ЛР3. PostgreSQL. Управление привилегиями

Привилегия – разрешение на использование определенной услуги управления данными для доступа к объекту данных, предоставляемое идентифицированному пользователю.

Роль – это именованная совокупность привилегий, которые могут быть предоставлены пользователям или другим ролям.

PostgreSQL использует концепцию ролей (`roles`) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того, как роль настроена. Роли могут владеть объектами базы данных (например, таблицами и функциями) и выдавать другим ролям разрешения на доступ к этим объектам. Кроме того, можно предоставить одной роли членство в другой роли, таким образом одна роль может использовать права других ролей. Концепция ролей включает в себя концепцию пользователей («`users`») и групп («`groups`»). До версии 8.1 в PostgreSQL пользователи и группы были отдельными сущностями, но теперь есть только роли. Любая роль может использоваться в качестве пользователя, группы, и того и другого.

Роли БД

Для создания роли используется команда `SQL CREATE ROLE:`

```
CREATE ROLE имя;
```

Здесь имя соответствует правилам именования идентификаторов SQL: либо обычное, без специальных символов, либо в двойных кавычках. (На практике, к команде обычно добавляются другие указания, такие как `LOGIN`. Подробнее об этом ниже.) Для удаления роли используется команда `DROP ROLE`:

```
DROP ROLE имя;
```

При начальной настройки кластера базы данных система сразу после инициализации всегда содержит одну предопределённую роль. Эта роль является суперпользователем («`superuser`»). Обычно эта роль называется `postgres`. Для создания других ролей, вначале нужно подключиться с этой ролью.

Атрибуты ролей

Роль базы данных может иметь атрибуты, определяющие её полномочия и взаимодействие с системой аутентификации клиентов.

Право подключения

Только роли с атрибутом LOGIN могут использоваться для начального подключения к базе данных. Роль с атрибутом LOGIN можно рассматривать как пользователя базы данных. Для создания такой роли можно использовать любой из вариантов:

```
CREATE ROLE имя LOGIN;
CREATE USER имя;
```

(Команда CREATE USER эквивалентна CREATE ROLE за исключением того, что CREATE USER по умолчанию включает атрибут LOGIN, в то время как CREATE ROLE — нет.)

Статус суперпользователя

Суперпользователь базы данных обходит все проверки прав доступа, за исключением права на вход в систему.

Создание базы данных

Роль должна явно иметь разрешение на создание базы данных (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется CREATE ROLE имя CREATEDB.

Создание роли

Роль должна явно иметь разрешение на создание других ролей (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется CREATE ROLE имя CREATEROLE. Роль с правом CREATEROLE может не только создавать, но и изменять и удалять другие роли, а также выдавать и отзывать членство в ролях. Однако для создания, изменения, удаления ролей суперпользователей и изменения членства в них требуется иметь статус суперпользователя; права CREATEROLE в таких случаях недостаточно.

Запуск репликации

Роль должна иметь явное разрешение на запуск потоковой репликации (за исключением суперпользователей, которые пропускают все проверки). Роль, используемая для потоковой репликации, также должна иметь атрибут LOGIN. Для создания такой роли используется CREATE ROLE имя REPLICATION LOGIN.

Пароль

Пароль имеет значение, если метод аутентификации клиентов требует, чтобы пользователи предоставляли пароль при подключении к базе данных. Методы аутентификации PASSWORD и md5 используют пароли. База данных и операционная система используют раздельные пароли. Пароль указывается при создании роли: CREATE ROLE имя PASSWORD 'строка'.

Членство в роли

Часто бывает удобно сгруппировать пользователей для упрощения управления правами: права можно выдавать и забирать для всей группы. В PostgreSQL для этого создаётся роль, представляющая группу, а затем членство в этой группе выдаётся ролям индивидуальных пользователей.

Для настройки групповой роли сначала нужно создать саму роль:

```
CREATE ROLE имя;
```

Обычно групповая роль не имеет атрибута LOGIN, хотя при желании его можно установить.

После того как групповая роль создана, в неё можно добавлять или удалять членов, используя команды GRANT и REVOKE:

```
GRANT групповая_роль TO роль1, ... ;
REVOKE групповая_роль FROM роль1, ... ;
```

Членом роли может быть и другая групповая роль (потому что в действительности нет никаких различий между групповыми и не групповыми ролями). При этом база данных не допускает замыкания членства по кругу. Также не допускается управление членством роли PUBLIC в других ролях.

Роли, имеющие атрибут INHERIT, автоматически используют права всех ролей, членами которых они являются, в том числе и унаследованные этими ролями права.

Привилегии

Когда в базе данных создаётся объект, ему назначается владелец. Владельцем обычно становится роль, с которой был выполнен оператор создания. Для большинства типов объектов в исходном состоянии только владелец (или суперпользователь) может делать с объектом всё, что угодно. Чтобы разрешить использовать его другим ролям, нужно дать им права.

Существует несколько типов прав: SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY, EXECUTE и USAGE. Набор прав, применимых к определённому объекту, зависит от типа объекта (таблица, функция и т. д.).

Неотъемлемое право изменять или удалять объект имеет только владелец объекта. Объекту можно назначить нового владельца с помощью команды ALTER для соответствующего типа объекта, например:

```
ALTER TABLE имя_таблицы OWNER TO новый_владелец;
```

Суперпользователь может делать это без ограничений, а обычный пользователь — только если он является одновременно текущим владельцем объекта (или членом роли владельца) и членом новой роли.

Для назначения прав применяется команда GRANT. Например, если в базе данных есть роль joe и таблица accounts, право на изменение таблицы можно дать этой роли так:

```
GRANT UPDATE ON accounts TO joe;
```

Если вместо конкретного права написать ALL, роль получит все права, применимые для объекта этого типа.

Чтобы лишить пользователей прав, используйте команду REVOKE:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

Особые права владельца объекта (то есть права на выполнение DROP, GRANT, REVOKE и т. д.) всегда неявно закреплены за владельцем и их нельзя назначить или отобрать. Но владелец объекта может лишить себя обычных прав, например, разрешить всем, включая себя, только чтение таблицы.

Обычно распоряжаться правами может только владелец объекта (или суперпользователь). Однако возможно дать право доступа к объекту «с правом передачи», что позволит получившему такое право назначать его другим. Если такое право передачи впоследствии будет отозвано, то все, кто получил данное право доступа (непосредственно или по цепочке передачи), потеряют его.

PostgreSQL по умолчанию назначает роли PUBLIC права для некоторых типов объектов, когда эти объекты создаются. Для таблиц, столбцов, последовательностей, обёрток сторонних данных, сторонних серверов, больших объектов, схем и табличных пространств PUBLIC по умолчанию никаких прав не получает. Для других типов объектов PUBLIC получает следующие права по

умолчанию: CONNECT и TEMPORARY (создание временных таблиц) для баз данных; EXECUTE — для функций и процедур; USAGE — для языков и типов данных (включая домены). Владелец объекта, конечно же, может отозвать (посредством REVOKE) как явно назначенные права, так и права по умолчанию. (Для максимальной безопасности команду REVOKE нужно выполнять в транзакции, создающей объект; тогда не образуется окно, в котором другой пользователь сможет обратиться к объекту.) Кроме того, эти изначально назначаемые права по умолчанию можно переопределить, воспользовавшись командой ALTER DEFAULT PRIVILEGES.

Удаление ролей

Так как роли могут владеть объектами баз данных и иметь права доступа к объектам других ролей, удаление роли не сводится к немедленному выполнению DROP ROLE. Сначала должны быть удалены или переданы другим владельцам все объекты, принадлежащие роли; также должны быть отозваны все права, данные роли.

Владение объектами можно передавать в индивидуальном порядке, применяя команду ALTER, например:

```
ALTER TABLE bobs_table OWNER TO alice;
```

Кроме того, для переназначения какой-либо другой роли владения сразу всеми объектами, принадлежащих удаляемой роли, можно применить команду REASSIGN OWNED. После того как все ценные объекты будут переданы новым владельцам, все оставшиеся объекты, принадлежащие удаляемой роли, могут быть удалены с помощью команды DROP OWNED. DROP OWNED также удаляет все права, которые даны целевой роли для объектов, не принадлежащих ей. Так как REASSIGN OWNED такие объекты не затрагивает, обычно необходимо запустить и REASSIGN OWNED, и DROP OWNED (в этом порядке!), чтобы полностью ликвидировать зависимости удаляемой роли.

При попытке выполнить DROP ROLE для роли, у которой сохраняются зависимые объекты, будут выданы сообщения, говорящие, какие объекты нужно передать другому владельцу или удалить.

Политики защиты строк

В дополнение к стандартной системе прав SQL, управляемой командой GRANT, на уровне таблиц можно определить политики защиты строк,

ограничивающие для пользователей наборы строк, которые могут быть возвращены обычными запросами или добавлены, изменены и удалены командами, изменяющими данные. Это называется также защитой на уровне строк (RLS, Row-Level Security). По умолчанию таблицы не имеют политик, так что если система прав SQL разрешает пользователю доступ к таблице, все строки в ней одинаково доступны для чтения или изменения.

Когда для таблицы включается защита строк (с помощью команды `ALTER TABLE ... ENABLE ROW LEVEL SECURITY`), все обычные запросы к таблице на выборку или модификацию строк должны разрешаться политикой защиты строк. (Однако на владельца таблицы такие политики обычно не действуют.) Если политика для таблицы не определена, применяется политика запрета по умолчанию, так что никакие строки в этой таблице нельзя увидеть или модифицировать. На операции с таблицей в целом, такие как `TRUNCATE` и `REFERENCES`, защита строк не распространяется.

Политики защиты строк могут применяться к определённым командам и/или ролям. Политику можно определить как применяемую к командам `ALL` (всем), либо `SELECT`, `INSERT`, `UPDATE` и `DELETE`. Кроме того, политику можно связать с несколькими ролями, при этом действуют обычные правила членства и наследования.

Чтобы определить, какие строки будут видимыми или могут изменяться в таблице, для политики задаётся выражение, возвращающее логический результат. Это выражение будет вычисляться для каждой строки перед другими условиями или функциями, поступающими из запроса пользователя. Строки, для которых это выражение возвращает не `true`, обрабатываться не будут. Чтобы независимо управлять набором строк, которые можно видеть, и набором строк, которые можно модифицировать, в политике можно задать отдельные выражения. Выражения политик обрабатываются в составе запроса с правами исполняющего его пользователя, но для обращения к данным, недоступным этому пользователю, в этих выражениях могут применяться функции, определяющие контекст безопасности.

Суперпользователи и роли с атрибутом `BYPASSRLS` всегда обращаются к таблице, минуя систему защиты строк. На владельца таблицы защита строк тоже не действует, хотя он может включить её для себя принудительно, выполнив команду `ALTER TABLE ... FORCE ROW LEVEL SECURITY`.

Неотъемлемое право включать или отключать защиту строк, а также определять политики для таблицы, имеет только её владелец.

Для создания политик предназначена команда `CREATE POLICY`, для изменения — `ALTER POLICY`, а для удаления — `DROP POLICY`. Чтобы включить или отключить защиту строк для определённой таблицы, воспользуйтесь командой `ALTER TABLE`.

Каждой политике назначается имя, при этом для одной таблицы можно определить несколько политик. Так как политики привязаны к таблицам, каждая политика для таблицы должна иметь уникальное имя. В разных таблицах политики могут иметь одинаковые имена.

Когда к определённому запросу применяются несколько политик, они объединяются либо логическим сложением (если политики разрешительные (по умолчанию)), либо умножением (если политики ограничительные). Это подобно тому, как некоторая роль получает права всех ролей, в которые она включена. Разрешительные и ограничительные политики рассматриваются ниже.

В качестве простого примера, создать политику для отношения `account`, позволяющую только членам роли `managers` обращаться к строкам отношения и при этом только к своим, можно так:

```
CREATE TABLE accounts (manager text, company text, contact_email text);

ALTER TABLE accounts ENABLE ROW LEVEL SECURITY;

CREATE POLICY account_managers ON accounts TO managers
    USING (manager = current_user);
```

Эта политика неявно подразумевает и предложение `WITH CHECK`, идентичное предложению `USING`, поэтому указанное ограничение применяется и к строкам, выбираемым командой (так что один менеджер не может выполнить `SELECT`, `UPDATE` или `DELETE` для существующих строк, принадлежащих другому), и к строкам, изменяемым командой (так что командами `INSERT` и `UPDATE` нельзя создать строки, принадлежащие другому менеджеру).

ЛР3. Вопросы для контроля

1. Привилегии. Роли.
2. Атрибуты ролей.
3. Привилегии. Удаление привилегий.
4. Удаление ролей.
5. Политики защиты строк.

ЛР3. Задание

1. Создать две роли `user1`, `user2` и новую схему `new_scheme`. Предоставить роли `user1` возможность создавать новые объекты в схеме `new_scheme`.

2. Ролью `user1` создать таблицы `table1` и `table2` в созданной схеме `new_scheme`. Заполнить их.

3. Ролью `user2` попытаться считать данные таблицы.

4. С помощью роли `user1` предоставить пользователю user2 возможность чтения данных таблицы table1

5. С помощью роли `user1` создать функцию, возвращающую число строк в таблице `table2`.

6. Ролью `user2` попытаться выполнить созданную функцию.

7. Изменить созданную пользователем `user1` функцию так, чтобы ею мог воспользоваться пользователь `user2`.

8. С помощью роли `postgres` создать новую таблицу `table3` в схеме `new_scheme` и включить роль `user1` в суперпользовательскую роль `postgres`. С помощью роли `user1` попытаться получить доступ к таблице `table3`.

9. Удалить созданные роли.

10. Настроить политику защиты доступа к таблице на уровне строк. Пусть имеется три роли (`admin, seller1, seller2`) и таблица `sales`. В данной таблице содержится информация о продажах каких-либо товаров в разных городах (`id,city,name,amount`). Требуется настроить политику доступа таким образом, чтобы `admin` имел полный доступ ко всем данным таблицы, `seller1` мог просматривать и изменять данные только для города "`Moscow`", `seller2` мог просматривать и изменять данные только для города "`Samara`".