САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Безопасность систем баз данных

Лекция 3
Учетные записи. Аутентификация

Агафонов Антон Александрович
к.т.н., доцент кафедры ГИиИБ

Самара

План лекции

- MySQL
  - Создание пользователей
  - Аутентификация
- PostgreSQL
  - Создание ролей
  - Аутентификация
- MongoDB
  - Создание пользователей
  - Аутентификация

Для противодействия несанкционированному доступу в большинстве современных СУБД реализована многоуровневая система обеспечения безопасности, включающая три процедуры:

- **идентификация** – назначение пользователю (процессу) уникального имени;
- **аутентификация** – процедура проверки подлинности пользователя, представившего свой идентификатор;
- если пользователь успешно прошел процедуру аутентификации, то сервер осуществляет его **авторизацию** – процедуру предоставления пользователю определенных ресурсов и прав на их использование

Прежде чем пользователь будет авторизован на доступ к определенным данным и на выполнение определенных действий в базе данных, необходимо, чтобы он получил доступ к экземпляру СУБД, что в свою очередь требует, чтобы пользователь был аутентифицирован.

В случае успешной аутентификации для данного пользователя устанавливается соединение, и все действия и запросы к данным, выполняемые в рамках этого соединения, контролируются системой авторизации в соответствии с привилегиями, определенными для этого пользователя.

В случае, если аутентификация не пройдена, соединение не будет установлено.

Аутентификация в СУБД

В контексте баз данных аутентификация пользователя может происходить на разных уровнях (различные методы аутентификации):

- на уровне СУБД;
- на уровне операционной системы;
- на уровне других внешних систем аутентификации.

Доступные методы аутентификации зависят от конкретной СУБД. Общей рекомендацией может являться выбор наиболее защищенного метода аутентификации из доступных для данного экземпляра СУБД.

При развертывании СУБД обычно автоматически создается пользователь, который имеет неограниченные привилегии. В MySQL это пользователь «root», в PostgreSQL – пользователь «postgres». Этих пользователей необходимо защищать надежными паролями и использовать исключительно в задачах администрирования сервера БД.

MySQL. Создание пользователей

```
CREATE USER [IF NOT EXISTS]
    user [auth_option]
    DEFAULT ROLE role [, role ]
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH resource_option [resource_option] ...]
    [password_option | lock_option]

CREATE ROLE [IF NOT EXISTS] role
```

Оператор CREATE USER создает новые учетные записи MySQL и позволяет

- устанавливать тип аутентификации,
- задавать роль по умолчанию,
- задавать SSL/TLS шифрование,
- ограничивать использование ресурсов,
- управлять паролями.

Оператор также определяет, будут ли учетные записи изначально заблокированы или разблокированы.

MySQL. Создание пользователей

Имена учетных записей MySQL состоят из имени пользователя и имени хоста, что позволяет создавать отдельные учетные записи для пользователей с одинаковым именем пользователя, которые подключаются с разных хостов.

- Имя пользователя указывается в формате `'user_name'@'host_name'`.
- Часть `@'host_name'` является необязательной. Имя учетной записи, состоящее только из имени пользователя, эквивалентно `'user_name'@'%'`. Например, `'test_user'` эквивалентно `'test_user'@'%'`.
- При указании хоста кроме конкретных имен узлов и ip-адресов можно использовать значение `'localhost'`, если соединение выполняется с того же компьютера, на котором установлен сервер, а также значения `'%'` и `''`, если соединение допускается с произвольного узла.
- Имя пользователя и имя хоста не нужно заключать в кавычки, если они допустимы как идентификаторы без кавычек.
- Части имени пользователя и имени хоста, если они указаны в кавычках, должны быть записаны отдельно. Т.е. необходимо указывать пользователя как `'me'@'localhost'`, а не `'me@localhost'`. Имя пользователя `'me@localhost'` эквивалентно `'me@localhost'@'%'`.

MySQL. Модификация пользователей

Получение текущего пользователя

```
SELECT CURRENT_USER();
```

Изменение пользователя

```
ALTER USER [IF EXISTS] user [options];
```

Удаление пользователя

```
DROP USER [IF EXISTS] user;
```

MySQL. Аутентификация

- Создание нового пользователя добавляет новую запись в системную таблицу `mysql.user`.
- Проверка подлинности выполняется на основе трех столбцов (`host`, `user` и `authentication_string`). Сервер устанавливает соединение только в том случае, если колонки `host` и `user` в какой-то строке таблицы пользователей совпадают с именем клиентского хоста и именем пользователя, а также клиент предоставляет пароль, совпадающий со значением в столбце `authentication_string`.
- MySQL не хранит пароли в открытом виде, непустые значения `authentication_string` в таблице `user` представляют собой зашифрованные пароли. С точки зрения MySQL, зашифрованный пароль – это и есть действительный пароль, поэтому никому не следует давать к нему доступ. В частности, *не допускается давать неадминистративным пользователям доступ на чтение данных в системной базе данных* `mysql`.
- В случае использования внешней аутентификации значение столбца `authentication_string` может не использоваться.

MySQL. Методы аутентификации

MySQL поддерживает различные методы аутентификации с использованием плагинов:

- Плагин `mysql_native_password`, выполняющий встроенную аутентификацию на основе метода хеширования пароля. Использовался до введения подключаемой аутентификации в MySQL.
- Плагин `sha256_password`, выполняющий аутентификацию на основе сравнения SHA-256 хэшей паролей. Плагин `caching_sha2_password` также использует SHA-256 аутентификацию, но с кэшированием на стороне сервера для лучшей производительности.
- Плагин на стороне клиента `mysql_clear_password`, который отправляет пароль на сервер без хеширования или шифрования. Этот подключаемый модуль используется в сочетании с подключаемыми модулями на стороне сервера, которым требуется пароль в открытом виде.

MySQL. Методы аутентификации

Коммерческая версия MySQL дополнительно поддерживает внешние методы аутентификации:

- Плагин `authentication_windows`, выполняющий внешнюю аутентификацию в Windows, позволяя MySQL Server использовать собственные службы Windows для аутентификации клиентских подключений.
- Плагин `authentication_ldap_simple`, выполняющий аутентификацию с использованием LDAP (облегчённого протокола доступа к каталогам) путем доступа к службам каталогов, таким как X.500.
- Плагин `authentication_kerberos`, выполняющий аутентификацию с использованием сетевого протокола аутентификации Kerberos, позволяющего передавать данные через незащищённые сети.

MySQL. Создание пользователей. Примеры

Создание пользователей

```
CREATE USER 'agafonov'@'localhost' IDENTIFIED BY 'P@ssw0rd';


CREATE USER 'agafonov'@'host'
    IDENTIFIED WITH mysql_native_password BY 'new_password'
    PASSWORD EXPIRE INTERVAL 180 DAY
    FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 2;
```

MySQL. Просмотр пользователей

Выборка пользователей

```
SELECT host, user, Select_priv, Insert_priv, plugin, authentication_string
FROM mysql.user;
```

Результат запроса

	host	user	Select_priv	Insert_priv	plugin	authentication_string
	host	agafonov	N	N	mysql_native_password	*0913BF2E2CE20CE21BFB1 961AF124D49204…
	localhost	agafonov	N	N	caching_sha2_password	A005$Bg*i +)[%%F*J:|RLtuwkHK…
	localhost	root	Y	Y	caching_sha2_password	A005$Fu_□W~~duUD 2}00O…

PostgreSQL. Управление доступом

PostgreSQL использует концепцию **ролей** (*roles*) для управления разрешениями на доступ к базе данных.

Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того, как роль настроена. Роли могут владеть объектами базы данных (например, таблицами и функциями) и выдавать другим ролям разрешения на доступ к этим объектам, управляя тем, кто имеет доступ и к каким объектам. Кроме того, можно предоставить одной роли *членство* в другой роли, таким образом одна роль может использовать права других ролей.

Концепция ролей включает в себя концепцию пользователей («users») и групп («groups»). До версии 8.1 в PostgreSQL пользователи и группы были отдельными сущностями, но теперь есть только роли. Любая роль может использоваться в качестве пользователя, группы, и того и другого.

```
CREATE ROLE имя [ [ WITH ] параметр [ ... ] ]

Параметр:
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | CONNECTION LIMIT предел_подключений
    | [ ENCRYPTED ] PASSWORD 'пароль' | PASSWORD NULL
    | VALID UNTIL 'дата_время'
    | IN ROLE имя_роли [, ...]
    | ROLE имя_роли [, ...]
    | ADMIN имя_роли [, ...]


CREATE USER имя [ [ WITH ] параметр [ ... ] ]
…
    | LOGIN | NOLOGIN
```

PostgreSQL. Создание ролей

Параметр	Описание
SUPERUSER | **NOSUPERUSER**	Параметры определяют, будет ли эта роль «суперпользователем», который может переопределить все ограничения доступа в базе данных.
CREATEDB | **NOCREATEDB**	Параметры определяют, сможет ли роль создавать базы данных.
CREATEROLE | **NOCREATEROLE**	Параметры определяют, сможет ли роль создавать (изменять, удалять) другие роли.
INHERIT | NOINHERIT	Параметры определяют, будет ли роль «наследовать» права ролей, членом которых она является. Роль с атрибутом INHERIT может автоматически использовать любые права, назначенные всем ролям, в которые она включена, непосредственно или опосредованно.
LOGIN | **NOLOGIN**	Параметры определяют, разрешается ли новой роли вход на сервер, т.е. может ли эта роль стать начальным авторизованным именем при подключении клиента.
REPLICATION | **NOREPLICATION**	Параметры определяют, будет ли роль ролью репликации с возможностью подключения к серверу в режиме репликации.

PostgreSQL. Создание ролей

Параметр	Описание
`CONNECTION LIMIT`	Определяет, сколько параллельных подключений может установить роль.
`PASSWORD`	Задаёт пароль роли.
`VALID UNTIL`	Устанавливает дату и время, после которого пароль роли перестаёт действовать
`IN ROLE`	Перечисляются одна или несколько существующих ролей, в которые будет немедленно включена новая роль.
`ROLE`	Перечисляются одна или несколько существующих ролей, которые автоматически становятся членами создаваемой роли.
`ADMIN`	Подобно `ROLE`, но перечисленные в нём роли включаются в новую роль с атрибутом `WITH ADMIN OPTION`, что даёт им право включать в эту роль другие роли

PostgreSQL. Модификация ролей

Получение текущего пользователя

```
SELECT CURRENT_ROLE;
```

Изменение роли

```
ALTER ROLE имя [ WITH ] параметр [ ... ];
```

Удаление роли

```
DROP ROLE [IF EXISTS] имя;
```

Аутентификация клиентов управляется конфигурационным файлом, который традиционно называется pg_hba.conf и расположен в каталоге с данными кластера базы данных. (HBA расшифровывается как host-based authentication — аутентификации по имени узла.)

Формат файла представляет собой набор записей, по одной в строке. Каждая запись включает в себя:

- тип соединения,
- диапазон IP-адресов клиента,
- имя базы данных, имя пользователя,
- способ аутентификации, который будет использован для соединения в соответствии с этими параметрами.

Первая найденная запись с соответствующим типом соединения, адресом клиента, указанной базой данных и именем пользователя применяется для аутентификации, если аутентификация не прошла, последующие записи не рассматриваются. Если же ни одна из записей не подошла, в доступе будет отказано.

PostgreSQL. Методы аутентификации

Поддерживаются следующие методы аутентификации:

1. Простая аутентификация (**trust**, **reject**). Метод «**trust**» безусловно доверяет пользователю и не выполняет аутентификацию. Метод «**reject**» безусловно отказывает в доступе. Можно использовать, чтобы отсечь любые соединения определенного типа или с определенных адресов (например, запретить нешифрованные соединения).

2. Аутентификация по паролю (**password**, **md5**, **scram-sha-256**, **LDAP**, **RADIUS**, **PAM**). Эти методы аутентификации запрашивают у пользователя пароль и проверяют его на соответствие паролю, который хранится либо в самой СУБД, либо во внешней службе.

   a) метод «password» сравнивает пароли в открытом вид;

   b) метод «md5» сравнивает MD5-хэш пароля с MD5-хэшем, хранящимся в базе;

   c) метод «scram-sha-256» (наиболее безопасный) использует протокол SCRAM и криптостойкий алгоритм SHA-256;

   d) методы «LDAP», «RADIUS» и «PAM» используют для хранения паролей сервер LDAP, сервер RADIUS или подключаемый модуль аутентификации (Pluggable Authentication Module) соответственно.

3. Внешние системы аутентификации (**peer**, **cert**, **GSSAPI**, **SSPI**, **Ident**). Метод подразумевает аутентификацию пользователя внешним сервером аутентификации вне СУБД. В результате успешной аутентификации PostgreSQL получает имя пользователя, идентифицированное внешней системой (внешнее имя). Внешние имена пользователей могут не совпадать с именами пользователей базы данных, в этом случае, для сопоставления имен используют файл `pg_ident.conf`.
   a) метод «peer» запрашивает имя пользователя у ядра ОС. Поскольку ОС уже аутентифицировала этого пользователя, СУБД доверяет ей;
   b) метод «cert» использует аутентификацию на основе клиентского сертификата и предназначен только для SSL-соединений;
   c) метод «GSSAPI» использует аутентификацию Kerberos по протоколу GSSAPI. Поддерживается автоматическая аутентификация;
   d) метод «SSPI» использует аутентификацию Kerberos или NTLM для систем на Windows. Поддерживается автоматическая аутентификация;
   e) метод «Ident» запрашивает имя пользователя ОС клиента от сервера Ident. Способ доступен только для подключений по TCP/IP.

PostgreSQL. Примеры записей pg_hba.conf

```
# Позволяет любому пользователю локальной системы подключаться к любой базе данных,
# используя любое имя пользователя баз данных, через Unix-сокеты
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             all                                     trust

# Позволяет любому пользователю компьютера 192.168.12.10 подключаться к базе данных "postgres",
# если он передаёт правильный пароль.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    postgres        all             192.168.12.10/32        scram-sha-256

# Позволяет любым пользователям с компьютеров в домене example.com
# подключаться к любой базе данных, если передаётся правильный пароль.
#
# Для всех пользователей требуется аутентификация SCRAM, за исключением
# пользователя 'mike', который использует старый клиент, не поддерживающий
# аутентификацию SCRAM.
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             mike            .example.com            md5
host    all             all             .example.com            scram-sha-256
```

Создание ролей

```
CREATE ROLE agafonov WITH LOGIN PASSWORD 'P@ssw0rd';

CREATE USER agafonov WITH PASSWORD 'P@ssw0rd';


CREATE ROLE ivanov WITH LOGIN PASSWORD 'pass'
       VALID UNTIL '2023-01-01';

CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

PostgreSQL. Просмотр ролей

Выборка пользователей

```
SELECT * FROM pg_user;
```

	username	usecreatedb	usesuper	userepl	passwd	valuntil
	postgres	true	true	true	*******	null
	agafonov	false	false	false	*******	null
	ivanov	false	false	false	*******	2023-01-01 00:00:00+04

```
SELECT * FROM pg_roles;
```

	rolname	rolsuper	rolinherit	rolcanlogin	rolpassword	rolvaliduntil
	pg_database_owner	false	true	false	*******	null
	…	…	…	…	…	…
	postgres	true	true	true	*******	null
	agafonov	false	false	false	*******	null
	ivanov	false	false	false	*******	2023-01-01 00:00:00+04

MongoDB . Методы аутентификации

Версия MongoDB Community поддерживает ряд механизмов аутентификации:

- SCRAM (по умолчанию) для аутентификации пользователей по паролю. Поддерживаются метод SCRAM-SHA-1, использующий хеш-функцию SHA-1, и метод SCRAM-SHA-256, использующий хеш-функцию SHA-256.
- Аутентификация на основе сертификата x.509 использует сертификаты вместо имени пользователя и пароля. Цифровой сертификат x.509 использует общепринятый стандарт инфраструктуры открытых ключей (PKI) x.509 для проверки того, что открытый ключ принадлежит тому, кто его предъявляет.

Кроме того, версии MongoDB Atlas и MongoDB Enterprise поддерживают следующие механизмы аутентификации:

- Аутентификация с использованием протокола LDAP (облегчённый протокол доступа к каталогам)
- Аутентификация с использованием сетевого протокола аутентификации Kerberos.

Эти механизмы позволяют MongoDB интегрироваться в существующую систему аутентификации.

MongoDB . Настройка аутентификации

- Пользователь в MongoDB создается в определенной базе данных. Эта база данных является базой данных аутентификации пользователя; для этой цели можно использовать любую базу данных. Однако привилегии пользователя не ограничиваются базой данных аутентификации.
- Сначала создается пользователь-администратор, а затем создаются дополнительные пользователи в БД для каждого пользователя/приложения, которое обращается к системе.
- Рекомендуется следовать принципу наименьших привилегий, т.е. сначала создать роли, определяющие точные права доступа, требуемые набором пользователей. Затем создать пользователей и назначить им только те роли, которые необходимы для выполнения их операций.

MongoDB . Использование SCRAM аутентификации

Настройка SCRAM-аутентификации для сервера, запущенного без режима репликации:

1. Запустить экземпляр сервера MongoDB без контроля доступа (режим по умолчанию). Подключиться к серверу.
2. Создать пользователя-администратора:

```
use admin
db.createUser( {
    user: "myUserAdmin",
    pwd: passwordPrompt(), // или указать пароль
    roles: [
       { role: "userAdminAnyDatabase", db: "admin" },
       { role: "readWriteAnyDatabase", db: "admin" }
    ]
} )
```

3. Перезапустить экземпляр MongoDB в режиме контроля доступа:
   - если экземпляр mongod запускается из командной строки, необходимо добавить параметр `--auth`:

```
mongod --auth --port 27017 --dbpath /var/lib/mongodb
```

   - если экземпляр mongod запускается с помощью файла конфигурации, необходимо добавить параметр файла конфигурации security.authorization:

```
security:
    authorization: enabled
```

4. Подключиться к серверу с указанием опций аутентификации или выполнить аутентификацию после подключения:

```
use admin
db.auth("myUserAdmin", passwordPrompt()) // или указать пароль
```

MongoDB . Создание пользователя

После аутентификации в качестве администратора методом `db.createUser()` можно создавать дополнительных пользователей. Пользователям можно назначать любые встроенные или определяемые пользователем роли.

```
use db_lectures
db.createUser( {
    user: "myTester",
    pwd: passwordPrompt(), // или указать пароль
    roles: [
       { role: "readWrite", db: "db_lectures" },
       { role: "read", db: "wikipedia" }
    ]
} )
```

Чтобы переключиться на нового пользователя, можно использовать метод `db.auth`.

```
use db_lectures
db.auth("myTester", passwordPrompt()) // или указать пароль
```

Проверка чтения из БД db_lectures

```
use db_lectures
db.inventory.find( ).count()
```

11

Проверка чтения из БД wikipedia

```
use wikipedia
db.wikipedia.find( ).count()
```

15138138

Проверка записи в БД wikipedia

```
use wikipedia
db.wikipedia.insertOne( {article: "New record" } );
```

```
MongoServerError: not authorized on wikipedia to execute command
```

Проверка чтения из БД analytics

```
use analytics
db.accounts.find( ).count()
```

```
MongoServerError: not authorized on analytics to execute command
```

MongoDB . Просмотр пользователей

Получение списка пользователей из БД db_lectures

```
db.getUsers( )
```

```
{
    _id: "db_lectures.myTester", userId: UUID("4a0869c7-0477-45f3-b637-133c8a58bda5"),
    user: "myTester", db: "db_lectures",
    roles: [Array], mechanisms: [Array]
}
```

Получение указанного пользователя из БД db_lectures

```
db.getUser("myTester")
```

```
{
    _id: "db_lectures.myTester", userId: UUID("4a0869c7-0477-45f3-b637-133c8a58bda5"),
    user: "myTester", db: "db_lectures",
    roles: [{role: "read", db: "wikipedia" }, { role: "readWrite", db: "db_lectures"}],
    mechanisms: ["SCRAM-SHA-1", "SCRAM-SHA-256"]
}
```

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

БЛАГОДАРЮ ЗА ВНИМАНИЕ

Агафонов А.А.
к.т.н., доцент кафедры ГИиИБ