (исключения), которая представляет собой нарушение условий выполнения программы, вызывающее прерывание или полное прекращение ее работы.

Исключительные ситуации могут возникать по различным причинам, например, в случае нехватки памяти, из-за ошибки преобразования, в результате выполнения вычислений и т. д.

В любом случае независимо от источника ошибки приложение информируется (получает сообщение) о ее возникновении. Исключительная ситуация остается актуальной до тех пор, пока не будет обработана глобальным обработчиком или локальными процедурами.

2. Обработка динамических ошибок

Локальная обработка исключительных ситуаций

1. Конструкция try .. finally состоит из двух блоков (try и finally) и имеет следующую форму:

```
try
// Операторы, выполнение которых может вызвать ошибку
finally
```

```
// Операторы, которые должны быть выполнены даже в случае ошибки
end;
```

Она применяется для выполнения всех необходимых действий перед передачей управления на следующий уровень обработки ошибки или глобальному обработчику. Такими действиями могут являться, к примеру, освобождение оперативной памяти или закрытие файла. Эта конструкция не обрабатывает объект исключительной ситуации и не удаляет его, а выполняет действия, которые должны быть произведены даже в случае возникновения ошибки.

Работает конструкция try .. finally следующим образом: если в любом из операторов блока try возникает исключительная ситуация, то управление передается первому оператору блока finally. Если же исключительная ситуация не возникла, то последовательно выполняются все операторы обоих блоков.

Конструкция try .. except также состоит из двух блоков (try и except) и имеет следующую форму:

```
try
{Операторы, выполнение которых может вызвать ошибку}
except
{ Операторы, которые должны быть выполнены в случае ошибки}
end;
```

В отличие от предыдущей, данная конструкция применяется для перехвата исключительной ситуации и предоставляет возможность ее обработки.

Конструкция try .. except работает следующим образом: если в операторах блока try возникает исключительная ситуация, то управление передается первому оператору блока except. Если же исключительная ситуация не возникла, то операторы секции except не выполняются. В случае проявления ошибки операторы блока except могут ликвидировать ошибочную ситуацию и восстановить работоспособность программы. Для исключений, обрабатываемых в конструкции try .. except, глобальный обработчик не вызывается, а обработку ошибок должен обеспечить программист.

Блок except может быть разбит на несколько частей с помощью конструкций on .. do, позволяющих анализировать класс исключительной ситуации для более удобной и полной ее обработки. Конструкция on .. do применяется в случаях, когда действия по обработке исключительной ситуации зависят от класса исключения, и имеет следующую форму:

```
on {Идентификатор: класс исключения} do
{Оператор обработки исключения этого класса};
else {Оператор};
```

В операторе on класс возникшей исключительной ситуации сравнивается с указанным классом исключения. В случае совпадения классов выполняются операторы после слова do, реализующие обработку этой исключительной ситуации. Идентификатор (произвольное имя, заданное программистом) является необязательным элементом и может отсутствовать, при этом не ставится и разделительный знак ":". Идентификатор — это локальная переменная, представляющая собой экземпляр класса исключения, который можно использовать для доступа к объекту возникшего исключения. Эта переменная доступна только внутри 'своей" конструкции on .. do.

Если класс возникшей исключительной ситуации не совпадает с проверяемым классом, то выполняются операторы после слова else. Блок else является необязательным и может отсутствовать.

Раздел 4.
Администрирование и эксплуатация УБД

Тема 4.1: Привилегии доступа к данным

1. Создание и удаление учетной записи пользователя
2. Назначение привилегий
3. Отмена привилегий

1. Виды привилегий

СУБД MySQL является многопользовательской средой и для доступа к таблицам БД могут быть созданы различные учетные записи с разным уровнем привилегий.

Учетная запись пользователя состоит из имени пользователя и наименования хоста, с которого пользователю разрешено обращаться к серверу MySQL. Например, учетная запись 'root'@'127.0.0.1' (или, 'root'@'localhost' что одно и тоже) означает, что пользователь с именем root может обращаться к серверу с хоста, на котором расположен сервер, а 'wet'@'62.78.56.34' означает, что пользователь с именем wet может обращаться к серверу с хоста с IP-адресом 62.78.56.34 и ни с какого другого. Знаки '' – можно опускать, если имя пользователя или хоста не содержат специальных символов – и %. Эти символы используются для задания диапазона значений, например, 'wet'@'%' разрешает пользователю wet обращаться к северу MySQL с любых компьютеров сети, кроме хоста с именем localhost, такая учетная запись называется **сетевой учетной записью**. 'wet'@'%' , 'wet' и wet – (сетевые учетные записи) эквивалентны.

Создать учетную запись пользователя можно при помощи оператора CREATE USER.

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

Если пароль не указывается, то в его качестве выступает пустая строка.

Удаление учетной записи пользователя

DROP USER <имя пользователя>;

Изменение имени существующей учетной записи пользователя

RENAME USER <имя пользователя> TO <новое имя пользователя>;

Все учетные записи хранятся в таблице user системной базы данных с именем mysql.

Для просмотра содержимого полей host, user, password (имя хоста, имя учетной записи пользователя и пароль учетной записи) таблицы user системной БД нужно выполнить команду:

SELECT host, user, password FROM mysql.user;

Задание пароля пользователя

Для задания пароля используется ключевое слово IDENTIFIED BY, за которым в одиночных кавычках следует пароль.

Пример: CREATE USER Softtime1@localhost IDENTIFIED BY '1234567';

Пароль для учетной записи Softtime1 будет храниться в виде обычного текста. Для хранения пароля в виде хэш-кода, полученного в результате необратимого шифрования, нужно добавить между ключевым словом IDENTIFIED BY и паролем ключевое слово PASSWORD, которое означает, что пароль перед сохранением будет зашифрован с помощью функции PASSWORD().

После того, как новая учетная запись создана, в любом клиенте (или в командной строке) можно авторизоваться с именем нового пользователя:

mysql –u <имя пользователя>.

2. Назначение привилегий

Для назначения привилегий пользователю используется оператор GRANT.

GRANT <список привилегий> ON <база данных.таблица>
TO <имя пользователя.имя хоста> IDENTIFIED BY '<пароль>';

Запрос создает пользователя с именем us и паролем pass. Этот пользователь имеет права ALL для всех таблиц всех баз данных (*.*).

Если такой пользователь уже существует, то его привилегии будут изменены на ALL.

Список привилегий пользователей

№ п.п.	Привилегия	Операция, разрешенная привилегией
1.	ALL	Все привилегии, кроме GRANT OPTION
2.	ALTER	Редактирование таблицы с помощью оператора Alter table
3.	ALTER ROUTINE	Редактирование или удаление хранимой процедуры
4.	CREATE	Создание таблицы с помощью оператора Create table
5.	CREATE ROUTINE	Создание хранимой процедуры
6.	CREATE USER	Позволяет работать с учетными записями при помощи операторов Create user, Drop user, Rename user
7.	CREATE VIEW	Создание представления с помощью оператора Create view
8.	DELETE	Удаление записей с помощью оператора Delete
9.	DROP	Удаление таблиц с помощью оператора Drop table
10.	EXECUTE	Выполнение хранимых процедур
11.	INDEX	Создание и удаление индексов с помощью операторов Create index и Drop index
12.	INSERT	Добавление записей с помощью оператора Insert
13.	SELECT	Выборка данных с помощью оператора Select
14.	SHOW DATABASES	Позволяет просматривать список всех таблиц с помощью оператора SHOW DATABASES
15.	SHOW VIEW	Позволяет использовать оператор Show create view
16.	SHUTDOWN	Позволяет завершать работу сервера при помощи mysqladmin shutdown
17.	TRIGGER	Создание и удаление триггеров
18.	UPDATE	Обновление записей с помощью оператора Update
19.	USAGE	Синоним для статуса «отсутствуют привилегии»
20.	GRANT OPTION	Позволяет управлять привилегиями

		других пользователей

Один оператор GRANT может использоваться для предоставления привилегий сразу нескольким пользователям.

Выполнить: GRANT SELECT ON Example1.* TO ret@localhost, tor@localhost;

Привилегии могут быть заданы на одном из четырех уровней:

Ключевое слово ON	Уровень
ON *.*	Глобальный уровень. Пользователь может обращаться ко всем базам данных и всем таблицам, входящим в их состав.
ON *	Если текущая БД не была выбрана при помощи оператора USE, то это эквивалентно ON *.*, а если была выбрана, то привилегии будут относиться ко всем таблицам текущей БД.
ON db.*	Уровень базы данных. Привилегии распространяются на все таблицы базы данных db.
ON db.tb1	Уровень таблицы. Привилегии распространяются на таблицу tb1 базы данных db.
ON db.tb1	Уровень столбца. Привилегии распространяются на столбцы в таблице tb1 базы данных db. Список столбцов указывается в скобках через запятую после ключевых слов Select, Insert, Update.

Выполнить: GRANT SELECT ON Example1.* TO ret@localhost, tor@localhost;

Выполнить: GRANT SELECT ON Example1.emp TO ret@localhost, tor@localhost;

Выполнить: GRANT SELECT(tabno) ON Example1.* TO ret@localhost, tor@localhost;

Привилегии EXECUTION, RELOAD, SHOW DATABASES, SHUTDOWN могут быть установлены лишь на глобальном уровне.
Привилегии: SELECT, INSERT, UPDATE, DELETE, CREATE, GRANT OPTION, INDEX и ALTER - можно установить только для таблиц.

Доступные привилегии и контекст их применения можно уточнить при помощи оператора SHOW PRIVELEGES;

Предложение WITH GRANT OPTION

Запросы GRANT ALL ON Example1.* TO ‘us’@’localhost’ WITH GRANT OPTION и пара запросов GRANT ALL ON Example1 TO ‘us’@’localhost’;
GRANT GRANT OPTION ON Example1 TO ‘us’@’localhost’; - эквивалентны.

Ключевое слово ALL нельзя использовать совместно с другими привилегиями.
Привилегия GRANT OPTION распространяется не только на те привилегии, которыми обладает пользователь на настоящий момент, но и на привилегии, которые он может получить в будущем.

Выполнить: GRANT SELECT ON Example1.* TO ret@localhost, tor@localhost WITH GRANT OPTION;

Просмотр существующих привилегий текущего пользователя
SHOW GRANTS;

3. Отмена привилегий

Для удаления привилегий пользователя используется оператор REVOKE.
REVOKE <список привилегий> ON <база данных.таблица>
FROM <имя пользователя.имя хоста>