

MySQL

АДМИНИСТРИРОВАНИЕ

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
АДМИНИСТРИРОВАНИЕ ПОЛЬЗОВАТЕЛЕЙ	3
ВВЕДЕНИЕ	3
ПРОСМОТР СУЩЕСТВУЮЩИХ ПОЛЬЗОВАТЕЛЕЙ И ИХ ПРАВ	4
СОЗДАНИЕ, РЕДАКТИРОВАНИЕ и УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЕЙ	6
АДМИНИСТРИРОВАНИЕ ПРАВ ПОЛЬЗОВАТЕЛЕЙ	9
ИСПОЛЬЗОВАНИЕ РОЛЕЙ	13
РЕЗЕРВНЫЕ КОПИИ	17
ВВЕДЕНИЕ	17
ИСПОЛЬЗОВАНИЕ УТИЛИТЫ MYSQLDUMP	18
ИСПОЛЬЗОВАНИЕ КОМАНДЫ SELECT INTO OUTFILE/DUMPFIL	22
ВОССТАНОВЛЕНИЕ ДАННЫХ ИЗ РЕЗЕРВНЫХ КОПИЙ	24
ЖУРНАЛИРОВАНИЕ (ЛОГИРОВАНИЕ)	27

АДМИНИСТРИРОВАНИЕ ПОЛЬЗОВАТЕЛЕЙ

ВВЕДЕНИЕ

- Начиная с самых первых занятий и заканчивая сегодняшним днем, мы активно вставляли, получали и обновляли данные в наших таблицах, создавали функции и процедуры, запускали триггеры и даже меняли структуру самих таблиц. Однако еще раньше – при инсталляции самой СУБД MySQL – мы создали пользователя с именем *root* и паролем *secret*. От имени этого суперпользователя, обладающего всеми возможными правами и привилегиями, мы и совершали все наши действия. Однако то, что хорошо при изучении технологии, не всегда безопасно и правильно при ее использовании в настоящих программах. Представим ситуацию, когда наша программа (сайт, мобильное приложение и т.д.) была взломана, и злоумышленники получили доступ к базе данных от имени пользователя *root* – это значит, что могут быть украдена или даже уничтожена не только база данных нашей программы, а все другие базы данных и пользователи. Таким образом, после взлома одного сайта могут потенциально быть уничтожены сотни других сайтов, чего крайне не хотелось бы. Поэтому в современных системах принято соблюдать принцип наименьших привилегий – т.е. во время работы иметь только те права, которые нам реально нужны, а все остальные права должны быть условно выключены.
- MySQL позволяет соблюдать принцип, указанный в предыдущем пункте – т.е. создавать разных пользователей, которые имеют право делать только строго определенные действия. Например, для работы сайтов обычно используют пользователей, которые могут делать только следующие действия – **INSERT ...**, **SELECT ...**, **UPDATE ...** и **DELETE ...** – и это все разрешено только для одной базы данных. Абсолютно все права на всю СУБД необходимо предоставлять только администраторам того дата-центра, в котором находится сайт. Хорошим тоном было бы ограничить те IP адреса, с которого такие суперпользователи могут попытаться воспользоваться СУБД. Администраторы могут помочь осуществить первичную установку сайта, т.к. только им будет доступно создание таблиц и связей между ними. Кроме создания пользователей и ограничения их прав, в MySQL можно создавать временные пароли, объединять пользователей в группы, называемые ролями, блокировать пользователей на определенное время, а также много чего другого, что мы подробно разберем в этой главе.

ПРОСМОТР СУЩЕСТВУЮЩИХ ПОЛЬЗОВАТЕЛЕЙ И ИХ ПРАВ

- Перед тем, как мы начнем непосредственно работать с правами и пользователями, было бы правильно начать с того, как просмотреть наше имя и права на данный момент, а также имена и права других пользователей во всей СУБД. После того, как мы заходим внутрь MySQL, мы можем выполнить функцию **USER()** – данная функция возвращает имя того пользователя, права которого мы в данный конкретный момент используем:

```
SELECT USER();
```

Если мы зашли под тем пользователем, который был создан при инсталляции СУБД MySQL, то в качестве результата мы получим следующую запись:

USER()
root@localhost

- В предыдущем пункте можно подметить одну странность – имя нашего пользователя действительно *root*, однако мы получили некий *root@localhost*. На самом деле имя пользователя в MySQL применяется как совокупность двух значений, разделенных знаком @. Первое значение – это то имя, которые мы выбрали при создании пользователя и которое мы используем как логин при входе в СУБД. Второе значение гораздо интереснее – это тот адрес, с которого нам позволено заходить в СУБД. Адреса могут быть трех видов:

- % (например, *john@%*) – это не совсем адрес, а скорее особый символ, который позволяет вход с абсолютно любых IP адресов.
- IP адрес (например, *samantha@192.168.0.2* или *abacaba@::1*) – конкретный IP адрес в формате IPv4 или IPv6 (если данный формат IP поддерживается нашей системой).
- Доменное имя (например, *root@localhost* или *username@example.com*) – мы подставляем некое имя, которое DNS получит из того адреса, с которого мы осуществляли подключение. Если мы подключаемся прямо с того компьютера, на котором работает СУБД, то адрес 127.0.0.1

преобразуется в localhost. Важный момент – преобразование с помощью DNS можно отключить, если в конфигурационном файле **my.ini** в разделе **[mysqld]** вставить настройку **skip-name-resolve=1** и перезапустить MySQL сервер. В данном случае будут обрабатываться будет только символ % и конкретные IP адреса.

- Если мы хотим увидеть всех пользователей, которые существуют на нашем MySQL сервере, то нам необходимо сделать запрос в таблицу user, которая находится внутри встроенной базы данных mysql. В таблице много столбцов, поэтому мы выберем только самые интересные – пароль (authentication_string – его мы выведем лишь частично, т.к. они займут слишком много места, если показать их целиком), имя и домен:

```
SELECT authentication_string AS password, host, user FROM mysql.user;
```

Можно заметить, что кроме нашего пользователя существуют также mysql.infoschema, mysql.session и mysql.sys. Эти пользователи используются самой СУБД и мы не должны как-нибудь взаимодействовать с ними.

password	host	user
\$A\$005\$...	localhost	mysql.infoschema
\$A\$005\$...	localhost	mysql.session
\$A\$005\$...	localhost	mysql.sys
.sm4tuD ...	localhost	root

- Как уже было сказано, мы показали сохраненные пароли лишь частично, но даже из того, что было показано, можно понять, что пароль у root никак не может равняться secret. Дело в том, что MySQL хранит пароли в преобразованном виде, и преобразование это осуществляется с помощью необратимой хэш-функции. Когда мы пытаемся зайти в систему, СУБД берет наш введенный пароль, пропускает его через такую же хэш-функцию и потом сравнивает с сохраненным паролем. Это сделано для безопасности, т.к. люди часто используют одинаковые пароли на многих сайтах – если наш сайт будет взломана, злоумышленник не получит пароли в явном виде и не сможет начать подставлять их в других местах.

- Чтобы просмотреть права конкретного пользователя в полном и относительно читаемом виде, мы можем использовать команду **SHOW GRANTS FOR ...** – применим ее для получения данных о нас самих – *root@localhost*. Результат этой команды слишком большой, поэтому мы не будем здесь его отображать.

```
SHOW GRANTS FOR 'root'@'localhost';
```

- В следующем примере мы можем посмотреть какие пользователи в данный момент авторизированны в нашей СУБД, и что именно они делают – для этого используется таблица *processlist* во встроенной базе данных *information_schema*:

```
SELECT user, host, db, command FROM
information_schema.processlist;
```

Кроме *root*, который в данный момент производил запрос на получение этой таблицы, мы также видим планировщик *event_scheduler*, который отвечает за запуск событий:

user	host	db	command
root	localhost:58510	information_schema	Query
event_scheduler	localhost	NULL	Daemon

СОЗДАНИЕ, РЕДАКТИРОВАНИЕ и УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЕЙ

- В MySQL добавление пользователя осуществляется при помощи команды **CREATE USER**, после которой надо указать имя и адрес нового пользователя, разделенные знаком *@*. Каждое из значений можно обернуть в кавычки, но это не всегда является обязательным – кавычки нужны в тех случаях, если в имени будет ключевое MySQL слово (например, **TABLE**) или пробел. Маленькое дополнение – символ *@* и адрес после него также не являются обязательными – если их не подставить, то в качестве адреса автоматически запишется символ *%* – то есть будет разрешен любой адрес.

```
CREATE USER 'new_user'@'localhost';
```

```
CREATE USER 'another_new_user';
```

- В прошлом пункте мы создали двух пользователей, но на данный момент они бесполезны – они даже не смогут зайти в СУБД, т.к. у них нет пароля. Для установки пароля после создания применяется команда **ALTER USER**, после которой нам следует указать полное название нашего пользователя, а уже затем команду **IDENTIFIED BY** и в кавычках наш новый пароль.

```
ALTER USER 'new_user'@'localhost'
IDENTIFIED BY 'opensesame123';
```

- Теперь существование одного из наших пользователей обрело определенный смысл, т.к. он даже может попасть на сервер. Чтобы это сделать на системе Windows, следует в командной строке перейти в директорию "C:\Program Files\MySQL\MySQL Server 8.0\bin" и выполнить команду **.\mysql.exe -unew_user -popensesame123**. В результате мы окажемся внутри нашей СУБД.
- В MySQL мы можем не только устанавливать пароль, но и управлять им – всего есть 6 дополнительных настроек:
 - **PASSWORD EXPIRE NEVER** – срок действия пароля неограничен (применяется по умолчанию)
 - **PASSWORD EXPIRE** – требует смены пароля при первом заходе
 - **PASSWORD EXPIRE INTERVAL N DAY** – срок действия пароля истекает через N дней
 - **PASSWORD HISTORY N** – запрещает повторное использование любого из N последних паролей, при его замене на новый
 - **PASSWORD REUSE INTERVAL N DAY** – новый пароль не должен быть равен паролю, который использовался за последние N дней
 - **PASSWORD REQUIRE CURRENT** – требует, чтобы при изменении пароля указывался текущий пароль

Применим наши знания, чтобы у пользователя `another_new_user` система попросила сменить пароль при первом заходе:

```
ALTER USER 'another_new_user'
IDENTIFIED BY '12345678' PASSWORD EXPIRE;
```

При следующем нашем входе в СУБД от имени этого пользователя, при попытке выполнить любую команду мы будем получать следующую ошибку (пока не сменим пароль на новый – команда смены нашего пароля будет нам разрешена):

ERROR 1820 (HY000): You must reset your password using ALTER USER statement before executing this statement.

- Мы можем управлять неудачными попытками входа на наш сервер. В этом случае предоставляется возможность указать точное количество неудачных попыток и то количество дней, на которое пользователь будет заблокирован после этих попыток. Например, мы хотим разрешить только 3 попытки пользователю 'new_user'@'localhost', а потом заблокировать его на 1 день:

```
ALTER USER 'new_user'@'localhost'
FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 1;
```

Также за неудачные попытки можно блокировать навсегда:

```
ALTER USER 'new_user'@'localhost'
FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME UNBOUNDED;
```

- Иногда нужно явно заблокировать пользователя (если он совершил некий проступок или вообще без всяких условий) – для этого используется опция **ACCOUNT LOCK**:

```
ALTER USER 'new_user'@'localhost' ACCOUNT LOCK;
```

Если нам надо разблокировать пользователя, то применяется конструкция **ACCOUNT UNLOCK**:

```
ALTER USER 'new_user'@'localhost' ACCOUNT UNLOCK;
```

- Для удаление пользователя мы должны использовать команду **DROP USER**, после которой должно идти имя пользователя. Также команду можно дополнить конструкцией **IF EXISTS**:

```
DROP USER 'another_new_user';
DROP USER IF EXISTS 'another_new_user';
```


АДМИНИСТРИРОВАНИЕ ПРАВ ПОЛЬЗОВАТЕЛЕЙ

- Мы создали пользователей и даже позволили им заходить в систему, однако у них отсутствуют какие-либо права – они не могут ни создавать данные, ни получать данные, ни их модифицировать. Количество всех возможных прав в MySQL поистине огромно, поэтому ниже приведем только самые часто используемые и полезные из них:

CREATE — создание новых баз данных и таблиц.

DROP — удаление баз данных и таблиц.

INSERT — вставка строк в таблицу.

UPDATE — обновление значений столбцов таблиц.

DELETE — позволяет удалять строки из таблицы.

ALTER — позволяет изменять структуру таблиц. Требуется **CREATE** и **INSERT** привилегии.

SELECT — позволяет производить выборку из таблиц.

LOCK TABLES — позволяет блокировать таблицы.

REFERENCES — позволяет создавать связь между таблицами по внешнему ключу.

EVENT — дает право на создани, изменение и удаление событий.

TRIGGER — позволяет создавать, изменять и удалять триггеры.

INDEX — позволяет добавлять и удалять индексы таблиц.

CREATE TEMPORARY TABLES — позволяет создавать временные таблицы.

CREATE VIEW — позволяет создавать представления.

CREATE ROUTINE — позволяет создавать функции и процедуры.

ALTER ROUTINE — позволяет изменять функции и процедуры.

EXECUTE — позволяет выполнять функции и процедуры.

CREATE USER — позволяет создавать, изменять, переименовывать и удалять пользователей.

PROXY — позволяет одному пользователю войти в MySQL под видом другого.

RELOAD — разрешает использование оператора FLUSH, который чистит кэш MySQL.

пользователей. Возможно дать и отобрать только те права, которыми назначающий сам располагает.

SHOW DATABASES — позволяет выполнять команду SHOW DATABASES.

SHUTDOWN — привилегия позволяет выполнить оператор SHUTDOWN, выключающий MySQL сервер.

GRANT OPTION — позволяет назначать и отбирать права

- Обычно права выдаются не просто так, а на некую конкретную базу данных и таблицы в ней – разделителем между ними служит точка. Однако при необходимости можно выдать права на все таблицы и на все базы данных. Приведем ниже все возможные варианты объектов, на которые выдаются права (сразу оговоримся, что все указанные базы данных и таблицы должны реально существовать):
 - `*.*` права будут выданы на все базы данных и все таблицы
 - `site.*` права выдаются на все таблицы базы данных site
 - `site.users` права выданы на таблицу users для базы site
- Мы, наконец, можем давать права нашим пользователям, но перед этим еще создадим новую базу данных и таблицу, которые будут использоваться для назначения прав:

```
CREATE DATABASE site;

USE site;

CREATE TABLE IF NOT EXISTS users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

Вот теперь предоставим права на вставку, выборку, обновление, удаление и выполнение функций для базы данных

site и таблицы *users*. Предоставление прав осуществляется при помощи конструкции **GRANT ... ON ... TO ...**:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON site.users
TO 'new_user'@'localhost';
```

Обратим внимание, что право на выполнение функций не может быть выдано на конкретную таблицу – его надо выдавать в контексте всех таблиц базы данных:

```
GRANT EXECUTE
ON site.*
TO 'new_user'@'localhost';
```

- Тот факт, что пользователь имеет права, еще не означает, что он сам сможет эти права назначать другим пользователям. Для этого используется право **GRANT OPTION** – когда мы объявляем его вместе с другими правами, то это означает, что оно будет распространяться только на эти права:

```
GRANT GRANT OPTION, SELECT, INSERT, UPDATE, DELETE
ON site.users
TO 'new_user'@'localhost';

GRANT GRANT OPTION, EXECUTE
ON site.*
TO 'new_user'@'localhost';
```

Интересно заметить, что право предоставления прав можно задать другим способом – прямо в момент назначения других прав в конце выражения можно поставить конструкцию **WITH GRANT OPTION** (это гораздо понятнее, чем тот случай, когда **GRANT OPTION** идет в общем перечислении с другими правами):

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON site.users
TO 'new_user'@'localhost' WITH GRANT OPTION;

GRANT EXECUTE
ON site.*
TO 'new_user'@'localhost' WITH GRANT OPTION;
```

- Иногда нам надо предоставить все доступные нам привилегии на все возможные объекты в СУБД – для этого можно использовать специальное право **ALL PRIVILEGES** (которое, тем не менее, не дает права наделять правами других, поэтому в этом случае конструкция **WITH GRANT OPTION** также необходима):

```
GRANT ALL PRIVILEGES
ON *.*
TO 'new_user'@'localhost' WITH GRANT OPTION;
```

- Очень часто в справочной литературе и статьях в Интернете после обновления прав советуется сбросить кэш прав (при помощи команды **FLUSH PRIVILEGES**), чтобы все изменения вступили в силу немедленно:

```
-- обновляем права
GRANT ALL PRIVILEGES
ON *.*
TO 'new_user'@'localhost' WITH GRANT OPTION;

-- сбрасываем кэш прав
FLUSH PRIVILEGES;
```

Нельзя сказать, что эти действия неправильные – лучше назвать их чрезмерными. Дело в том, что MySQL сервер в нормальном режиме (т.е. если он не был запущен с параметром **--skip-grant-tables**) держит права в памяти, а не читает их каждый раз из таблицы *mysql.user*. Однако в документации MySQL написано, что при использовании команды GRANT кэш будет чиститься сам.¹ Однако если мы сами поменяем что-либо в таблице *mysql.user* (чего делать крайне не рекомендуется), то тогда команда **FLUSH PRIVILEGES** будет действительно необходима, чтобы изменения применились должным образом.

- Теперь пришла пора поговорить об удалении прав. Это действие осуществляется при помощи команды **REVOKE ... ON ... FROM**. Например, отнимем возможность удалять и обновлять данные:

```
REVOKE DELETE, UPDATE, GRANT OPTION
ON *.*
FROM 'new_user'@'localhost';
```

¹ <https://dev.mysql.com/doc/refman/8.0/en/privilege-changes.html>

В этом запросе право **GRANT OPTION** удален не для всех прав, а только для **DELETE** и **UPDATE**. На самом деле, пользователь не может присваивать другим пользователям те права, которыми сам не владеет. Но для строгости и абсолютной уверенности в своих действиях, указывать **GRANT OPTION** при удалении прав все же стоит.

- Если нам надо убрать абсолютно все права, а не перечислять их все по одному, то в списке удаляемых следует прописать конструкцию **ALL PRIVILEGES** и дополнить ее конструкцией **GRANT OPTION**. В этом случае – если мы в списке удаляемых прав указываем как **ALL PRIVILEGES**, так и **GRANT OPTION** – указывать объект применения прав (базу данных или таблицу) указывать нельзя:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'new_user'@'localhost';
```

Если же мы укажем только **ALL PRIVILEGES**, то объект применения прав надо прописывать обязательно:

```
REVOKE ALL PRIVILEGES
ON *.*
FROM 'new_user'@'localhost';
```

ИСПОЛЬЗОВАНИЕ РОЛЕЙ

- Представим ситуацию – у нас есть множество пользователей, у которых установлены одинаковые права. Однако в определенный момент таким пользователям понадобилось установить новое право – это значит, что мы должны повторить одни и те же действия для каждого пользователя. Было бы гораздо удобнее объединить пользователей с одинаковыми правами в некие группы и давать этим группам, а не каждому пользователю в отдельности. К счастью, в MySQL есть такой механизм – он осуществляется при помощи ролей.
- Чтобы создать роль, необходимо применить конструкцию **CREATE ROLE**, после которого должно следовать название роли:

```
CREATE ROLE 'site_owner';
```

Можно даже создать несколько ролей одновременно, перечислив их названия через запятую:

```
CREATE ROLE 'site_developer', 'site_administrator';
```

- Чтобы назначить права для каждой роли, используется точно такой же синтаксис, какой использовался для назначения прав для пользователя. Только вместо имени пользователя надо подставлять название роли:

```
GRANT ALL PRIVILEGES
ON site.*
TO 'site_owner' WITH GRANT OPTION;

GRANT INSERT, UPDATE, DELETE, SELECT, EXECUTE
ON site.* TO 'site_developer';

GRANT SELECT, DROP
ON site.*
TO 'site_administrator';
```

- Удаляются права у роли тоже схожим с пользователями образом – при помощи ключевого слова **REVOKE**. Для примера удалим у роли *site_developer* право запускать функции и процедуры:

```
REVOKE EXECUTE
ON site.*
FROM 'site_developer';
```

- Подробный просмотр прав роли осуществляется посредством команды `SHOW GRANTS FOR ...` после которой должно находиться имя роли – опять идентично пользователю:

```
SHOW GRANTS FOR site_developer;
```

```
Grants for site_developer@%
```

```
GRANT USAGE ON *.* TO `site_developer`@`%`
GRANT SELECT, INSERT, UPDATE, DELETE ON `site`.* TO
`site_developer`@`%`
```

- Теперь назначим роли для пользователей – это осуществляется при помощи конструкции **GRANT ... TO ...**:

```
-- для начала создадим два новых пользователя
CREATE USER 'mary_owner'@'localhost' IDENTIFIED BY '01234';

CREATE USER 'john_employee'@'localhost' IDENTIFIED BY
'56789';

-- назначим новым пользователям роли
GRANT 'site_owner' TO 'mary_owner'@'localhost';

GRANT 'site_developer', 'site_administrator' TO
'john_employee'@'localhost';
```

- Если мы зайдем под каким-нибудь из упомянутых новых пользователей в СУБД, то с удивлением обнаружим, что у нас нет никаких прав, и мы не можем производить необходимые для нас операции. Дело в том, что по умолчанию даже созданные роли выключены – для использования этих прав надо эти роли сначала включить, причем включить для каждого пользователя при помощи команды **SET DEFAULT ROLE ... TO ...** :

```
SET DEFAULT ROLE ALL TO
'mary_owner'@'localhost', 'john_employee'@'localhost';
```

В этом случае пользователи действительно смогут пользоваться своими правами. Важный момент – если вместо слова **ALL** указать только конкретную роль, будет активирована только она. Кстати, все вышеописанное поведение можно изменить, если в конфигурационном файле `my.ini` в разделе `[mysqld]` поместить настройку `activate-all-roles-on-login=1`. Если мы так сделаем, то для активации ролей команду **SET DEFAULT ROLE** применять не надо.

- Для просмотра своих активных ролей пользователь должен воспользоваться функцией **CURRENT_ROLE()**. Например, если ей воспользуется пользователь `john_employee'@'localhost`, то он увидит следующее:

```
SELECT CURRENT_ROLE();
```

```
current_role()
```

```
`site_administrator`@`%`,`site_developer`@`%`
```

- Если мы хотим выключить все роли для пользователя, то мы должны применить ту же конструкцию **SET DEFAULT ROLE**, только вместо **ALL** указать **NONE**:

```
SET DEFAULT ROLE NONE TO
'mary_owner'@'localhost', 'john_employee'@'localhost';
```

- При желании роль можно не только выключить, но и полностью отнять у пользователя – для этого применяется знакомая нам команда **REVOKE**. Попробуем с ее помощью отнять роль *site_administrator* у пользователя *john_employee@localhost*:

```
REVOKE site_administrator FROM 'john_employee'@'localhost';
```

Отныне пользователь *john_employee@localhost* может пользоваться только одной ролью:

```
current_role()
```

```
`site_developer`@`%`
```

- Чтобы окончательно и необратимо удалить роль, надо применить команду **DROP ROLE**:

```
DROP ROLE site_administrator;
```


РЕЗЕРВНЫЕ КОПИИ

ВВЕДЕНИЕ

- Резервное копирование является важной процедурой для обеспечения сохранности данных. Ни одна система не защищена от сбоев, и в таких случаях важно иметь резервную копию информации. Если мы управляем базой данных, которая содержит критически важную информацию, то потеря этих данных может иметь серьезные последствия для нашего бизнеса. Резервное копирование минимизирует риски потери данных и обеспечивает возможность восстановления системы. В MySQL существует утилита *mysqldump*, которая идеально подходит для такой цели. Кроме того, есть специальные команды **SELECT INTO OUTFILE** и **SELECT INTO DUMPFILE**, которые мы также рассмотрим.
- Перед началом копирования создадим две базы данных и три таблицы с данными – эти сущности мы и будем использовать:

```
CREATE DATABASE company;
CREATE DATABASE documents;

CREATE TABLE company.employees(
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);
CREATE TABLE company.salaries(
    employee_id INT UNSIGNED NOT NULL,
    salary DECIMAL(10,2) NOT NULL,
    FOREIGN KEY(employee_id) REFERENCES company.employees(id)
);
CREATE TABLE documents.documents(
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    content TEXT
);

INSERT INTO company.employees VALUES (1, 'John Smith'), (2, 'Mary Sue');

INSERT INTO company.salaries VALUES (1, 1000), (2, 1200);

INSERT INTO documents.documents VALUES (1, 'Important Document 1'), (2, 'Important Document 2');
```

ИСПОЛЬЗОВАНИЕ УТИЛИТЫ MYSQLDUMP

- Если мы работаем на ОС Windows, то утилита ***mysqldump*** находится в директории "C:\Program Files\MySQL\MySQL Server 8.0\bin". Перейдем в эту директорию и начнем процесс резервного копирования. Команда копирования в схематическом виде для простой командной строки представлена ниже:

```
.\mysqldump.exe -u[пользователь] -p[пароль] [база] > [файл].sql
```

Здесь есть небольшая заминка - если мы будем использовать не простую командную строку Windows, а утилиту PowerShell, то файл резервной копии сохранится в кодировке UTF-16, что потом может создать проблемы. Эту проблему можно обойти, если вместо символа > применять параметр `--result-file`, который должен равняться файлу, в который копируются данные:

```
.\mysqldump.exe -u[пользователь] -p[пароль] [база]  
--result-file=[файл].sql
```

Предположим, что нам надо скопировать базу данных `documents`. Применим наши новые знания, чтобы это сделать (договоримся, что будем копировать от имени пользователя `root`) - в результате данные должны попасть в файл `documents.sql`.

Командная строка:

```
.\mysqldump.exe -uroot -psecret documents > documents.sql
```

PowerShell:

```
.\mysqldump.exe -uroot -psecret documents  
--result-file=documents.sql
```

- Может случиться, что нам надо скопировать не одну базу данных, а несколько. Утилита `mysqldump` также может нам в этом помочь - общая схема в этом случае следующая (добавляется параметр `--databases` и перечисляются все необходимые базы данных, а не одна).

Командная строка (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] --databases [база база  
база ...] > [файл].sql
```

PowerShell (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] --databases [база база база ...] --result-file=[файл].sql
```

Потребуем информацию из двух баз - company и documents. Эта информация сохранится в файл *company_and_documents.sql*.

Командная строка:

```
.\mysqldump.exe -uroot -psecret --databases company documents  
> company_and_documents.sql
```

PowerShell:

```
.\mysqldump.exe -uroot -psecret --databases company documents  
--result-file=company_and_documents.sql
```

- Иногда у нас нет необходимости копировать всю базу данных, но требуется сохранить резервную копию одной или нескольких таблиц. Чтобы это осуществить, мы должны выполнить команду, которая практически ничем не отличается от самой первой команды для резервного копирования, только после названия базы следует указать таблицы, которые нам необходимы.

Командная строка (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] [база] [таблица1  
таблица2 таблица3 ...] > [файл].sql
```

PowerShell (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] [база] [таблица1  
таблица2 таблица3 ...] --result-file=[файл].sql
```

Допустим, что нам требуется скопировать таблицу employees из базы company. Сохраним данные в файл *company_employees.sql*.

Командная строка:

```
.\mysqldump.exe -uroot -psecret company employees >  
company_employees.sql
```

PowerShell:

```
.\mysqldump.exe -uroot -psecret company employees  
--result-file=company_employees.sql
```

- При неких крайних обстоятельствах нам может понадобиться скопировать все данные всех баз данных. Конечно, мы воспользоваться параметром `--databases` а потом вручную перечислить все базы данных вручную. Однако есть более удобный способ - поставить параметр `--all-databases` и вообще не указывать базы данных - MySQL найдем их все сама.

Командная строка (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] --all-databases > [файл].sql
```

PowerShell (схема):

```
.\mysqldump.exe -u[пользователь] -p[пароль] --all-databases --result-file=[файл].sql
```

Сохраним же абсолютно все наши базы данных в файл *all_databases.sql*.

Командная строка:

```
.\mysqldump.exe -uroot -psecret --all-databases > all_databases.sql
```

PowerShell:

```
.\mysqldump.exe -uroot -psecret --all-databases --result-file=all_databases.sql
```

- При копировании баз данных иногда возникает ситуация, что какие-то из таблиц не нужны. Чтобы их исключить из финальной копии, надо воспользоваться параметром `--ignore-table`, который должен равняться игнорируемой таблице (в формате *название_базы_данных.название_таблицы*). Если надо проигнорировать несколько таблиц, то параметр `--ignore-table` следует применить несколько раз. Воспользуемся этой настройкой, чтобы получить базу данных *company* без таблицы *salaries*.

Командная строка:

```
.\mysqldump.exe -uroot -psecret --ignore-table=company.salaries company > company_without_salaries.sql
```

PowerShell:

```
.\mysqldump.exe -uroot -psecret
--ignore-table=company.salaries company
--result-file=company_without_salaries.sql
```

- Во всех предыдущих примерах мы использовали только самые простые базы данных и таблиц. Однако мы должны помнить, что еще существуют функции, процедуры, триггеры, представления и события. В случае этих сущностей утилита *mysqldump* ведет себя несколько по другому. Дело в том, что по умолчанию в резервную копию дополнительно включаются только триггеры и представления – все остальные объекты полностью игнорируются. Однако у нас есть возможность их скопировать – для этого при копировании надо указывать соответствующие параметры (они могут применяться как по одному, так и все сразу одновременно):
 - **--routines** – будут скопированы также функции и процедуры
 - **--events** – копируются события
 - **--skip-triggers** – в данном случае обратная ситуация, т.к. целевые объекты, т.е. триггеры, копироваться не будут
 - **--no-data** – если мы хотим, чтобы копировались только структуры таблиц, а данные были бы проигнорированы
- При создании резервных копий мы часто можем столкнуться с тем, что нам запрещено делать то или иное действие. Дело в том, что система прав MySQL распространяется на копирование также, как и на все другие явления нашей СУБД. Запомним следующие основные пары прав и соответствующие им сущности (полный список можно найти по ссылке внизу страницы)²:
 - **SELECT** – позволяет копировать базы данных, таблицы и их данные, а также дает возможность копировать хранимые процедуры и функции
 - **SHOW VIEW** – разрешает копировать представления
 - **TRIGGER** – разрешает копировать триггеры

² <https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html>

ИСПОЛЬЗОВАНИЕ КОМАНДЫ SELECT INTO OUTFILE/DUMPFIL

- С одной стороны внутренняя MySQL команда **SELECT INTO OUTFILE** лишена тех шикарных возможностей, которыми обладает `mysqldump` – она может копировать только строки каких-либо таблиц, но не умеет копировать базы данных, триггеры, события, функции и процедуры. Однако она умеет обрабатывать и менять значения столбцов на лету, сохранять выборку из нескольких таблиц одновременно, фильтровать сохраняемые данные, а также копировать данные в нестандартном для SQL формате.
- Перед началом использования команды следует знать, что копировать данные с ее помощью можно только в одну директорию, которая определяется настройкой `secure-file-priv` в файле `my.ini`. В Windows по умолчанию эта настройка равняется `"C:/ProgramData/MySQL/MySQL Server 8.0/Uploads"`.
- Внизу представлена общая схема использования командой копирования – она довольно очевидна, поэтому комментировать подробно ее не будем:

```
SELECT столбец, столбец
INTO OUTFILE путь_к_файлу
FIELDS TERMINATED BY 'символы_разделения_значений_столбцов'
OPTIONALLY ENCLOSED BY 'символы_оборачивающие_текстовые_значение'
LINES TERMINATED BY 'символы_разделителя_строк'
FROM название_таблицы;
```

- Попробуем, наконец применить наши новые знания, чтобы скопировать все данные из таблицы `employees` базы данных `company`. Строки будем разделять символом `\n` (переход на другую строку), значения текстов оборачивать в двойные кавычки, а разделителем между значениями сделаем запятую:

```
SELECT id, name
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/employees.txt'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM employees;
```

Данные будут сохранены в следующем виде (employees.txt):

```
1,"John Smith"
2,"Mary Sue"
```

- Теперь попробуем сделать то, что обещали – сделать выборку из двух таблиц – будем копировать имена работников из таблицы employees и их зарплаты из таблицы salaries. Разделители оставим такими же, как и в первом запросе:

```
SELECT name, salary
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/employees_salaries.txt'
FIELDS TERMINATED BY '|'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM employees
JOIN salaries ON employees.id = salaries.employee_id;
```

```
"John Smith"|1000.00
"Mary Sue"|1200.00
```

- У команды **SELECT INTO OUTFILE** есть альтернативная команда **SELECT INTO DUMPFILE**, которая не поддерживает разделители между строками и значениями, а также обрачиваемые значения. Она сохраняет все выбранные значения столбцов в виде одной длинной строки. Эта команда полезна в том случае, если мы храним в столбце какую-то часть файла в бинарном формате и хотим получить в резервной копии весь этот файл целиком. Применим **SELECT INTO DUMPFILE** чтобы соединить все данные столбца content (таблица documents в базе данных documents) – он не является бинарным, но мы это сделаем просто для примера:

```
SELECT content
INTO DUMPFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/documents.txt'
FROM documents;
```

```
Important Document 1Important Document 2
```

- Для осуществления команд **SELECT INTO OUTFILE** и **SELECT INTO DUMPFIL** пользователю должно быть присвоено право **FILE** (разрешает доступ к файлам на том компьютере, где установлен сервер).

ВОССТАНОВЛЕНИЕ ДАННЫХ ИЗ РЕЗЕРВНЫХ КОПИЙ

- После создания резервных копий рано или поздно может прийти момент, когда придется их применить. Перед применением надо запомнить одно простое правило – копии сделанные с помощью утилиты *mysqldump* по умолчанию стирают все существующие данные, а уже потом вставляют свои. Например, если у нас есть база данных *documents*, а мы пытаемся восстановить данные из ранее сохраненной копии *documents*, то существующая база сначала будет удалена, и ее полностью заменит восстановленная. Тот же самый вариант с таблицами – если мы восстанавливаем таблицу *employees* из базы данных *company*, то существующая таблица удалится, а только потом на ее места встанет резервная копия. Т.е. по умолчанию дополнение сущностей не происходит, происходит их полная замена.
- Восстановление резервных копий осуществляется при помощи утилиты *mysql*, которая находится в уже знакомой для нас директории **"C:\Program Files\MySQL\MySQL Server 8.0\bin"**. Чтобы восстановить базы данных *company* и *documents* (предположим, что они были удалены), мы должны выполнить в простой командной строке Windows следующую команду, используя файл *company_and_documents.sql*, созданный в прошлой главе при помощи атрибута `--databases` и перечисления интересующих нас баз данных:

```
.\mysql.exe -uroot -psecret < company_and_documents.sql
```

Если мы используем не простую командную строку, а PowerShell, то синтаксис будет немного отличаться, т.к. для PowerShell запрещен символ `<`:

```
Get-Content C:\Users\documents_and_company.sql | .\mysql.exe -uroot -psecret
```

- Иногда нам надо восстановить не все базы данных, находящиеся в резервной копии, а только одну. В этом случае необходимо применить атрибут `--one-database` и указать название

какой-либо одной базы из тех, чьи данные хранятся в копии. Для примера восстановим базу данных company:

Командная строка:

```
.\mysql.exe -uroot -psecret --one-database company < company_and_documents.sql
```

PowerShell:

```
Get-Content documents_and_company.sql | .\mysql.exe -uroot -psecret --one-database company
```

- В позапрошлой подглаве мы создали резервную копию documents.sql в которой хранилась информация базы данных documents. При создании мы использовали только название базы и не применяли никаких дополнительных параметров. Если мы заглянем внутрь файла, то увидим, что в этом случае там находятся только таблицы и их данные, но нет никакой явной информации о том, какой базе данных эти данные принадлежат (только комментарии). Поэтому при восстановлении данного файла нужно всегда обязательно указывать ту базу данных, в которую будут восстанавливаться данные из файла. Она не обязательно должна называться documents – она просто должна уже существовать внутри СУБД. Дополнительно отметим, что при восстановлении в этой базе не будут предварительно удалены все таблицы, а только те, которые имеют одинаковое название с таблицами из резервной копии.

Командная строка:

```
.\mysql.exe -uroot -psecret documents < documents.sql
```

PowerShell:

```
Get-Content documents.sql | .\mysql.exe -uroot -psecret documents
```

- Приведем еще один пример с файлом company_employees.sql в который в позапрошлой подглаве мы сохранили таблицы employees из базы данных company. Здесь, как и в прошлом пункте, отсутствует явная информация о базе данных. Поэтому при восстановлении этой таблицы нам надо обязательно указать ту базу, куда мы будем помещать зарезервированные данные. Опять же, ее название не имеет значения, она просто должна быть.

Командная строка:

```
.\mysql.exe -uroot -psecret company < company_employees.sql
```

PowerShell:

```
Get-Content company_employees.sql | .\mysql.exe -uroot  
-psecret company
```

- Если мы еще помним, то для резервного копирования мы использовали не только утилиту `mysqldump`, но и внутренние команды MySQL **SELECT INTO OUTFILE** и **SELECT INTO DUMPFILE**. Очевидно, что результат выполнения второй команды сложно использовать для восстановления данных (они склеиваются в одну большую строку – мы не сможем вычлениить из этой строки значения конкретных столбцов), однако данные, полученные при помощи команды **SELECT INTO OUTFILE** можно попытаться восстановить при помощи другой команды – **LOAD DATA INFILE**.
- Команда **LOAD DATA INFILE** не отличается особой гибкостью, а также она имеет одно ограничение – в файле резервной копии значения столбцов должны располагаться в том же порядке, в каком они были перечислены при создании таблицы. В противоположном случае данные могут попасть в неверные столбцы. К счастью в нашем прошлой подглаве мы как раз сохранили столбцы таблицы `employees` из базы данных `company` как раз в необходимом порядке. Представим, что наша таблица `employee` по прежнему существует, но ее данные по какой-то причине пропали – применим нашу новую команду, чтобы их восстановить:

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server  
8.0/Uploads/employees.txt'  
INTO TABLE employees  
FIELDS TERMINATED BY ','  
OPTIONALLY ENCLOSED BY ''''  
LINES TERMINATED BY '\n';
```

- Если файл с резервной копией находится на том компьютере, где установлен MySQL, то у пользователя, который осуществляет восстановление копии, должно быть право **FILE**. Если пользователь использует файл, находящийся на своем компьютере, то право **FILE** может отсутствовать.

ЖУРНАЛИРОВАНИЕ (ЛОГИРОВАНИЕ)

- При работе с крупными системами или же системами, где хранятся данные строгой отчетности, важную роль играет проблема стабильной работы СУБД, ее безопасности, а также тщательного мониторинга всех возможных ошибок и опасностей. Для решения этой проблемы в качестве основного инструмента применяется т.н. журналирование – фиксация всех основных событий внутри баз данных и запись этих данных в специальные журналы (логи). Журналирование позволяет вам отслеживать, кто и когда вносил изменения в базу данных, а также предоставляет информацию о том, какие операции были выполнены и какие данные были изменены. Это может быть полезным в случае, если нужно восстановить данные после ошибочных изменений или если произошла атака на базу данных.
- MySQL поддерживает различные методы логирования данных, такие как журналы ошибок, журналы медленных запросов, бинарное и глобальное журналирование. Мы можем выбрать подходящий метод в зависимости от потребностей, а можем включить все сразу. Начнем с самого всеохватывающего метода – глобального журналирования. Включить его можно двумя способами. Во-первых, в файле `my.ini` в разделе `[mysqld]` следует приравнять переменную `general-log` к единице:

```
general-log=1
```

Во-вторых, будучи залогиненным на сервере можно выполнить следующую команду:

```
SET GLOBAL general_log = 'ON';
```

Далее мы должны выбрать куда именно будет записываться данные о наших событиях. Если в `my.ini` переменная `log-output` имеет значение `TABLE`, информация будет попадать в таблицу `mysql.general_log`. Однако чаще для этих целей используют файл. Чтобы информация начала в него записываться, сначала в качестве значения переменной `log-output` надо установить слово `FILE`. Затем, в том же самом файле `my.ini` следует найти переменную `general_log_file` и в качестве ее значения установить путь к тому файлу, где у нас будет храниться информация (это может любой файл, какой мы захотим).

```
log-output=FILE
general-log=1
general_log_file=/path/to/file
```

В результате в наш лог будут попадать вся информация о подключении к базам данных (имя пользователя и время подключения), абсолютно все запросы (как для работы с данными, так и для модификации структуры баз данных – например запрос на создание таблицы), а также время включения и выключения сервера баз данных.

- Далее разберем журнал ошибок. У него нет специальной таблицы в базе данных mysql, поэтому его ошибки можно увидеть либо работая в командной строке (отображение ошибок таким образом правильно называется запись в поток stderr), либо назначив специальный файл, куда эти ошибку будут последовательно заноситься. Чтобы назначить такой файл, надо в файле my.ini найти переменную log-error и вписать в нее путь этому файлу.

```
log-error=/path/to/file
```

Больше ничего не требуется (нам не надо вручную включать и выключать этот метод журналирования). Что-же конкретно будет записываться в журнал ошибок? Во-первых, туда попадет информация о том, правильно или неправильно был запущен или приостановлен MySQL сервер. Во-вторых, в процессе работы сервера в журнал могут попадать системные ошибки, например, информация о том, что данные некой таблицы оказались поврежденными, и ситуация требует вмешательства. К сожалению, журнал ошибок не предназначен для хранения ошибок запросов. Таким образом, если мы будем использовать неправильный синтаксис запроса, либо обратимся к несуществующей базе данных, данные об этом в log-error не попадут. При необходимости этим должно заниматься приложение, которое использует MySQL в качестве хранилища.

- Третий метод журналирования – журналирование медленных запросов. Его, как и глобальное журналирование, надо включать и выключать явно, однако по умолчанию он включен. Если же его кто-то выключил, то включается он двумя способами. Во-первых, мы можем сделать это в файле my.ini – для этого надо найти переменную slow-query-log и приравнять ее значение к единице:

slow-query-log=1

Во-вторых, после аутентификации на MySQL сервере можно выполнить следующую команду:

```
SET GLOBAL slow_query_log = 'ON';
```

Теперь скажем несколько слов о том, какие запросы считаются медленными. За это отвечает еще одна переменная в файле `my.ini` – `long_query_time`. В нее можно записать целое число, которое будет означать количество секунд, после которого запрос будет идентифицирован как медленный и будет записан в соответствующий журнал. По умолчанию эта переменная равна десяти, но мы можем поставить любое другое число:

long_query_time=10

Как и в случае с глобальным журналированием, то, куда записывается информация, определяет переменная `log-output`. Если она равна `TABLE`, то данные о медленных ошибках будут записываться в таблицу `slow_log` системной базы данных `mysql`. Если `log-output` равна `FILE`, то медленные запросы будут фиксироваться в том файле, путь к которому вписан в еще одну переменную файла `my.ini` под названием `slow_query_log_file`.

```
log-output=FILE  
slow-query-log=1  
long_query_time=10  
slow_query_log_file=/path/to/file
```

- Последний вид – логическое журналирование, которое фиксирует только все структурные изменения в базах данных (создание и удаления таблиц, добавление и удаления столбцов и т.д.). Журнал данного вида хранится в двоичном формате и используется для репликации – синхронизации (размещения одной и той же копии) данных на нескольких MySQL серверах с целью повышения отказоустойчивости – если откажет один сервер, данные можно будет получать из другого. Чтобы бинарное логирование заработало, в нашем конфигурационном файле `my.ini` надо прописать следующие переменные:

```
# путь к файлу для бинарного журналирования
log_bin=/path/to/file

# если файл просуществует данное время, то он будет удален
binlog_expire_logs_seconds=2592000

# максимальный размер файла - при превышении создается новый
max_binlog_size=1024M
```

- Ниже представим пример файла `my.ini`, где включены все виды журналирования:

```
log-output=FILE

# глобальное журналирование
general-log=1
general_log_file=/path/to/file

# журналирование ошибок
log-error=/path/to/file

# журналирование медленных ошибок
slow-query-log=1
long_query_time=10
slow_query_log_file=/path/to/file

# бинарное журналирование
binlog_expire_logs_seconds=2592000
max_binlog_size=1024M
log_bin=/path/to/file
```