**Безопасность баз данных на примере Oracle**

Не секрет, что, в наше время миром правит информация. Если предприятие дорожит своей интеллектуальной собственностью, если каждый работник может легко получить необходимую (и не более того) информацию, то предприятие может надеться на рост. Если данные находятся в хаосе… несмотря на энтузиазм сотрудников, в большинстве случаев предприятие ожидает крах.

Сколько раз мы слышали, о том, что база данных какого-то предприятия или государственной структуры появилось на прилавках магазина. Пятнадцать минут назад я ехал в метро на работу, и мне в открытую предлагали купить последние базы данных телефонов любых операторов, налоговые базы данных и кучу другой полезной и бесполезной информации. Это говорит о том, что компании, которые должны следить за безопасностью своих данных относятся к своему делу несерьезно или просто наплевательски.

Сегодня мы рассмотрим основы безопасности баз данных и немного поговорим о примерах защиты информации в Oracle (правда, теоретические основы защиты информации в БД, которые мы рассмотрим в этой статье, будут полезны и людям, работающим с другими базами данных).

**Основа прав доступа**

В большинстве организаций, где я работал или с которыми мне приходилось сталкиваться по работе, все администраторы и программисты имели полный доступ к базам данных, а любой сотрудник отдела ИТ являлся богом в сети и мог делать все, что угодно. Почему так происходит? На это есть две причины:

1. Работая в одном отделе или департаменте, сотрудники видят друг друга каждый день по 8 часов, завязываются дружеские отношения. Как же не дать другу полный доступ? Дружба дружбой, а работа есть работа.

2. Распределение некоторых прав доступа и изменения какой-то конфигурации может потребовать определенных привилегий. Администраторы иногда ленятся или просто считают, что программисты выполнят настройку лучше, поэтому дают программистам излишние права доступа. Никогда программисты не должны заниматься администрированием и для этого у них не должно быть прав.

На моем опыте вторая проблема встречается очень часто, поэтому мы рассмотрим ее более подробно. Разрабатывая программы для баз данных, программисты знают пароль супер пользователя или просто имеют права администратора базы данных. Это излишне и абсолютно небезопасно. Полные права должен иметь только администратор базы и больше никто. Даже начальник отдела, директор и лучшие друзья могут обойтись меньшими привилегиями. Например, для разработки программ достаточно иметь права владельца схемы в базе данных Oracle, где находятся рабочие таблицы. Этого достаточно для создания новых таблиц, пакетов, индексов и любых других объектов, а также, для распределения прав доступа к объектам схемы другим пользователям. Права системы для полноценной работы абсолютно не нужны.

Если в вашем штате нет администратора баз данных, то это очень плохо. Лучше, когда за базу данных, ее производительность, оптимизацию и безопасность отвечает отдельный сотрудник. В крайнем случае, необходимо выделить одного программиста, который будет ответственным, и только он будет иметь исключительные права.

Судя по статистике, потери данных чаще всего происходят изнутри компании, т.е. ее сотрудниками. Странно, но не смотря на это, большинство продолжают игнорировать эту угрозу, а различные базы данных продолжают появляться на дисках в свободном доступе даже в метрополитене.

**Пользователи и роли**

Для распределения прав доступа в базах данных используется два типа объектов, которым можно распределять права – пользователи и роли. Роли – это как группы, которые объединяют пользователей, которые должны иметь схожие права. Например, всем бухгалтерам может потребоваться доступ к финансовым документам. Чтобы каждому бухгалтеру в отдельности не выдавать права, их можно объединить в роль, а потом этой роли дать необходимый доступ.

Как видите, принцип работы ролей схож с группами в ОС Windows. Схож, но имеет и отличия. Недаром, в базе данных могут быть реализованы все три типа объектов – пользователи, группы и роли, но в большинстве баз реализуют только два – первый и последний. Всеми этими объектами необходимо управлять и отслеживать, чтобы каждый пользователь, наследуя права, предоставляемые ролями, имел доступ только к тем данным, которые действительно необходимы для решения поставленных задач. К правам доступа нужно относиться как к правилам сетевого экрана, потому что с их помощью можно решить одну и ту же задачу – позволить пользователю выполнять только определенные действия и предотвратить возможную потерю или разрушение.

С помощью ролей, достаточно удобно управлять доступом, выделять разрешения или запрет целой группе пользователей. Одна роль может входить в другую, то есть наследовать ее права доступа. Таким образом, можно создать иерархическую структуру прав.

Все сотрудники, которые должны иметь доступ к корпоративной базе данных могут быть включены в одну роль, которой будут предоставлены минимальные права. Теперь, если необходимы дополнительные привилегии, доступ к определенным таблицам, то для этого можно добавлять пользователя в другие роли (если такой же доступ нужен группе) или выделять права конкретной учетной записи.

В программе, для контроля доступа к таблицам, можно после каждой попытки открытия набора данных проверять результат на наличие ошибки. Если произошла ошибка доступа, то пользователю запрещено обращаться к указанной таблице. Таким образом, нет необходимости реализовывать права доступа к клиентском приложении. Но если вы захотите это реализовать, хуже от этого не будет. По крайней мере я, ничего критического в этом не вижу, а даже наоборот.

**Аудит безопасности**

Если в вашей базе данных каждый пользователь работает под своей учетной записью, то для определения текущего пользователя можно обратиться к переменной user, например, так:

SELECT user from dual

Этот запрос вернет имя пользователя, под которым открыто соединение с сервером. Если под одним именем может войти несколько человек, то этот запрос уже для обоих вернет одно и тоже имя и для аудита такое положение дел ничего не даст. Особенно, если управление блокировками будет происходить на уровне сервера.

Если под одним именем может войти несколько человек, то идентифицировать, кто именно вощел все еще можно, но уже сложнее. Следующий запрос, позволяет получить более детальную информацию о сессии:

select s.user#, s.username, s.osuser, s.terminal, s.program

from sys.v\_$session s

WHERE username=user

Запрос вернул имя пользователя, подключенного к базе, имя в ОС, имя терминала (компьютера) и программу

Здесь происходит обращение к представлению v\_$session из системной схемы sys. Это представление возвращает информацию о соединениях с сервером. В секции WHERE мы ограничиваем вывод только текущим пользователем. В качестве результата запрос вернет идентификатор пользователя, имя, имя в ОС, терминал и программу, из которой создано подключение.

Теперь, зная имя и компьютер, пользователь идентифицируется однозначно. Чтобы узнать, в какие роли входит текущий пользователь, можно выполнить следующий запрос:

SELECT GRANTED\_ROLE

FROM dba\_role\_privs

WHERE grantee=user

**Безопасность учетных записей**

Нет, мы не будем говорить о том, что не должно быть паролей по умолчанию. Некоторые базы данных (например, MySQL) грешат тем, что после установки устанавливают простой и общеизвестный системный пароль. Мы также не будем говорить о том, что пароль должен быть сложным. Это правило относиться ко всем ИТ областям и должно быть известно даже начинающему пользователю. Мы поговорим о том, что касается именно баз данных.

Как показывает мой опыт, очень часто при разработке корпоративных программ, программисты используют одну учетную запись для доступа к базе данных. Параметры этой учетной записи прописываются прямо в программе, а доступ к определенным модулям (бухгалтерия, склад, отдел кадров и т.д.) реализовывается программно. С одной стороны, этот способ удобен, потому что программа может подключаться к базе с администраторскими правами и не придется мучаться с ролями и распределением прав на таблицы. А с другой стороны, этот способ далеко не безопасен. Уровень пользователей в нашей стране растет, да и знакомые у сотрудников вашей компании могут быть достаточно просвещенными в сфере безопасности и взлома. Узнав имя и пароль, под которыми программа соединяется с базой, простой пользователь сможет получить больше возможностей, чем вы этого хотели.

Язык SQL не является секретом и слишком уж сложным языком. Существует множество программ, с помощью которых, можно легко просмотреть любые данные в базе. Зная имя и пароль, кто угодно может украсть всю информацию и через пару дней она будет в любом ларьке, торгующем дисками.

Имя пользователя и пароль никогда не должны сохраняться в программе. Доступ в саму программу также должен быть ограниченным и невозможным для стороннего человека. Вполне логично будет объединить обе авторизации в одну. Каждому пользователю в организации необходимо завести отдельную учетную запись на сервере баз данных с необходимым минимумом прав и именно эти имя/пароль должны использоваться при авторизации в программе. Можно использовать распространенную и очень эффективную логику - если с помощью введенных данных программа смогла авторизоваться в базе данных, то доступ разрешен, если нет, то необходимо прервать выполнение программы. Просто и эффективно, потому что мы задействовали авторизацию базы данных.

Чтобы проконтролировать, что под одним именем одновременно не работают с двух разных компьютеров, можно выполнить следующий запрос:

select s.username, s.terminal, count(\*)

from sys.v\_$session s

WHERE username=user

HAVING count(\*)>1

GROUP BY s.username, s.terminal

В идеале, он не должен вернуть ни одной записи.

**Представления**

Представления или View являются очень удобным способом абстрагироваться от структуры данных и в то же время построить дополнительный уровень защиты. Да, представления отрицательно сказываются на производительности, но положительно сказываются на безопасности. На них можно выдавать собственные права доступа, при этом, сами таблицы, из которых выбираются данные, могут оставаться недоступными для данной учетной записи.

Когда лучше использовать представления, а когда нет? Для начала определимся, в чем кроется недостаток. Представление – это запрос SELECT на выборку данных. Он может обращаться как к одной, так и к нескольким таблицам. Если просто выбрать данные из представления, то падения производительности не будет. Но мы можем использовать представления в других запросах на выборку данных, и обращаться к ним, как к таблицам. Например:

SELECT список полей

FROM представление, таблица1, таблица2

WHERE какие-то параметры

В данном случае, запрос выбирает данные из представления и двух таблиц. При этом, наводиться связь между всеми объектами и может быть установлено дополнительное условие. А теперь посмотрим на этот запрос так, как видит его оптимизатор базы данных:

SELECT список полей

FROM (SELECT ...), таблица1, таблица2

WHERE какие-то параметры

В данном случае, я заменил представление на абстрактный запрос SELECT (он указан в скобках) из которого и состоит view. Получается, что сервер видит запрос с подзапросом. Если подзапрос возвращает динамические данные, а не статическое значение, то такая выборка данных будет работать медленнее, чем если бы мы написали то же самое, но избавились бы от подзапроса.

Если у вас нет глубоких вложения запросов (запрос-представление1-представление2...), то можно смело использовать view, без опаски потерять слишком много тактов в производительности. Потери минимальны, зато вы получаете лишний уровень безопасности, на который можно распределять права доступа.

**Безопасность данных**

Никогда не стоит выдавать полный доступ на объекты без особой необходимости. Чтобы не выдать лишнего, я всегда начинаю распределение прав с разрешения только выборки данных, т.е. выполнения запроса SELECT. Если пользователю действительно необходима вставка новых записей и он не может выполнять поставленные перед ним задачи, то в этом случае добавляем разрешение INSERT на определенную таблицу.

Для данных наиболее опасными являются операции изменения и удаления, т.е. UPDATE и DELETE соответственно. К распределению этих прав нужно подходить еще более тщательно. Убедитесь, что данные действительно могут изменяться или удаляться и только в этом случае выделяйте соответствующие права. Некоторые таблицы по своей сути должны только пополняться.

Необходимо также убедиться, что данные будут подвергаться воздействию достаточно часто. Например, таблица сотрудников организации может пополняться и изменяться, но никогда и ни одна запись не должна удаляться. Удаление может повлиять на историю работы сотрудников в организации, на отчетность и целостность данных. Да, возможен вариант, когда сотрудник отдела кадров случайно завел лишнюю запись и хочет ее удалить, но такие случаи происходят редко и исправление ошибки можно возложить на администратора базы данных. Мы понимаем, что никому не хочется выполнять лишнюю работу и проще выдать разрешение, но безопасность дороже.

Предоставление прав происходит с помощью оператора GRANT. В общем виде он выглядит следующим образом:

GRANT на что давать права ON объекты TO пользователи или роли

Например, следующий запрос предоставляет право на вставку и просмотр таблицы TestTable пользователю User1:

GRANT Select, Insert ON TestTable TO User1

Какие роли уже выданы пользователю, можно легко увидеть с помощью SQL Navigator. В версии 4.4 для этого в дереве объектов выбираем раздел Users/Имя пользователя. Здесь вы увидите раздел Object Privileges, в котором и находятся все разрешенные пользователю действия.

**Внешние Ключи**

Многие бояться внешних ключей, потому что их наличие очень часто запрещает удаление данных, когда есть внешнее соединение. Проблема легко решается, если установить каскадное удаление, но большинство специалистов резко отрицательно относятся к этой возможности. Одно нелепое действие может уничтожить очень важные данные, без каких либо запросов на подтверждение. Связанные данные должны удаляться явно, и только если пользователь осознано подтвердил, что данные могут удаляться.

Не стоит бояться внешних ключей. Они предоставляют нам удобный способ контроля целостности данных со стороны сервера баз данных, т.е. это то же безопасность, а безопасность лишней не бывает. То, что могут возникнуть проблемы с удалением, то это только плюс. Лучше лишний раз не удалить данные, чем потерять их на веки. Внешний ключ, если он используется совместно с индексом, практически не уменьшает производительности. Это всего лишь небольшая проверка во время удаления данных или изменения ключевого поля, по которому происходит связь данных в разных таблицах.

**Резервное копирование**

Резервное копирование тоже один из факторов защиты, которым мы пренебрегаем до первой потери данных, но я бы отнес его к основным. Потеря данных может быть не только от хакеров и неумелых действий пользователей, но и из-за неисправностей в оборудовании. Сбои в оборудовании могут привести к потери базы данных, на восстановление которой могут уйти часы, а может даже и дни.

Необходимо заранее разработать наиболее эффективную политику резервного копирования. Что под этим понимается? Простои в работе, после потери данных и до момента восстановления работоспособности должны быть минимальны. Минимальны должны быть и потери данных, а само резервирование не должно мешать работе пользователей. Если позволяют финансы и возможности, то лучше использовать такие системы как RAID, кластер или даже репликацию данных.

Резервное копирование и восстановление – это отдельная и очень интересная тема. Не затронуть ее здесь мы не могли, но и рассматривать полностью не будем.

**Итого**

Мы рассмотрели основы безопасности, с небольшим уклоном в сторону Oracle. Это только начало и в будущем мы вернемся к этой теме, чтобы рассмотреть ее глубже. Но не стоит забывать, что средства защиты, предоставляемые базой данных – это только один уровень, а защита должна быть многоуровневой. Чтобы хоть немного спать спокойно, необходимо реализовать в сети, серверах и на всех компьютерах весь комплекс безопасности, а именно – антивирусы, антишпионы, сетевые экраны VPN, IDS и т.д. Чем больше уровней защиты, тем сложнее будет их преодолеть.

Должна быть четкая политика безопасности и контроль. Если какой-либо сотрудник увольняется, то его учетная запись удаляется. Если же у вас пользователи работают под одним паролем или для выполнения каких либо действий используют групповой пароль, то все эти пароли должны измениться. Например, если увольняется администратор, а у вас все администраторы используют одну учетную запись, то обязательно поменяйте пароль.

Безопасность необходима и не только от хакеров, но и от "начинающих" пользователей, которые могут разрушить данные по своей неопытности. Хорошая политика безопасности позволяет предотвратить и такие случаи и такую возможность нельзя исключать. Лучше заранее предусмотреть возможность обезопасить данные от неопытности, чем потом потерять уйму времени на восстановление данных и получить излишние простои в работе.