Тема 10. Безопасность баз данных. Администрирование баз данных. Настройка прав доступа и пользователей. Управление пользователями и ролями. Системы управления доступом (DBMS). Шифрование данных. Брандмауэры.

Цель занятия:

Ознакомиться с основами администрирования MySQL.

Учебные вопросы:

1. Введение в безопасность баз данных.
2. Администрирование баз данных.
3. Управление пользователями и настройка прав доступа.
4. Управление ролями.
5. Шифрование данных.
6. Защита с помощью брандмауэров.
7. Практические меры безопасности

1. Введение в безопасность баз данных.

Безопасность баз данных — это комплекс мер и технологий, направленных на защиту данных, хранящихся в базе данных, от несанкционированного доступа, потери, утечек или повреждений.

Это включает как защиту на уровне программного обеспечения (например, настройка доступа, шифрование), так и на уровне инфраструктуры (например, использование брандмауэров, защита серверов).

Основные угрозы для баз данных:

- **Несанкционированный доступ**. Это когда злоумышленники получают доступ к базе данных без разрешения. Они могут украсть, изменить или удалить данные, что может привести к серьезным последствиям, таким как потеря конфиденциальной информации или повреждение структуры данных.
- **Утечки данных**. Происходят, когда конфиденциальные данные, такие как персональная информация или финансовые записи, становятся доступными неавторизованным лицам. Это может произойти из-за слабой защиты, ошибок в конфигурации, либо целенаправленных атак.
- **SQL-инъекции**. Вид атаки, при котором злоумышленники вводят вредоносные SQL-запросы через формы ввода или параметры URL. Цель таких атак — манипулировать базой данных, изменяя или удаляя данные, или получать доступ к конфиденциальной информации.

Зачем нужна защита данных в MySQL?

- **Защита конфиденциальной информации**. В базах данных часто хранятся чувствительные данные, такие как персональные данные пользователей, финансовые записи, медицинские данные и т.д. Их утечка может привести к значительным репутационным и юридическим последствиям для организации.
- **Поддержание целостности данных**. Защита данных помогает предотвратить несанкционированные изменения, которые могут привести к искажению данных, нарушению бизнес-процессов и некорректной работе приложений.
- **Соблюдение нормативных требований**. Многие страны и отрасли имеют строгие требования к защите данных (например, GDPR в ЕС, HIPAA в США). Организациям, работающим с базами данных, необходимо соответствовать этим нормам, чтобы избежать штрафов и других санкций.
- **Защита от атак и вредоносных действий**. Наличие мер безопасности помогает предотвратить и минимизировать последствия атак, таких как DDoS, SQL-инъекции, кражи учетных записей и другие вредоносные действия, направленные на базы данных.

2. Администрирование баз данных.

Администратор базы данных (DBA) отвечает за поддержание стабильной, безопасной и эффективной работы базы данных. **Основные задачи DBA включают**:

- **Установка и настройка базы данных**. Установка серверов MySQL, их первоначальная настройка, выбор правильных конфигураций для оптимальной производительности и безопасности.
- **Управление пользователями и правами доступа**. Создание учетных записей пользователей, назначение и регулирование их прав доступа, управление ролями для обеспечения безопасной и четко организованной работы с базой данных.
- **Обеспечение безопасности базы данных**. Настройка правил безопасности, шифрование данных, защита от атак и SQL-инъекций. DBA также отвечает за регулярное обновление ПО для устранения уязвимостей.

- **Мониторинг и оптимизация производительности**. Мониторинг производительности базы данных, анализ нагрузок, устранение узких мест, настройка индексов и кэширования для ускорения выполнения запросов.
- **Резервное копирование и восстановление данных**. Настройка регулярного резервного копирования данных, контроль за его выполнением. DBA отвечает за быструю и эффективную восстановку данных в случае сбоев или потерь.
- **Управление обновлениями и патчами**. Регулярное обновление MySQL и связанных инструментов для внедрения новых функций и исправлений безопасности.
- **Решение проблем и устранение неполадок**. Отслеживание и устранение ошибок, анализ логов и выполнение аварийного восстановления при возникновении проблем с базой данных.

Обзор инструментов для администрирования в MySQL

1. MySQL Workbench

Это мощный инструмент с графическим интерфейсом, предназначенный для администрирования MySQL. Основные функции:

- Управление базами данных: создание, редактирование, удаление баз и таблиц.
- Конструктор запросов: создание, тестирование и выполнение SQL-запросов.
- Мониторинг производительности: встроенные инструменты анализа нагрузки на сервер.
- Визуальный редактор схем базы данных: разработка и моделирование структуры базы.
- Инструменты для резервного копирования и восстановления баз данных.
- Интеграция с MySQL Enterprise для управления и мониторинга.

2.Консольные команды MySQL.

Консоль MySQL предоставляет гибкость и возможность выполнения всех действий с базой данных через командную строку. К основным командам для администрирования относятся:

- **mysql** — консольный клиент для подключения и работы с базой данных.
- **mysqldump** — инструмент для резервного копирования базы данных.
- **mysqladmin** — инструмент для выполнения административных задач (перезапуск сервера, проверка состояния и пр.).
- **SHOW и DESCRIBE** — команды для просмотра структуры базы данных и статистики.
- **GRANT и REVOKE** — команды для управления правами пользователей.
- **CREATE DATABASE, DROP DATABASE, ALTER TABLE** — команды для создания, удаления и изменения структур баз данных.

Консольные команды предоставляют администратору более детальный контроль над всеми процессами и позволяют автоматизировать многие задачи с помощью скриптов.

3. Управление пользователями и настройка прав доступа.

Система управления доступом (Access Control System) — это механизм в DBMS (системе управления базами данных), который контролирует, кто и какие действия может выполнять с данными.

В MySQL и других реляционных СУБД контроль доступа осуществляется на основе привилегий, назначаемых пользователям и ролям.

Правильная настройка системы управления доступом важна для обеспечения безопасности данных и предотвращения несанкционированных действий.

Основные уровни управления доступом в DBMS:

- **Уровень пользователя** (User Level). Это основной уровень, на котором назначаются права и роли отдельным пользователям. Пользователь получает доступ к базе данных только через учётную запись, которой присваиваются определённые права.
- **Уровень базы данных** (Database Level). Привилегии могут быть назначены на уровне всей базы данных. Например, можно предоставить пользователю или роли доступ к базе целиком или ограничить его права на выполнение определённых операций.
- **Уровень таблиц** (Table Level). На этом уровне можно назначить права на доступ к отдельным таблицам. Например, одному пользователю можно разрешить только выборку данных, а другому — и выборку, и изменение данных в этой таблице.
- **Уровень строк** (Row Level). В некоторых СУБД (например, Oracle) возможно ограничение доступа к определённым строкам таблицы. В MySQL контроль на уровне строк можно реализовать через использование условий (например, в сочетании с триггерами или политиками безопасности данных).
- **Уровень столбцов** (Column Level). В MySQL можно назначать привилегии на доступ к отдельным столбцам таблицы. Это может быть полезно, если определённые столбцы содержат чувствительные данные, доступ к которым должен быть ограничен для некоторых пользователей.

Создание пользователей в MySQL: команды CREATE USER и DROP USER

Команда CREATE USER используется для создания новых учетных записей пользователей в MySQL. Пример создания пользователя:

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
```

с которого разрешен доступ. Если доступ должен быть разрешен с любого хоста, вместо localhost можно указать %.

'password' — пароль для аутентификации пользователя.

Пример создания пользователя с доступом с любого хоста:

```
CREATE USER 'analyst'@'%' IDENTIFIED BY 'securepassword';
```

Удаление пользователя.

Команда DROP USER используется для удаления пользователя из MySQL:

```
DROP USER 'username'@'localhost';
```

Назначение прав доступа: команда GRANT и ее параметры

Команда GRANT позволяет назначать пользователю определенные привилегии на уровне базы данных, таблиц или даже отдельных столбцов.

Назначение прав на уровне базы данных

Пример предоставления пользователю всех прав на определенную базу данных:

```
GRANT ALL PRIVILEGES ON database_name.* TO 'username'@'localhost';
```

ALL PRIVILEGES — назначение всех прав на указанную базу данных.

database_name.* — указывает на всю базу данных. Звездочка обозначает, что привилегии распространяются на все таблицы базы данных.

Назначение прав на уровне таблицы

Пример предоставления права на выборку и вставку данных только в одну таблицу:

```
GRANT SELECT, INSERT ON database_name.table_name TO 'username'@'localhost';
```

SELECT, INSERT — конкретные привилегии, которые получает пользователь.

database_name.table_name — указание конкретной таблицы, к которой применяются привилегии.

Назначение прав на уровне столбцов

Можно ограничить доступ не только к таблицам, но и к отдельным столбцам в таблице:

```
GRANT SELECT (column1, column2) ON database_name.table_name TO 'username'@'localhost';
```

(column1, column2) — только эти столбцы доступны для выборки.

Управление правами: команды REVOKE и SHOW GRANTS.

Отзыв прав с помощью команды REVOKE.

Команда REVOKE используется для отмены ранее назначенных прав.

Пример отмены права на выборку и вставку данных:

```
REVOKE SELECT, INSERT ON database_name.table_name FROM 'username'@'localhost';
```

Просмотр текущих привилегий с помощью команды SHOW GRANTS

Команда SHOW GRANTS позволяет увидеть, какие привилегии предоставлены конкретному пользователю:

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

Пример вывода привилегий:

```
GRANT SELECT, INSERT ON database_name.table_name TO 'username'@'localhost';
```

Примеры настройки прав для различных пользователей

Администратор базы данных. Администратор должен иметь полные права на базу данных:

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

WITH GRANT OPTION позволяет администратору передавать права другим пользователям.

Аналитик. Аналитику нужны права только на выборку данных из определенных таблиц:

```
GRANT SELECT ON database_name.* TO 'analyst'@'%';
```

Разработчик. Разработчику могут понадобиться права на чтение и запись данных, но без удаления:

```
GRANT SELECT, INSERT, UPDATE ON database_name.* TO 'developer'@'localhost';
```

Настройка привилегий на уровне базы данных, таблиц и столбцов.

На уровне базы данных. Пользователь получает все права на конкретную базу данных:

```
GRANT ALL PRIVILEGES ON database_name.* TO 'username'@'localhost';
```

На уровне таблиц. Ограничение привилегий для конкретной таблицы в базе данных:

```
GRANT SELECT, INSERT ON database_name.table_name TO 'username'@'localhost';
```

На уровне столбцов. Доступ к выборке только определенных столбцов:

```
GRANT SELECT (column1, column2) ON database_name.table_name TO 'username'@'localhost';
```

Такая детальная настройка прав доступа позволяет эффективно управлять безопасностью данных в MySQL, предоставляя пользователям только те права, которые необходимы для их работы.

4. Управление ролями.

Роли в MySQL — это наборы привилегий, которые можно назначать пользователям.

Это позволяет упростить управление доступом в крупных системах, где много пользователей с одинаковыми привилегиями.

Вместо назначения привилегий каждому пользователю отдельно, можно создать роль с определенным набором привилегий и назначать её различным пользователям.

Преимущества использования ролей:

- Упрощение управления привилегиями.
- Легкость в изменении прав для группы пользователей.
- Повышение безопасности за счет централизованного управления доступом.

Создание ролей и назначение пользователям.

Создание ролей. Команда CREATE ROLE используется для создания новой роли. Пример создания роли для аналитиков:

```
CREATE ROLE 'analyst_role';
```

Назначение привилегий роли. Привилегии назначаются роли с помощью команды GRANT. Например, предоставим роли analyst_role права на выборку данных из определенной базы:

```
GRANT SELECT ON database_name.* TO 'analyst_role';
```

Назначение ролей пользователям. После создания роли и назначения ей привилегий, эту роль можно назначить пользователям. Для этого используется команда GRANT:

```
GRANT 'analyst_role' TO 'analyst1'@'localhost';
```

Теперь пользователь analyst1 получает все привилегии, которые были назначены роли analyst_role.

Управление ролями: команды SET ROLE, REVOKE, DROP ROLE.

Активация ролей для пользователя: команда SET ROLE.

После назначения роли пользователю, она может быть активирована для текущей сессии с помощью команды SET ROLE. Пример активации роли:

```
SET ROLE 'analyst_role';
```

Чтобы активировать все роли, назначенные пользователю, можно использовать:

```
SET ROLE ALL;
```

По умолчанию, если не заданы активные роли, у пользователя включены все назначенные ему роли.

Отзыв ролей у пользователей: команда REVOKE.

Если нужно отозвать роль у пользователя, используется команда REVOKE. Пример:

```
REVOKE 'analyst_role' FROM 'analyst1'@'localhost';
```

Удаление ролей: команда DROP ROLE. Роль можно удалить с помощью команды DROP ROLE. Это приведет к удалению роли, но привилегии, уже предоставленные пользователям напрямую, не будут отменены:

```
DROP ROLE 'analyst_role';
```

Примеры использования ролей для организации прав пользователей.

Создадим роль для **аналитиков**, которые могут только просматривать данные:

```
CREATE ROLE 'analyst_role';
GRANT SELECT ON database_name.* TO 'analyst_role';
GRANT 'analyst_role' TO 'analyst1'@'localhost', 'analyst2'@'localhost';
```

Роль для **разработчиков**, которым нужны права на чтение и изменение данных:

```
CREATE ROLE 'developer_role';
GRANT SELECT, INSERT, UPDATE ON database_name.* TO 'developer_role';
GRANT 'developer_role' TO 'dev1'@'localhost', 'dev2'@'localhost';
```

Создадим роль для администратора базы данных, которому нужны все права, включая право на передачу привилегий другим пользователям:

```
CREATE ROLE 'admin_role';
GRANT ALL PRIVILEGES ON *.* TO 'admin_role' WITH GRANT OPTION;
GRANT 'admin_role' TO 'admin1'@'localhost';
```

Активация ролей для пользователя. Пользователь может активировать роль для текущей сессии:

```
SET ROLE 'developer_role';
```

Отключение активной роли. Чтобы отключить роль и работать без привилегий роли:

```
SET ROLE NONE;
```

5. Шифрование данных.

Шифрование данных является важным элементом защиты информации.

Оно помогает предотвратить несанкционированный доступ к конфиденциальным данным как на уровне хранения, так и во время передачи данных по сети.

Основные причины, по которым необходимо шифровать данные в базах данных:

- Защита конфиденциальных данных — шифрование помогает защитить личные данные пользователей, финансовую информацию, коммерческие тайны и другие чувствительные данные.
- Снижение риска утечек — в случае компрометации базы данных зашифрованные данные остаются недоступными без ключа шифрования.
- Соответствие стандартам и требованиям — многие законы и стандарты безопасности данных (GDPR, PCI DSS и другие) требуют обязательного шифрования конфиденциальной информации.
- Защита данных при резервном копировании — шифрование резервных копий предотвращает несанкционированный доступ к данным в случае потери или кражи резервных носителей.

Типы шифрования:

- **Шифрование на уровне файловой системы** (Data-at-Rest Encryption).Это тип шифрования, при котором данные шифруются на уровне хранения — на диске или в файловой системе. При этом все файлы базы данных шифруются целиком. Преимущество этого метода в том, что шифруются все данные базы данных, однако контроль над доступом и шифрованием лежит на уровне операционной системы или внешних систем защиты. Пример технологии — Transparent Data Encryption (TDE).
- **Шифрование на уровне столбцов** (Column-Level Encryption). Этот тип шифрования применяется непосредственно к данным в отдельных столбцах базы данных. Преимущество заключается в том, что можно шифровать только те данные, которые действительно чувствительны (например, номера кредитных карт или пароли). Шифрование на уровне столбцов используется для тонкой настройки безопасности и позволяет более гибко управлять доступом к зашифрованным данным.

Встроенные механизмы шифрования в MySQL

В MySQL существует встроенная поддержка шифрования на уровне таблиц и таблиспейсов для файловой системы.

Основной механизм - **INNODB_TABLESPACES_ENCRYPTION**, который позволяет шифровать данные в InnoDB таблицах и таблиспейсах.

Основные шаги для настройки шифрования InnoDB таблиц:

- Включение поддержки шифрования. Необходимо убедиться, что шифрование таблиц InnoDB включено в конфигурации MySQL:

```
innodb_file_per_table=ON
innodb_encrypt_tables=ON
innodb_tablespace_encrypt=ON
```

- Шифрование таблицы. Пример шифрования таблицы с помощью команды ALTER TABLE:

```
ALTER TABLE sensitive_data ENCRYPTION='Y';
```

- Проверка состояния шифрования. Вы можете проверить состояние шифрования таблицы с помощью команды:

```
SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS
FROM information_schema.tables WHERE CREATE_OPTIONS LIKE '%ENCRYPTION%';
```

Шифрование журналов транзакций. Кроме данных, можно зашифровать журналы транзакций InnoDB, чтобы защитить данные, находящиеся в процессе обработки.

Настройка SSL-соединений для защиты передачи данных.

Чтобы защитить передачу данных между клиентом и сервером MySQL, используется SSL (Secure Sockets Layer) или его современный аналог TLS (Transport Layer Security). SSL-соединения обеспечивают защиту от атак типа «человек посередине», предотвращая перехват данных.

Основные шаги для настройки SSL-соединений:

1.Создание SSL-сертификатов. Для начала необходимо создать или получить сертификаты для сервера и клиента. Обычно это включает:

- Создание корневого сертификата CA (Certification Authority).
- Создание серверного сертификата и ключа.
- Создание клиентского сертификата и ключа.

2.Настройка MySQL для использования SSL.

В конфигурационном файле MySQL (my.cnf) необходимо указать пути к сертификатам и ключам:

```
[mysqld]
ssl-ca=/path/to/ca-cert.pem
ssl-cert=/path/to/server-cert.pem
ssl-key=/path/to/server-key.pem
```

3. Настройка пользователей для работы через SSL. После того как SSL настроен, можно требовать от пользователей подключаться только через зашифрованное соединение:

```
CREATE USER 'ssl_user'@'%' IDENTIFIED BY 'password' REQUIRE SSL;
```

4. Проверка SSL-соединения. Чтобы убедиться, что соединение зашифровано, можно использовать команду:

```
SHOW VARIABLES LIKE 'have_ssl';
```

Также можно проверить информацию о текущем соединении:

```
SHOW STATUS LIKE 'Ssl_cipher';
```

6. Защита с помощью брандмауэров.

Брандмауэр — это система безопасности, которая контролирует и фильтрует сетевой трафик на основе заранее определённых правил.

В контексте защиты баз данных, брандмауэр играет важную роль в предотвращении несанкционированного доступа, атак по сети, таких как DDoS или попытки SQL-инъекций, и в ограничении круга лиц, которые могут подключаться к серверу MySQL.

Основные функции брандмауэра в защите базы данных:

- Фильтрация трафика — брандмауэр блокирует нежелательные или подозрительные запросы, которые не соответствуют установленным политикам безопасности.
- Контроль доступа — можно настроить доступ к серверу базы данных только для определённых IP-адресов или диапазонов адресов, ограничив доступ из внешних сетей.
- Минимизация атак — блокируя несанкционированные попытки подключения, брандмауэр помогает предотвратить атаки на уязвимости базы данных, такие как попытки SQL-инъекций или сканирование портов.

Настройка брандмауэров для ограничения доступа к базе данных MySQL

Настройка внешнего брандмауэра

На уровне операционной системы можно настроить брандмауэр для ограничения доступа к серверу MySQL. Например, в Linux можно использовать iptables или firewalld, чтобы разрешить доступ только с определённых IP-адресов или по нужным портам.

Пример настройки с помощью iptables для разрешения доступа только с одного IP-адреса:

```
iptables -A INPUT -p tcp -s <allowed_ip> --dport 3306 -j ACCEPT

iptables -A INPUT -p tcp --dport 3306 -j DROP
```

3306 — это стандартный порт для MySQL.

<allowed_ip> — IP-адрес, которому разрешён доступ.

Все остальные IP-адреса будут заблокированы от подключения к серверу MySQL.

Фильтрация IP-адресов и портов

Фильтрация IP-адресов

Брандмауэры позволяют ограничить доступ к серверу базы данных только с определённых IP-адресов. Это эффективно защищает сервер от атак извне, разрешая доступ лишь проверенным пользователям.

Пример фильтрации с использованием iptables:

```
iptables -A INPUT -p tcp -s 192.168.1.100 --dport 3306 -j ACCEPT
iptables -A INPUT -p tcp --dport 3306 -j DROP
```

В этом примере доступ к MySQL открыт только для IP 192.168.1.100, а остальные запросы на порт 3306 будут заблокированы.

Фильтрация портов

Стандартный порт для MySQL — 3306. Чтобы защитить базу данных, можно ограничить доступ к этому порту через брандмауэр, разрешив доступ только для определённых IP или приложений. Также можно изменить стандартный порт MySQL, чтобы усложнить задачу потенциальным злоумышленникам.

Пример фильтрации по порту:

```
iptables -A INPUT -p tcp --dport 3306 -j ACCEPT
```

Также можно изменить порт MySQL в файле конфигурации my.cnf для повышения безопасности:

```
[mysqld]
port=12345
```

Закрытие ненужных портов.

Кроме того, брандмауэры можно использовать для блокировки всех неиспользуемых портов, оставив открытыми только необходимые для работы. Это минимизирует риски атак, направленных на уязвимые порты.

7. Практические меры безопасности.

1. Использование сложных паролей.

Одной из ключевых мер защиты базы данных является правильная настройка паролей для пользователей и администраторов MySQL.

Сложные пароли затрудняют их подбор, а ограничение попыток входа защищает от атак грубой силы (brute-force).

2.Ограничение количества попыток входа.

Чтобы предотвратить атаки методом подбора паролей, MySQL поддерживает ограничение количества неудачных попыток входа. Это можно реализовать через плагины аутентификации, например, с использованием caching_sha2_password (начиная с MySQL 8.0).

Настройка блокировки пользователя после нескольких неудачных попыток:

Установите плагин validate_password:

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

Настройте параметры, такие как минимальная длина пароля и ограничения на количество попыток:

```
SET GLOBAL validate_password.length = 12;
SET GLOBAL validate_password.mixed_case_count = 1;
```

Включите блокировку после определённого количества неудачных попыток:

```
ALTER USER 'secure_user'@'localhost' FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 30;
```

Это правило блокирует пользователя после трёх неудачных попыток на 30 минут.

3.Регулярное обновление MySQL для устранения уязвимостей.

Как и любое программное обеспечение, MySQL может содержать уязвимости, которые устраняются с каждым новым релизом. Регулярное обновление серверного ПО помогает защитить систему от новых атак.

Пример обновления MySQL на Ubuntu:

```
sudo apt update
sudo apt upgrade mysql-server
```

4.**Настройка резервного копирования и восстановления данных**

Резервное копирование является важным элементом безопасности, так как помогает восстанавливать данные в случае сбоя, атаки или утраты данных. Настройка регулярного бэкапа базы данных снижает риски потери важной информации.

Домашнее задание:

1. Повторить материал лекции.

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com