SATBAYEV UNIVERSITY

Computing & Engineering

Volume 1 (2023), Issue 3, 12-20

https://doi.org/10.51301/ce.2023.i3.03

Analysis of mechanisms of optimization and security of database storage of the program for auditing passwords of user accounts of domain infrastructure users

E.A. Kovalenko, Ye.Ye. Begimbayeva*

Satbayev University, Almaty, Kazakhstan

**Corresponding author: Y.begimbayeva@satbayev.university*

Abstract. The analysis work conducted focuses on the importance of databases (DBs) in the context of modern information technology, emphasizing their key role in the efficient operation of applications. This paper analyzes different types of databases such as relational (MySQL, PostgreSQL, Oracle) and NoSQL (MongoDB, Cassandra) and emphasizes the need to choose the right type of database depending on the application requirements. The paper describes mechanisms and methods for optimizing and securing relational databases, focusing on MySQL/MariaDB in the context of a domain infrastructure account password auditing application under development, since the correct choice of database type depends on the application requirements and the nature of the data. The main mechanisms for optimizing relational database performance include indexing mechanisms to speed up queries, optimal database configuration to reduce data access time, cryptographic encryption of table contents to ensure security, analysis of cryptographic authentication protocols, and advanced monitoring techniques to detect anomalies. Application of cryptographic encryption such as SHA-1/2, DES, AES, and others ensures the security of table contents at the DBMS level. Comparing different cryptographic authentication protocols is also important to ensure data security. And also, modern monitoring and logging techniques can detect anomalies in database operation, including unauthorized access attempts. This study offers a more detailed analysis of these methods and mechanisms in order to optimize the performance and security of relational databases in the context of developing software for auditing domain infrastructure user account passwords.

Keywords: *relational databases, MySQL, MariaDB, optimization, security, cryptographic encryption, indexing*.

1. Введение

С появлением современных технологий и ростом сложности приложений в сфере информационных технологий становится ясным, что базы данных (БД) играют ключевую роль в обеспечении эффективной работы приложений. Базы данных представляют собой организованный набор данных, обеспечивающий хранение, управление и доступ к информации. Соответственно, оптимизация, безопасность и правильное использование БД становятся важными компонентами успешной разработки программного обеспечения, где объем данных постоянно растет.

Одним из важных аспектов использования баз данных является выбор подходящего типа БД для конкретного приложения. Такие реляционные базы данных как MySQL, PostgreSQL и Oracle, широко используются для приложений, требующих сложных связей между данными. С другой стороны, такие базы данных как MongoDB или Cassandra, именуемые NoSQL, дают возможность гибкого хранения и обработки больших объемов неструктурированных данных. Правильный выбор типа базы данных зависит от требований приложения, его структуры и характера данных.

Исходя из этого, в данной работе будут описаны и проанализированы различные механизмы и методы обеспечения оптимизации работы и безопасности реляционных баз данных, так как разрабатываемое ПО аудита паролей учетных записей пользователей доменной инфраструктуры среди своих основных компонентов имеет использование MySQL/MariaDB базы данных.

Такие механизмы и методы как:

- одним из основных механизмов оптимизации работы реляционных баз данных является индексирование. Индексы позволяют ускорить поиск и сортировку данных, снижая временные затраты на выполнение запросов.

- различные варианты оптимальной конфигурации БД позволяют уменьшить время доступа к данным, оптимизируя работу СУБД при взаимодействии с оперативной памятью хоста. Это особенно полезно при выполнении повторяющихся запросов, таких как отчеты или поисковые запросы.

- применение различных методов криптографического шифрования содержимого таблиц (Шифрование и хэширование на уровне СУБД – SHA-1/2, DES, AES, crypt() и другие).

- анализ и сравнение различных криптографических протоколов аутентификации.

- современные методы мониторинга и журналирования для выявления аномалий в работе базы данных, включая попытки несанкционированного доступа.

Рассмотрим и проанализируем данные методы и механизмы более подробно.

2. Механизмы оптимизации работы реляционных баз данных

В большинстве случае быстрота отклика самого приложения аудита паролей учетных записей пользователей доменной инфраструктуры зависит от скорости работы используемой базы данных. Отсюда следует, что на процессы исполнения основного скрипта влияет замедленная обработка запросов, что означает накапливание огромного количества операций, с которыми сервер не справляется.

Управление приведенным процессом обеспечивается через применение различных систем управления базами данных (СУБД). Наиболее популярной на сегодняшний день является MySQL - программное обеспечение с открытым исходным кодом (Open-source), созданное компанией Oracle (MySQL AB) в 1995 году. Также существует MariaDB - вариант реляционной СУБД MySQL, разрабатываемый сообществом под лицензией GPL. MariaDB совместима с приложениями, использующими MySQL, и переход на эту СУБД в разработке приложения по аудиту паролей учетных записей пользователей доменной инфраструктуры обоснован, поскольку по сравнению с MySQL она обладает более высокой производительностью, новыми возможностями и меньшим количеством ошибок в коде (в том числе за счёт более активного тестирования и развития) [1].

MariaDB включает в себя улучшенный оптимизатор запросов, безопасную и быструю репликацию, более быстрые индексы для механизма хранения данных MEMORY(HEAP), повышенную производительность перекодировки символов, использование пула потоков и множество других усовершенствований, положительно влияющих на производительность. Проведение операций по оптимизации СУБД позволит избежать проблем с производительностью сервера, существенно ускоряя и улучшая отклик самого приложения.

hash_check.diplome: 3 строк (приблизительно)

id	objectsid	username	samAccountName	email	hash	password	last_logon	pass_last_set
1	1	Джон До	John_Doe	johndoe@example.com	cbb5021c461bxxxxxxxxx1614a83880f	AaXXXX1	2022-12-20 03:51:16	2022-12-05 03:21:27
2	2	Anonymous	Anon_anon	anon@example.com	cbb5021c461bxxxxxxxxx1614a83880f	AaXXXX1	2022-12-20 03:51:16	2022-12-05 03:21:27
3	3	Emil Kov	Emil_Kov	emil@example.com	cbb5021c461bxxxxxxxxx1614a83880f	AaXXXX1	2022-12-20 03:51:16	2022-12-05 03:21:27

Рисунок 1. Пример заполненной таблицы пользователей с простыми паролями

Следующий раздел содержит различные методы улучшения производительности баз данных MySQL/MariaDB с использованием специального скрипта, а также представлены различные параметры настройки, которые будут рассмотрены далее.

2.1. Оптимизация взаимодействия базы данных с памятью хоста и ее реализация

Ядро MySQL имеет сложную конфигурацию, но оптимизация запросов не обязательно должна выполняться вручную. Для решения проблем с производительностью существует специальный скрипт с открытым исходным кодом, написанный на Perl - MySQLTuner. Он позволяет быстро оценить текущую конфигурацию MySQL и внести изменения для улучшения производительности и стабильности. Текущие настройки и статус извлекаются и представляются в кратком формате вместе с основными рекомендациями по улучшению производительности.

Затем следует провести операцию по тестированию базы данных. На основе выявленных проблем будет организована оптимизация. С помощью следующей команды запускается сам тест: perl ./mysqltuner.pl. Этот инструмент предоставляет исчерпывающую статистику функционирования базы данных. Все значимые проблемы отмечаются красным восклицательным знаком [2].

Для сравнительного анализа работы базы данных параметры будут изменены в соответствии с рекомендациями утилиты и требованиями разрабатываемого приложения для аудита паролей учетных записей пользователей доменной инфраструктуры. В противном случае могут возникнуть практические проблемы, такие как недостаточный размер буфера InnoDB.

Запустим скрипт на тестовом стенде с работающей СУБД MariaDB:

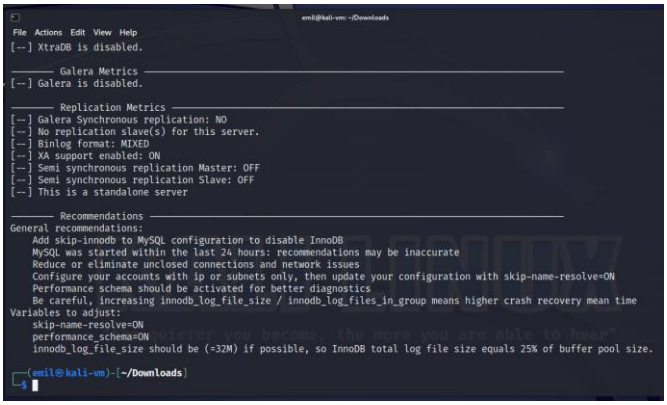

Рисунок 2. Тестовая проверка конфигурации СУБД

Чтобы настроить производительность вручную, требуется внести изменения в соответствующие параметры в файле конфигурации MySQL, расположенном по адресу /etc/mysql/conf.d/app.cnf. Эти изменения параметров направлены на более эффективное использование оперативной памяти сервера:

Таблица 1. Параметры конфигурации

Атрибут конфигурации (ключ)	**Определение**
tmp_table_size	Это максимальный объем оперативной памяти, отведенный для временных таблиц, создаваемых при выполнении SQL-запросов для извлечения данных из основной таблицы для их последующей обработки, требует экспериментальной настройки. Следует руководствоваться формулой: tmp_table_size = ОЗУ*0,75 (где tmp_table_size может быть равна 64 М).

max_heap_table_size	Это максимальный объем таблицы, который может хранится в оперативной памяти, должен соответствовать рекомендуемому значению параметра max_heap_table_size (max_heap_table_size = tmp_table_size.)
query_cache_size	Это размер оперативной памяти, выделенный под кэш SQL-запросов, рекомендуется отключить в проектах с обширными наборами данных, то есть установить query_cache_size = 0.
query_cache_type	Это параметр, который включает или выключает сервис управления кэшем в MySQL. Рекомендуется его выключить, установив значение query_cache_type = 0.
join_buffer_size	Рекомендуемый размер оперативной памяти, отведенный для объединения таблиц баз данных, составляет 8 Мб.
sort_buffer_size	Рекомендуемый объем оперативной памяти, предназначенный для сортировки данных, составляет 10 Мб.
max_connections	Это параметр, который устанавливает максимальное количество соединений с базой данных, начинает принимать значение после возникновения ошибки («Too many connections»). Увеличение этого значения рекомендуется постепенно до исчезновения ошибки.

После применения оптимизаций данной конфигурации следует повторно запустить проверку скриптом для анализа вывода и изменений в работе.

Исходя из анализов вывода повторного тестирования конфигурации, можно отметить, что большинство проблем связанных с начальной установкой СУБД было исправлено, а также оптимизированы показатели расхода ОЗУ, что положительно сказывается на работе самого приложения по аудиту паролей учетных записей.

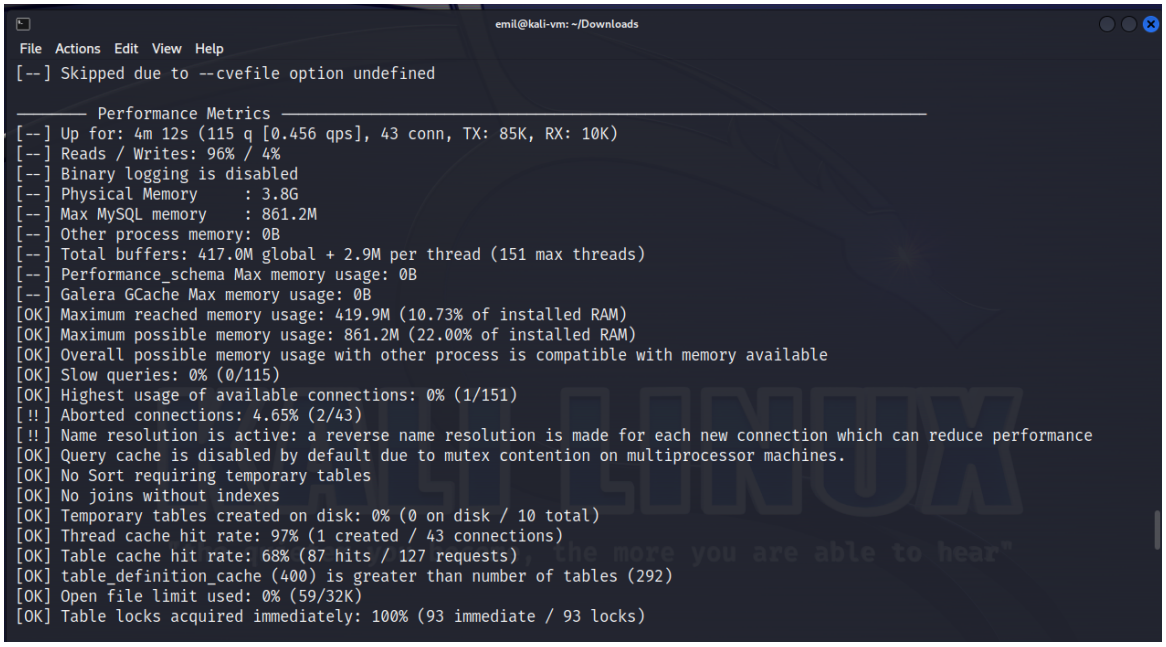

Рисунок 3. Повторное тестирование конфигурации

2.2. Механизмы оптимизаций выборки данных и их реализация

Скорость доступа к данным в базе данных является критическим фактором, влияющим на эффективность работы приложения. Первым шагом к оптимизации производительности базы данных является правильное создание индексов для таблиц. Индексы значительно ускоряют процесс доступа к данным, позволяя MySQL быстрее находить нужные записи. Тем не менее, необходимо осторожно использовать индексы, поскольку избыточное количество может отрицательно сказаться на производительности операций вставки или обновления данных.

Оптимизация SQL-запросов также играет важную роль в улучшении скорости доступа к данным MySQL. Рекомендуется избегать использования «SELECT *», поскольку это может привести к избыточному объему передаваемых данных и замедлению запросов. Вместо этого следует явно указывать необходимые поля таблицы. Кроме того, целесообразно применять индексы в SQL-запросах, где это возможно, для повышения эффективности выполнения запросов MySQL [3].

Дополнительно, рекомендуется использовать кэширование данных с целью повышения производительности приложения. Кэширование позволяет сохранять результаты запросов в памяти сервера и возвращать их при следующем обращении, если данные остаются неизменными. Это значительно сокращает время доступа к данным. Для реализации кэширования можно использовать различные инструменты, такие как Redis или Memcached.

В данном разделе будет проанализировано сравнение результатов применения метода индексирования таблиц.

Индексирование таблиц. Индексы представляют собой структуры данных, разработанные для оптимизации процесса поиска записей в таблицах с целью повышения производительности. В базах данных таблицы могут содержать большое количество строк, которые сохраняются в произвольном порядке, и поиск записей по определенному критерию путем последовательного сканирования строк может занимать значительное время. Индекс создается на основе значений одного или нескольких столбцов таблицы и содержит указатели на соответствующие строки, что упрощает поиск строк, удовлетворяющих заданному критерию.

Индексы - ключ к высокой производительности MySQL, их важность увеличивается по мере роста объема

данных в базе. Индексы нужно создавать для столбцов, по которым:

- производится поиск в части WHERE
- соединяются таблицы при JOIN
- сортируются и группируются записи при ORDER BY и GROUP BY
- производится поиск MIN() и MAX()

Индексы могут быть составными, в этом случае важен порядок столбцов.

Для анализа сравнения изменения скорости обработки запросов используем команду SELECT * FROM employee WHERE emp_id > 30 для получения первого значения скорости обработки запросов:

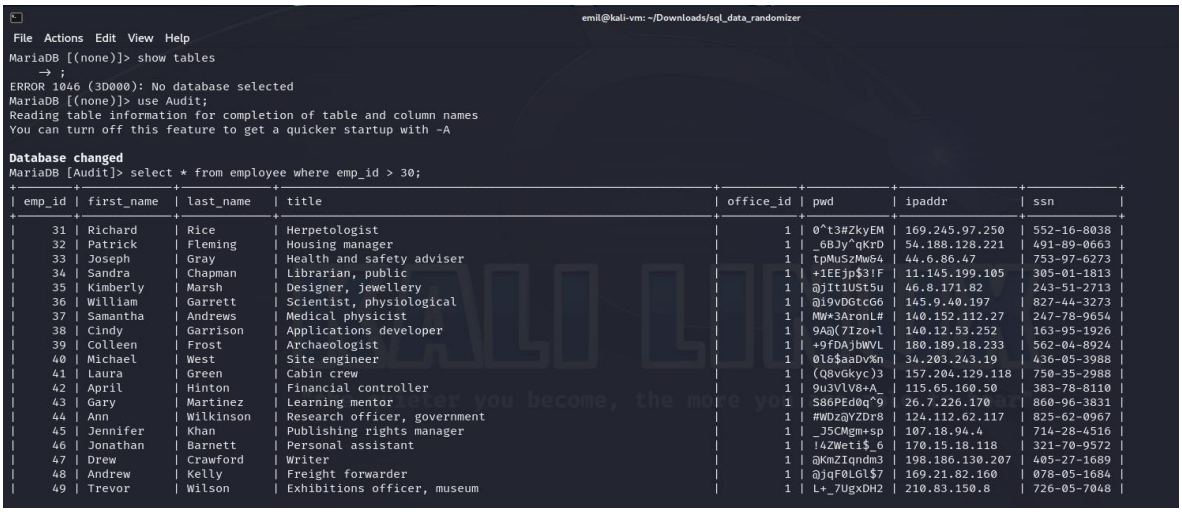

Рисунок 4. Запуск команды

Получаем значение обработанного запроса в 0.039 сек.:

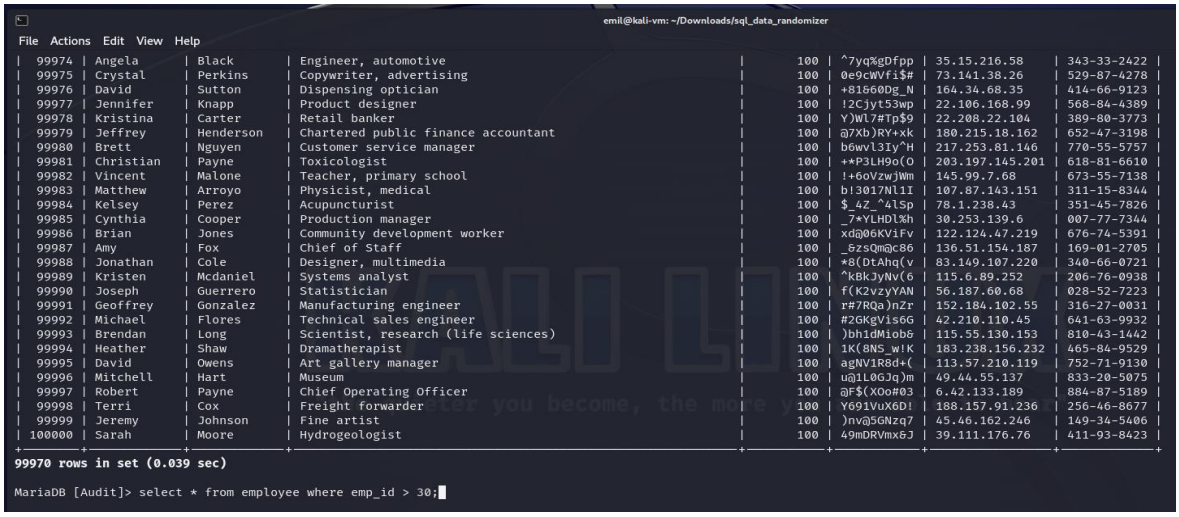

Рисунок 5. Результаты запущенной команды по времени

Далее создадим индекс на первичный ключ именуемый EMP_ID и снова запустим команду из предыдущего этапа:

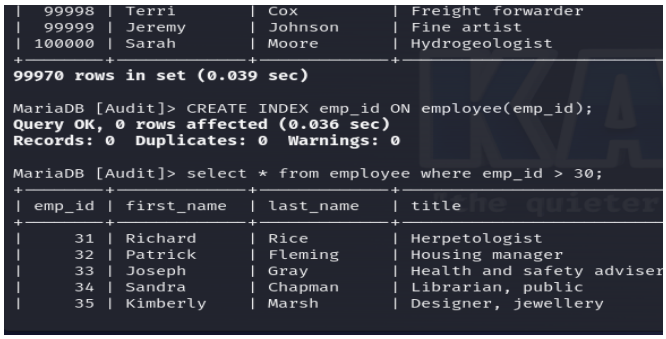

Рисунок 6. Создание индекса

По результатам выполнения команды с использованием индекса видно, что общее время работы сократилось до 0.031 секунды, что действительно оптимизирует дальнейшее выполнение подобных запросов внутри СУБД:

Рисунок 7. Результаты запроса с использованием индекса

Корректная настройка и оптимизация MySQL способны обеспечить высокую производительность сервера и приложений, использующих его. Для этого полезно использовать скрипты, которые могут быстро выявить проблемы, влияющие на работу базы данных [4].

Помимо автоматической оптимизации, проводимой с помощью скриптов, важно также ручное настройка основных параметров СУБД для улучшения производительности. Для повышения эффективности MySQL необходимо оптимизировать процесс выборки данных из нескольких объединенных таблиц, включая индексацию, кэширование запросов и проведение дефрагментации.

3. Результаты и обсуждение

3.1. Обеспечение безопасности данных в СУБД аудитора паролей

3.1.1 Криптографические методы шифрования данных содержимого таблиц

Базы данных (БД) представляют собой один из наиболее ценных активов для любого предприятия. В БД программы аудита паролей хранятся учетные записи пользователей доменной инфраструктуры, поэтому вопрос защиты данных требует особого внимания. Любые попытки несанкционированного доступа к СУБД и базе данных могут иметь серьезные последствия [5].

Следующие механизмы относятся к основным средствам защиты информации:

- защита паролей;
- защита содержимого БД: полей и записей таблиц;
- настройка БД касательно прав доступа к объектам;
- установка шифрования данных;
- настройка защиты учетных записей пользователей, а также идентифицированных объектов;

Также возможно ограничить доступ не только к самой системе управления базами данных, но и к конкретным базам данных, таблицам, записям в них, а также к значениям полей таблиц или записей. MySQL/MariaDB имеют два аналогичных набора функций шифрования: один использует алгоритм DES, а другой — AES. Помимо этого, MySQL предлагает несколько алгоритмов хэширования. СУБД предоставляет набор криптографических функций следующего вида:

Таблица 2. Криптографические функции СУБД

Название функции (ключ)	Назначение функции
AES_ENCRYPT()	Функция шифровки данных алгоритмом AES.
AES_DECRYPT()	Функция расшифровки данных алгоритмом AES.
DES_ENCRYPT()	Функция шифровки данных алгоритмом DES.
DES_DECRYPT()	Функция расшифровки данных алгоритмом DES.
ENCRYPT()	Функция шифровки данных функцией crypt().
MD5()	Функция хэширования данных алгоритмом MD5.
SHA1() или SHA2()	Функция хэширования данных алгоритмом SHA-1 или SHA-2.

Симметричные криптографические функции и SSL. Функции шифрования данных с использованием алгоритма AES в стандартной конфигурации применяют 128-битный ключ шифрования. В MariaDB версии 11.2 также доступна функция шифрования с ключами размером 192 и 256 бит в качестве дополнительной опции. При использовании этой функции ключ шифрования должен быть явно указан как один из параметров. В отличие от этого, функции DES_ENCRYPT() и DES_DECRYPT(), которые используют алгоритм TripleDES, могут также управляться с помощью ключевого файла, содержащего пронумерованные значения ключей, помимо явного указания ключа шифрования. Однако эти функции по умолчанию отключены, и для их использования необходимо включить поддержку протокола SSL в конфигурации СУБД [6].

SSL в MariaDB, начиная с версии 11.3, работает следующим образом: клиент подключается к серверу, происходит TLS handshake, сервер отправляет свой сертификат, который клиент проверяет. Если сертификат не подписан, он считается «сомнительным», однако, в отличие от предыдущих версий, соединение не разрывается сразу. Если сервер запросит пароль в открытом виде, соединение разорвется без обмена паролями. В другом случае, если используется двойной SHA1 или ed25519 для аутентификации и пароль верен, сервер отправит клиенту дополнительную подпись SHA2 («хэш пароля» + scramble + «fingerprint сертификата»). Возможно, «Man-in-the-middle» может знать scramble и fingerprint, но не знает хэш пароля и не сможет подделать подпись. Клиент знает все три части, может вычислить такое же значение и убедиться, что сервер действительно знает пароль, и сертификат не был подменен в процессе.

Хэширование данных и ее реализация в БД аудита паролей. Функция ENCRYPT() работает только в операционных системах из семейства Unix, так как она использует системный вызов crypt() для шифрования данных. Что касается функций хэширования, документация на MySQL/MariaDB предупреждает о том, что алгоритмы, используемые ими, были взломаны, и уязвимости были устранены в последних версиях СУБД. Поэтому важно регулярно обновлять и использовать актуальные версии СУБД с учетом всех обновлений безопасности. Упомянутые криптографические функции также просты в использовании. Например, следующий запрос добавляет в таблицу значение «text», зашифрованное с использованием ключа «password» - INSERT INTO table VALUES (1, AES_ENCRYPT('text', 'password'));

После применения данного запроса на тестовом стенде видно, что пароли при выводе не показываются пользователю в открытом виде, а представлены в виде хэша:

office_id	pwd	ipaddr
1	0^t3#ZkyEM	169.245.97.250
1	_6BJy^qKrD	54.188.128.221
1	tpMuSzMw&4	44.6.86.47
1	+1EEjp$3!F	11.145.199.105
1	@jIt1USt5u	46.8.171.82
1	@i9vDGtcG6	145.9.40.197
1	MW*3AronL#	140.152.112.27
1	9A@(7Izo+l	140.12.53.252
1	+9fDAjbWVL	180.189.18.233
1	0l&$aaDv%n	34.203.243.19
1	(Q8vGkyc)3	157.204.129.118
1	9u3VlV8+A_	115.65.160.50
1	S86PEd0q^9	26.7.226.170
1	#WDz@YZDr8	124.112.62.117
1	_J5CMgm+sp	107.18.94.4
1	!4ZWeti$_6	170.15.18.118
1	@KmZIqndm3	198.186.130.207
1	@jqF0LGl$7	169.21.82.160
1	L+_7UgxDH2	210.83.150.8
1	hmn@8yPk!I	20.177.46.94

Рисунок 8. Зашифрованное представление паролей

Важно учитывать, что формат поля, предназначенный для зашифрованного значения, должен соответствовать требованиям, установленным для используемого криптоалгоритма. Например, в данном случае это должно быть двоичное поле (например, типа VARBINARY), которое должно быть выровнено в соответствии с размером блока алгоритма AES, равным 128 битам [7].

3.1.2. Анализ протоколов аутентификации MySQL и MariaDB

Для проведения анализа уровней безопасности протоколов аутентификации СУБД MySQL/MariaDB требуется сравнить несколько наиболее популярных вариантов [8].

Первый протокол аутентификации mysql-3.20, 1996.

С самого начала истории MySQL пароли никогда не передавались в открытом виде. Вместо этого использовались случайные строки, основанные на хэшах. Это правило легло в основу первого протокола аутентификации mysql-3.20, который появился в 1996 году:

Сервер хранил хэш от пароля размером в 32 бита. Пример скрипта:

```
for (; *password ; password++)
{
  tmp1 = *password;
  hash ^= (((hash & 63) + tmp2) * tmp1) + (hash << 8);
  tmp2 += tmp1;
}
```

Рисунок 9. Пример генерируемого хэша

Во время процесса аутентификации, сервер отправлял клиенту случайную последовательность из восьми символов.

Клиент вычислял хэш этой последовательности и хэш пароля. Затем путем применения операции XOR к этим двум 32-битным числам инициализировал генератор случайных чисел, генерировал восемь «случайных» байт и отправлял их на сервер.

Сервер, тем временем, использовал хэш пароля (который он знал) и выполнял операцию XOR с хешем той случайной строки. Затем запускал генератор случайных чисел на своей стороне и сравнивал результат с тем, что прислал клиент.

Это решение имело следующие преимущества. Пароль никогда не передавался и не хранился в открытом виде. Однако 32 бита были слишком недостаточны для размера хэша. Поэтому в следующем основном релизе (mysql-3.21) размер хэша был увеличен до 64 бит. Этот протокол, известный как «Старая аутентификация MySQL», по-прежнему используется. В MySQL 5.7 он был удален, но в MariaDB он сохраняется до версии 10.4.

Протокол двойного SHA-1.

Основная сложность предыдущего механизма заключалась в том, что хоть в базе данных хранился хэш от пароля, а не сам пароль, однако даже в этом случае клиенту необходим был пароль — для аутентификации использовался именно хэш. Это означало, что злоумышленнику достаточно было получить таблицу mysql.user с хэшами паролей и после небольшой модификации клиентской библиотеки он мог подключиться как любой пользователь.

Для решения этой проблемы был разработан протокол «двойной-SHA1», который был внедрен в MySQL-4.1, а также все еще используется без изменений:

Сервер сохраняет значение SHA1(SHA1(password)).

Для проверки подлинности, сервер по-прежнему посылает клиенту случайную последовательность (20 символов), исторически называемую «scramble».

Затем клиент отправляет серверу строку, содержащую операцию XOR:

```
SHA1( scramble || SHA1( SHA1( password ) ) ) ⊕ SHA1( password )
```

Рисунок 10. Применение двойного SHA-1

Следовательно, сервер не имеет доступа к SHA1(password), однако он имеет информацию о значении «scramble» и SHA1(SHA1(password)), поэтому может вычислить первый операнд в этой конструкции, а затем, применяя операцию XOR, получить второй операнд, то есть SHA1(password). Затем, вычисляя SHA1 от полученного значения, сервер может сравнить его с тем, что хранится в таблице для данного пользователя.

Протокол оказался успешным в своей реализации, так как он достиг поставленных целей. Попытки перехватить аутентификацию и получить хэши паролей были бессмысленными. Однако появилась проблема, связанная с тем, что, если бы злоумышленнику удалось получить таблицу mysql.user с хэшами паролей и перехватить аутентификацию, тогда была возможность повторить действия сервера и восстановить SHA1(password), чтобы затем выдавать себя за соответствующего пользователя. Также по-прежнему существовала угроза перебора хэшей по словарю.

Протокол с использованием SHA256 и RSA.

Из-за ухудшения надежности алгоритма SHA1 с течением времени, компания Oracle решила внедрить криптографию с открытым ключом. Поэтому данный протокол в MySQL-5.7 пользуется SHA256 (256-битную версию SHA2) и RSA. Принцип работы данного протокола заключается в следующем:

Сервер хранит значение SHA256 (password).

Сервер, как и прежде, отправляет клиенту случайную последовательность из 20 символов, называемую «scramble».

Клиент извлекает открытый RSA-ключ сервера из предварительно подготовленного файла.

Клиент выполняет операцию XOR с паролем, полученным из «scramble» (если пароль длиннее, «scramble» повторяется циклически), затем шифрует его с использованием ключа сервера и отправляет обратно.

Сервер, в свою очередь, расшифровывает полученные данные с помощью своего секретного ключа, проводит обратную операцию XOR, извлекает пароль в исходном открытом виде, вычисляет для него значение SHA256 и сравнивает с хранимым значением.

Недостатком этой реализации является неудобство предварительного распределения серверного открытого ключа всем клиентам, особенно если имеется несколько серверов. Для решения этой проблемы в MySQL была внедрена возможность запроса клиентом открытого

ключа с сервера. Однако это представляет угрозу атакам Man-in-the-middle. В связи с опасениями со стороны финансового сектора было принято решение искать альтернативу данному протоколу.

Протокол на основе ed25519.

Новый протокол был разработан на основании ed25519 (криптографическая подпись с применением эллиптической кривой, предложенная Даниэлем Дж. Бернштейном — автором нескольких готовых к использованию реализаций, включая популярный OpenSSH). Принцип работы ed25519 заключается в следующем:

Генерируется последовательность из 32 случайных байт — это будет новый секретный ключ.

Из этих байт вычисляется значение SHA512, а затем математическим методом создается открытый ключ.

Далее текст подписывается с использованием секретного ключа. Подпись можно проверить с помощью открытого ключа.

Новый протокол аутентификации в MariaDB работает следующим образом:

Вместо случайной последовательности из 32 байт используется пароль пользователя (таким образом, пароль фактически является секретным ключом), после чего применяется SHA512 и вычисляется открытый ключ.

На сервере в качестве пароля в таблице mysql.user хранится открытый ключ (представленный 43 байтами в формате base64).

Для аутентификации сервер отправляет клиенту случайную последовательность из 32 байт.

Клиент подписывает эту последовательность.

Сервер проводит операцию по проверке подписи.

Недостатки предыдущих протоколов были устранены - пароль не хранится на сервере, не передается и не виден серверу, что исключает возможность его восстановления. Возможность атак Man-in-the-middle была устранена.

Этот протокол начал использоваться в MariaDB-10.1.22 в качестве плагина и был интегрирован в сервер в версиях 10.2-10.3.

3.1. Дальнейшая поддержка оптимальных процессов работы базы данных

3.1.1. Мониторинг и журналирование

Регулярный анализ выполнения запросов поможет выявить узкие места и оптимизировать их. Необходимо использовать инструменты мониторинга, такие как EXPLAIN, для анализа и улучшения запросов. Улучшение производительности базы данных значительно повысит эффективность ее работы [9].

Журналы играют важную роль в мониторинге состояния сервера. В случае атак они позволяют легко обнаружить любые действия, связанные с вторжением, в журнальных файлах. Чтобы включить ведение журнала MySQL, необходимо добавить соответствующую переменную в раздел [mysqld]: log=/var/log/mysql.log.

Также в MySQL имеется Performance Schema, которая представляет собой функцию для мониторинга действий MySQL на низком уровне. При включении (что является настройкой по умолчанию) она постепенно динамически выделяет память, масштабируя использование памяти до фактической нагрузки сервера, вместо того чтобы выделять требуемую память во время запуска сервера. После выделения памяти она не освобождается до перезапуска сервера. Именно поэтому Performance Schema постоянно занимает все большие объемы памяти без ее возврата, что может отразиться на производительности сервера.

4. Выводы

Данная исследовательская работа посвящается анализу механизмов оптимизации и безопасности хранения базы данных программы аудита паролей учетных записей пользователей доменной инфраструктуры.

Были исследованы различные стандартные и сторонние методы и механизмы оптимизации работ реляционных баз данных и СУБД на их основе, в частности MySQL и Maria/DB. Также были рассмотрены и проанализированы возможности обеспечения безопасности данных в СУБД с использованием криптографии. Были рассмотрены наиболее предпочтительные варианты дальнейшей поддержки оптимальных процессов работы баз данных, в частности использование мониторинга и журналирования.

В ходе анализа процессов оптимизации БД было выявлено два наиболее результативных метода в случае программы аудита паролей – оптимизация взаимодействия базы данных с памятью хоста на основе тюнинга конфигурации СУБД и сравнения полученных показателей до/после, а также обзор механизмов оптимизации выборки данных с применением индексирования наиболее часто используемых данных в таблицах. Вследствие чего было получено преимущество во времени исполнения запросов.

Касательно обеспечения безопасности данных в СУБД, были проанализированы и сравнены различные криптографические методы шифрования данных содержимого таблиц, рассмотрены функции хэширования и применены на тестовом стенде. Далее были проанализированы протоколы аутентификации MySQL и MariaDB, начиная с самых первых версий по актуальные механизмы на сегодняшний день с подробным объяснением алгоритмов работы и недочетов каждого.

Рассмотренные методы и механизмы оптимизации и обеспечения безопасности были реализованы на тестовом стенде базы данных программы аудита паролей учетных записей пользователей доменной инфраструктуры. Результаты были прикреплены в виде скриншотов с наглядным сравнением результатов до/после.

В заключение хочется отметить, что оптимизация и безопасность реляционных баз данных представляют собой критически важные аспекты современного информационного управления. Они способствуют повышению производительности, обеспечивают целостность и безопасность данных, соблюдают комплаенс-контроль и снижают издержки. Организации, стремящиеся к успешной и устойчивой работе в цифровой эре, не могут проигнорировать необходимость внимания к оптимизации и безопасности реляционных баз данных.

References / Литература

[1] Kovalenko E., Begimbayeva E. (2023). Development of a program for auditing and storing passwords in encrypted form. *Satbayev Conference-2023: Science and Technology: From Idea to Implementation, (4), 96-100.*

[2] EternalHost. (2019). Optimizacija proizvoditel'nosti MySQL servera. *Retrieved from*: https://eternalhost.net/blog/hosting/optimizatsiya-mysql
[3] Szhatie, defragmentacija i optimizacija bazy dannyh MariaDB/MySQL. (2020). *Retrieved from*: https://ip-calculator.ru/blog/ask/szhatie-defragmentatsiya-i-optimizatsiya-bazy-dannyh-mariadb-mysql/
[4] Kak uskorit' vyborku dannyh v MySQL. (2023). *Retrieved from*: https://uchet-jkh.ru/i/kak-uskorit-vyborku-dannyx-v-mysql/
[5] Demin E., Sergeev D. (2013). Proizvoditel'nost' MySQL. Chast' 1. Analiz i optimizacija zaprosov. *Retrieved from*: https://hosting101.ru/articles/mysql-performance-1.html
[6] Dudkina, A.S. (2017). Metody zashhity i bezopasnost' bazy dannyh. *Retrieved from*: https://novainfo.ru/article/13078
[7] Petrov, P. (2022). Jevoljucija protokolov autentifikacii MySQL i MariaDB. *Retrieved from*: https://habr.com/ru/articles/323670/
[8] Petrov P. (2023). SSL dlja MariaDB. *Retrieved from*: https://habr.com/ru/articles/762628/
[9] Official manual, MariaDB. Encryption, Hashing and Compression Functions. *Retrieved from*: https://mariadb.com/kb/en/encryption-hashing-and-compression-functions/
[10] Official manual, MySQL. Encryption and Compression Functions. *Retrieved from*: https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html

Домендік инфрақұрылым пайдаланушыларының құпия сөздеріне аудит жасайтын бағдарламасының деректер базасын оңтайландыру және сақтау қауіпсіздігі механизмдерін талдау

Э.А. Коваленко, Е.Е. Бегимбаева*
Satbayev University, Алматы, Қазақстан
**Корреспонденция үшін автор: Y.begimbayeva@satbayev.university*

Аңдатпа. Талдау бойынша жүргізілген жұмыс қазіргі заманғы ақпараттық технологиялар контекстіндегі мәліметтер базасының (ДБ) маңыздылығына назар аударады, олардың қосымшалардың тиімді жұмысындағы шешуші рөлін атап көрсетеді. Бұл жұмыс реляциялық (MySQL, PostgreSQL ,acacle) және NoSQL (MongoDB, Cassandra) сияқты мәліметтер базасының әртүрлі түрлерін талдайды және қосымшаның талаптарына байланысты мәліметтер базасының түрін дұрыс таңдау қажеттілігін көрсетеді. Жұмыста MySQL/MariaDB-ге назар аудара отырып, реляциялық мәліметтер базасын оңтайландыру мен қауіпсіздікті қамтамасыз етудің тетіктері мен әдістері домендік Инфрақұрылым шоттарының құпия сөз аудиті бойынша әзірленіп жатқан қосымшаның контекстінде сипатталған, өйткені мәліметтер базасының түрін дұрыс таңдау қосымшаның талаптары мен деректердің сипатына байланысты. Реляциялық мәліметтер базасының жұмысын оңтайландырудың негізгі механизмдеріне сұраныстарды жеделдету үшін индекстеу механизмдері, деректерге қол жеткізу уақытын қысқарту үшін оңтайлы мәліметтер базасының конфигурациясы, қауіпсіздікті қамтамасыз ету үшін кесте мазмұнын криптографиялық шифрлау, криптографиялық аутентификация хаттамаларын талдау және ауытқуларды анықтаудың заманауи бақылау әдістері кіреді. SHA-1/2, DES, AES және басқалары сияқты криптографиялық шифрлауды қолдану ДҚБЖ деңгейінде кесте мазмұнының қауіпсіздігін қамтамасыз етеді. Деректердің қауіпсіздігін қамтамасыз ету үшін әртүрлі криптографиялық аутентификация хаттамаларын салыстыру да маңызды. Сондай-ақ мониторинг пен журналдаудың заманауи әдістері ДБ жұмысындағы ауытқуларды, соның ішінде рұқсатсыз қол жеткізу әрекеттерін анықтауға мүмкіндік береді. Бұл зерттеу домендік инфрақұрылымды пайдаланушылардың есептік жазбаларының парольдерін тексеруге арналған бағдарламалық жасақтаманы әзірлеу контекстінде жұмысты оңтайландыру және реляциялық мәліметтер базасының қауіпсіздігін қамтамасыз ету мақсатында осы әдістер мен механизмдерге егжей-тегжейлі талдау ұсынады.

Негізгі сөздер: *реляциялық мәліметтер базасы, MySQL, MariaDB, оңтайландыру, қауіпсіздік, криптографиялық шифрлау, индекстеу.*

Анализ механизмов оптимизации и безопасности хранения базы данных программы аудита паролей учетных записей пользователей доменной инфраструктуры

Э.А. Коваленко, Е.Е. Бегимбаева*
Satbayev University, Алматы, Казахстан
**Автор для корреспонденции: Y.begimbayeva@satbayev.university*

Аннотация. Проведенная работа по анализу фокусируется на важности баз данных (БД) в контексте современных информационных технологий, подчеркивая их ключевую роль в эффективной работе приложений. В данной работе анализируются различные типы БД, такие как реляционные (MySQL, PostgreSQL, Oracle) и NoSQL (MongoDB,

Cassandra), и подчеркивается необходимость правильного выбора типа БД в зависимости от требований приложения. В работе описываются механизмы и методы оптимизации и обеспечения безопасности реляционных БД, сосредотачиваясь на MySQL/MariaDB в контексте разрабатываемого приложения по аудиту паролей учетных записей доменной инфраструктуры, так как правильный выбор типа БД зависит от требований приложения и характера данных. Основные механизмы оптимизации работы реляционных БД включают в себя механизмы индексирования для ускорения запросов, оптимальную конфигурацию БД для сокращения времени доступа к данным, криптографическое шифрование содержимого таблиц для обеспечения безопасности, анализ криптографических протоколов аутентификации и современные методы мониторинга для выявления аномалий. Применение криптографического шифрования, такого как SHA-1/2, DES, AES, и других, обеспечивает безопасность содержимого таблиц на уровне СУБД. Сравнение различных криптографических протоколов аутентификации также важно для обеспечения безопасности данных. А также современные методы мониторинга и журналирования позволяют выявлять аномалии в работе БД, включая попытки несанкционированного доступа. Данное исследование предлагает более детальный анализ этих методов и механизмов с целью оптимизации работы и обеспечения безопасности реляционных БД в контексте разработки программного обеспечения для аудита паролей учетных записей пользователей доменной инфраструктуры.

Ключевые слова: *реляционные базы данных, MySQL, MariaDB, оптимизация, безопасность, криптографическое шифрование, индексация.*

Received: 14 April 2023
Accepted: 15 September 2023
Available online: 30 September 2023