pgAudit
Расширение для аудита PostgreSQL с открытым исходным кодом

Введение

Расширение для аудита PostgreSQL (pgAudit) обеспечивает детальное журналирование сессий и объектов для целей аудита, используя стандартные средства журналирования PostgreSQL.

Цель pgAudit - предоставить пользователям PostgreSQL возможность создавать журналы аудита, которые часто требуются для соответствия государственным, финансовым или ISO-сертификациям.

Аудит - это официальная проверка счетов физического лица или организации, как правило, независимым органом. Информация, собранная pgAudit, называется контрольным журналом или журналом аудита. В данной документации используется термин “журнал аудита”.

Почему pgAudit?

Базовое журналирование операций может быть обеспечено стандартным средством ведения журнала с опцией `log\_statement = all`. Это приемлемо для мониторинга и других способов использования, но не обеспечивает уровень детализации, обычно необходимый для аудита. Недостаточно иметь список всех запросов, выполняемых к базе данных. Также должна быть возможность найти конкретные запросы, представляющие интерес для аудитора. Стандартное средство ведения журнала показывает, что запросил пользователь, в то время как pgAudit фокусируется на деталях того, что произошло, пока база данных удовлетворяла запрос.

Например, аудитор может захотеть проверить, что определенная таблица была создана внутри документально зафиксированного окна обслуживания. На первый взгляд, эта задача с легкостью решается с помощью утилиты grep, но что делать, если перед вами что-то вроде этого (намеренно запутанного) примера:

```
DO $$
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;
```

Стандартное журналирование выдаст следующее:

```
LOG:  statement: DO $$
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;
```

Похоже, что для поиска интересующей таблицы могут потребоваться некоторые навыки программирования в тех случаях, когда таблицы создаются динамически. Было бы предпочтительнее просто искать по имени таблицы. В таких случаях будет полезен

pgAudit. При тех же входных данных он будет производить следующий вывод в журнале:

```
AUDIT: SESSION,33,1,FUNCTION,DO,,,"DO $$
BEGIN
    EXECUTE 'CREATE TABLE import' || 'ant_table (id INT)';
END $$;"
AUDIT: SESSION,33,2,DDL,CREATE TABLE,TABLE,public.important_table,CREATE TABLE
important_table (id INT)
```

Здесь регистрируется не только блок DO, но и подзапрос 2 с полным текстом CREATE TABLE с типом запроса, типом объекта и полным именем, что существенно облегчает поиск.

При регистрации запросов SELECT и DML pgAudit может регистрировать отдельные записи для каждого отношения, на которое ссылается запрос. Можно найти все запросы, которые касаются определенной таблицы, без выполнения синтаксического анализа. В идеале текст запросов должен предоставляться в первую очередь для глубокого детального анализа и не должен быть нужен для аудита.

Рекомендации по использованию

В зависимости от настроек журнал pgAudit может достигать очень больших объемов. Следует максимально точно определить, что именно должно быть записано в журнал аудита в вашей среде, чтобы избежать слишком большого количества записей.

Например, при работе в среде OLAP, вероятно, было бы неразумно проверять вставки журналов в большую таблицу фактов. Размер файла журнала, скорее всего, будет во много раз больше фактического размера данных вставок, поскольку файл журнала хранится в текстовом виде. Поскольку журналы обычно хранятся на диске вместе с ОС, это может привести к очень быстрому исчерпанию дискового пространства. В тех случаях, когда невозможно ограничить ведение журнала аудита определенными таблицами, обязательно оцените влияние на производительность во время тестирования и выделите достаточно места на томе журнала. Это также может быть верно для сред OLTP. Даже если объем операций вставки не так высок, влияние журнала аудита на производительность все равно может привести к заметному росту задержки.

Чтобы ограничить количество записей аудита отношений для операторов SELECT и DML, рассмотрите возможность использования журнала аудита объектов (см. раздел Аудит объектов). Ведение журнала аудита объектов позволяет выбрать отношения, которые будут регистрироваться, что позволяет уменьшить общий объем журнала. Однако все новые создаваемые отношения должны быть явно добавлены в журнал аудита объектов. В этом случае хорошим вариантом может быть программное решение, в котором некоторые определенные таблицы исключаются из ведения журнала, а все остальные включаются.

Настройки

Настройки могут быть изменены только суперпользователем. Если разрешить обычным пользователям изменять свои настройки, то ведение журнала аудита теряет смысл.

Настройки могут быть заданы глобально (в файле `postgresql.conf` или с помощью `ALTER SYSTEM ... SET`), на уровне базы данных (с помощью `ALTER DATABASE ... SET`) или на уровне роли (с помощью `ALTER ROLE ... SET`). Обратите внимание, что настройки не наследуются через обычное наследование ролей, и `SET ROLE` не изменит настройки pgAudit пользователя. Это ограничение системы ролей, а не особенность pgAudit.

Расширение pgAudit должно быть загружено в shared_preload_libraries. В противном случае во время загрузки возникнет ошибка, и ведение журнала аудита не будет происходить. Кроме того, `CREATE EXTENSION pgaudit` следует вызвать до того, как будут заданы параметры `pgaudit.log`. Если расширение pgaudit удалено и его необходимо воссоздать, то сначала должен быть удален `pgaudit.log`, иначе возникнет ошибка.

pgaudit.log

Указывает, какие классы операторов будут регистрироваться при ведении журнала аудита сессии. Возможные значения:

- **READ**: `SELECT`и `COPY`, если источник - отношение или запрос.
- **WRITE**: `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, и `COPY`, если цель - отношение.
- **FUNCTION**: Вызовы функций и блоки `DO`.
- **ROLE**: Операторы, связанные с ролями и привилегиями: `GRANT`, `REVOKE`, `CREATE/ALTER/DROP ROLE`.
- **DDL**: Все `DDL`, не входящие в класс `ROLE`.
- **MISC**: Прочие команды, включая `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, `SET`.
- **MISC_SET**: Прочие команды `SET`, включая `SET ROLE`.
- **CONNECTION**: События, связанные с подключением к серверу. Существуют 4 типа таких событий: `OPEN`, `CLOSED`, `FAILED`, `CHANGE USER`. Событие `FAILED` регистрируется в случае неудачной попытки аутентификации по паролю и независимо от значения `pgaudit.log`.
- **PROTECTION**: Функции настройки механизма защиты от привилегированных пользователей.
- **ALL**: Включить всё вышеперечисленное.

Можно включить несколько классов, перечислив их через запятую, или исключить определеные классы, поставив перед ними знак - (см. раздел Журнал аудита сессии).

Значение по умолчанию - none.

pgaudit.log_catalog

Указывает, должно ли быть включено ведение журнала сессии в том случае, если все отношения в операторе находятся в pg_catalog. Отключение этого параметра уменьшит шум в журнале от таких инструментов, как psql и pgAdmin, которые часто обращаются к каталогу.

Значение по умолчанию - on.

pgaudit.log_client

Указывает, будут ли сообщения журнала видны клиентскому процессу, такому как psql. Этот параметр обычно следует оставить отключенным, но он может быть полезен для отладки или других целей.

Обратите внимание, что pgaudit.log_level включен только тогда, когда pgaudit.log_client **включен**.

Значение по умолчанию - off.

pgaudit.log_level

Указывает уровень журналирования, который будет использоваться для записей журнала (см. Уровни важности сообщений). Обратите внимание, что значения ERROR, FATAL и PANIC не допускаются). Этот параметр используется для регрессионного тестирования, а также может быть полезен конечным пользователям для тестирования или других целей.

pgaudit.log_level включен только тогда, когда pgaudit.log_client **включен**; в противном случае будет использоваться значение по умолчанию.

Значение по умолчанию - log.

pgaudit.log_parameter

Указывает, что ведение журнала аудита должно включать параметры, переданные вместе с оператором. При наличии параметров они будут включены в формате CSV после текста оператора.

Значение по умолчанию - off.

pgaudit.log_relation

Указывает, должно ли ведение журнала аудита сессии создавать отдельную запись журнала для каждого отношения (TABLE, VIEW и т.д.), на которое ссылается оператор SELECT или DML. Это полезный прием для исчерпывающего ведения журнала без использования журнала аудита объектов.

Значение по умолчанию - off.

pgaudit.log_statement_once

Указывает, будут ли текст и параметры оператора прикрепляться к первой записи в журнале для комбинации оператора и вложенных операторов или к каждой записи. Отключение этого параметра приведет к менее подробному ведению журнала, но может затруднить определение оператора, сгенерировавшего запись журнала, хотя пары оператора и вложенного оператора вместе с идентификатором процесса должно быть достаточно для идентификации текста оператора, записанного в предыдущей записи.

Значение по умолчанию - `off`.

pgaudit.role

Указывает главную роль, используемую для ведения журнала аудита объектов. Можно определить несколько ролей аудита, предоставив их главной роли. Это позволяет нескольким группам отвечать за различные аспекты ведения журнала аудита.

Значения по умолчанию нет.

Ведение журнала аудита сессии

Журнал аудита сессии предоставляет подробные записи всех операций, выполняемых пользователем на сервере.

Конфигурация

Ведение журнала сессии включается с помощью параметра `pgaudit.log`.

Включить ведение журнала сессии для всех `DML` и `DDL` и протоколировать все отношения в операторах DML:

```
set pgaudit.log = 'write, ddl';

set pgaudit.log\_relation = on;
```

Включить ведение журнала сессии для всех команд, кроме `MISC`, и выводить сообщения журнала аудита с пометкой `NOTICE`:

```
set pgaudit.log = 'all, -misc';

set pgaudit.log\_level = notice;
```

Пример

В этом примере ведение журнала аудита сессии используется для ведения журнала операторов `DDL` и `SELECT`. Обратите внимание, что оператор `INSERT` не регистрируется, так как класс `WRITE` не включен.

SQL:

```
set pgaudit.log = 'read, ddl';

create table account
```

```
(
    id int,
    name text,
    password text,
    description text
);

insert into account (id, name, password, description)
             values (1, 'user1', 'HASH1', 'blah, blah');

select *
    from account;
```

Вывод в журнале:

```
AUDIT: SESSION,1,1,DDL,CREATE TABLE,TABLE,public.account,create table account
(
    id int,
    name text,
    password text,
    description text
);,<not logged>
AUDIT: SESSION,2,1,READ,SELECT,,,select *
    from account,,<not logged>
```

Ведение журнала аудита объектов

Журнал аудита объектов регистрирует операторы, влияющие на определенное отношение. Поддерживаются только команды `SELECT`, `INSERT`, `UPDATE` и `DELETE`. `TRUNCATE` не включается в журнал аудита объектов.

Ведение журнала аудита объектов - более подробная альтернатива `pgaudit.log = 'read, write'`. Таким образом, возможно, нет смысла использовать их вместе, но можно, например, использовать ведение журнала сеансов для регистрации каждого оператора, а затем дополнить его ведением журнала объектов, чтобы получить более подробную информацию о конкретных отношениях.

Конфигурация

Ведение журнала аудита на уровне объектов осуществляется через систему ролей. Параметр `pgaudit.role` определяет роль, которая будет использоваться для ведения журнала аудита. Отношение (`TABLE`, `VIEW`) будет записываться в журнал аудита, если роль аудита имеет разрешения для выполняемой команды или наследует разрешения от другой роли. Это позволяет эффективно использовать несколько ролей аудита, даже если в любом контексте существует одна главная роль.

Задайте для `pgaudit.role` значение `auditor` и наделите её правами `SELECT` и `DELETE` над таблицей `account`. Теперь операции `SELECT` и `DELETE` над таблицей `account` будут заноситься в журнал:

```
set pgaudit.role = 'auditor';
```

```
grant select, delete
   on public.account
   to auditor;
```

Пример

В этом примере ведение журнала аудита объектов используется для иллюстрации того, как может быть применен детализированный подход к журналированию операторов SELECT и DML. Обратите внимание, что ведение журнала операций над таблицей account контролируется разрешениями на уровне столбцов, а ведение журнала операций над табицей account_role_map - на уровне таблиц.

SQL:

```
set pgaudit.role = 'auditor';

create table account
(
    id int,
    name text,
    password text,
    description text
);

grant select (password)
   on public.account
   to auditor;

select id, name
  from account;

select password
  from account;

grant update (name, password)
   on public.account
   to auditor;

update account
   set description = 'yada, yada';

update account
   set password = 'HASH2';

create table account_role_map
(
    account_id int,
    role_id int
);

grant select
   on public.account_role_map
```

```
   to auditor;

select account.password,
       account_role_map.role_id
  from account
       inner join account_role_map
            on account.id = account_role_map.account_id
```

Вывод в журнале:

```
AUDIT: OBJECT,1,1,READ,SELECT,TABLE,public.account,select password
  from account,<not logged>
AUDIT: OBJECT,2,1,WRITE,UPDATE,TABLE,public.account,update account
   set password = 'HASH2',<not logged>
AUDIT: OBJECT,3,1,READ,SELECT,TABLE,public.account,select account.password,
       account_role_map.role_id
  from account
       inner join account_role_map
            on account.id = account_role_map.account_id,<not logged>
AUDIT: OBJECT,3,1,READ,SELECT,TABLE,public.account_role_map,select
account.password,
       account_role_map.role_id
  from account
       inner join account_role_map
            on account.id = account_role_map.account_id,<not logged>
```

Формат

Записи аудита записываются в стандартное средство ведения журнала и содержат следующие столбцы в формате CSV. Выходные данные соответствуют формату CSV только в том случае, если префикс строки журнала каждой записи удален.

- **AUDIT_TYPE** - `SESSION` или `OBJECT`.
- **STATEMENT_ID** - Уникальный идентификатор оператора для этой сессии. Каждый идентификатор оператора представляет собой внутренний вызов. Идентификаторы операторов идут последовательно, даже если некоторые операторы не регистрируются. Когда регистрируется более одного отношения, для одного идентификатора оператора может быть несколько записей.
- **SUBSTATEMENT_ID** - Последовательный идентификатор для каждого вложенного оператора в основном операторе. Например, вызов функции из запроса. Идентификаторы вложенных операторов идут последовательно, даже если некоторые вложенные операторы не регистрируются. Когда регистрируется более одного отношения, для одного идентификатора вложенного оператора может быть несколько записей.
- **CLASS** - `READ`, `ROLE` и т.д. (см. раздел `pgaudit.log`).
- **COMMAND** - `ALTER TABLE`, `SELECT` и т.д.

- **OBJECT_TYPE** - TABLE, INDEX, VIEW и т.д. Применимо для SELECT, DML и большинства DDL.
- **OBJECT_NAME** - Полное имя объекта (например, public.account). Применимо для SELECT, DML и большинства DDL.
- **STATEMENT** - Операция, выполняемая на сервере.
- **PARAMETER** - Если параметр pgaudit.log_parameter включен, здесь в кавычках будут перечислены параметры в формате CSV. Если параметров нет, здесь будет <none>. Иначе в этом поле будет запись <not logged>.
- **EXECUTION_RESULT** - Если операция завершается с ошибкой, вызванной недостаточными привилегиями или механизмом защиты от привилегированных пользователей, в этом поле будет запись ERROR: INSUFFICIENT PRIVILEGE. Если операция выполнена успешно и оператор принадлежит к классу PROTECTION, в поле заносится запись SUCCESS.

У записей класса CONNECTION собственный формат.

- **AUDIT_TYPE** - Всегда SESSION.
- **CLASS** - Всегда CONNECTION.
- **EVENT** - OPEN, CLOSED, FAILED, CHANGE USER.
- **SESSION_TIME** - Только для события CLOSED. Формат записи: "session time: %d:%02d:%02d.%03d" - часы, минуты, секунды, миллисекунды соответственно.
- **DATA_BASE** - Имя базы данных, к которой осуществляется подключение. Формат записи: - "database = %s" (имя базы данных).
- **USER** - Имя пользователя, подключающегося к базе данных. Формат записи: - "user = %s" (имя пользователя).

Для добавления любых других полей, необходимых для удовлетворения ваших требований к журналу аудита, может использоваться префикс строки журнала (log_line_prefix). Пример префикса строки журнала: "%m %u %d \[%p\]:" - включить дату/время, имя пользователя, имя базы данных и идентификатор процесса для каждой записи журнала аудита.

Пример 1

Защита включена, пользователь sec_admin выполняет select pm_get_policies();

```
AUDIT: SESSION,2,1,READ,SELECT,,,select pm_get_policies();,<not logged>
AUDIT: SESSION,2,2,PROTECTION,EXECUTE,FUNCTION,pg_catalog.pm_get_policies,select
pm_get_policies();,<not logged>
AUDIT: SESSION,2,2,PROTECTION,EXECUTE,FUNCTION,pg_catalog.pm_get_policies,select
pm_get_policies();,<not logged>,SUCCESS
```

Пример 2

Защита включена, `select pm_get_policies();` выполняет пользователь без соответствующих прав

```
AUDIT: SESSION,2,1,READ,SELECT,,,select pm_get_policies();,<not logged>
AUDIT: SESSION,2,2,PROTECTION,EXECUTE,FUNCTION,pg_catalog.pm_get_policies,select
pm_get_policies();,<not logged>
AUDIT: SESSION,2,2,PROTECTION,EXECUTE,FUNCTION,pg_catalog.pm_get_policies,select
pm_get_policies();,<not logged>,ERROR: INSUFFICIENT PRIVILEGE
```

Пример 3

`select pm_get_policies();` выполняется с отказом по любой другой причине

```
AUDIT: SESSION,2,1,READ,SELECT,,,select pm_get_policies();,<not logged>
AUDIT: SESSION,2,2,PROTECTION,EXECUTE,FUNCTION,pg_catalog.pm_get_policies,select
pm_get_policies();,<not logged>
```

Примечания

Переименования объектов регистрируются под тем именем, на которое они были переименованы. Например, переименование таблицы приведет к следующему результату:

```
ALTER TABLE test RENAME TO test2;

AUDIT: SESSION,36,1,DDL,ALTER TABLE,TABLE,public.test2,ALTER TABLE test RENAME
TO test2,<not logged>
```

Одна команда может быть зарегистрирована несколько раз. Например, когда таблица создается с первичным ключом, указанным во время создания, индекс для первичного ключа будет зарегистрирован сразу, а затем еще раз в рамках записи `CREATE`. При этом несколько записей такого рода будут иметь общий идентификатор оператора.

`Autovacuum` и `Autoanalyze` не регистрируются.

Операторы, выполняемые после того, как транзакция перешла в прерванное состояние, не регистрируются. Однако оператор, вызвавший ошибку, и все последующие операторы, выполненные в прерванной транзакции, будут регистрироваться как ошибки стандартным средством ведения журнала.

Авторы

Расширение аудита PostgreSQL основано на проекте 2ndQuadrant pgaudit project, автором которого являются Саймон Риггс, Абхиджит Менон-Сен и Ян Барвик, и представлено в качестве расширения ядра PostgreSQL. Дополнительные разработки были сделаны Дэвидом Стилом из Crunchy Data.