

Задание:

Для спроектированной ранее базы данных предметной области предложить категории пользователей и их права (как минимум на чтение таблиц/строк/столбцов, на запись в различные таблицы), настроить их на уровне СУБД. Продемонстрировать различие в возможностях созданных пользователей и владельца базы данных.

Роли в PostgreSQL

Роли включают в себя группы и пользователя. Главное их отличие в том, что пользователи имеют логины, в отличие от групп.

Для создания и удаления роли используется команда CREATE ROLE/DROP ROLE соответственно:

```
CREATE ROLE имя;
```

```
DROP ROLE имя;
```

Часто бывает удобно сгруппировать пользователей для упрощения управления правами: права можно выдавать для всей группы и у всей группы забирать. В Postgres 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. После того как групповая роль создана, в неё можно добавлять или удалять членов, используя команды GRANT и REVOKE:

```
GRANT групповая_роль TO роль
```

```
REVOKE групповая_роль FROM роль
```

Каждый член группы может явно выполнить SET ROLE, чтобы временно «стать» групповой ролью. В этом состоянии сеанс базы

данных использует полномочия групповой роли вместо оригинальной роли, под которой был выполнен вход в систему. При этом для всех создаваемых объектов базы данных владельцем считается групповая, а не оригинальная роль. RESET ROLE позволяет выйти из временной роли.

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

Привилегии:

Когда создаётся любой объект, ему назначается владелец. Обычно владелец — это та роль, которая запустила оператор создания данного объекта. Для большинства видов объектов, начальное состояние таково, что делать что-либо с объектом может только владелец (или суперпользователь). Чтобы разрешить другим ролям использовать его, им должны быть предоставлены привилегии.

Существует различные виды привилегий, например: SELECT, INSERT, UPDATE, DELETE. Эти привилегии применяются к отдельному объекту, в зависимости от типа объекта (таблица, функция и т.д.)

Выдача привилегий ролям производится при помощи команды GRANT.

Основной синтаксис для команды GRANT выглядит следующим образом:

GRANT privilege [, ...]

ON object [, ...]

TO { PUBLIC | GROUP group | username }

значениями **privilege** могут быть: SELECT, INSERT, UPDATE, DELETE, ALL.

object — имя объекта, к которому предоставляется доступ. Возможные объекты: таблица, представление, последовательность

PUBLIC — краткая форма, представляющая всех пользователей.

group – группа, которой нужно предоставить привилегии.

username – имя пользователя, которому необходимо предоставить привилегии.

Чтобы отозвать привилегию, надо выполнить команду REVOKE [привилегия] FROM [роль];

Ролевая политика:

Политика защиты строк таблицы позволяет предоставлять разным ролям доступ только к определенному множеству строк. Это множество строк получается накладыванием определенных условий, которые задаются в синтаксисе создания политики.

```
CREATE POLICY имя ON имя_таблицы
    [ FOR { ALL | SELECT | INSERT | UPDATE | DELETE } ]
    [ TO { имя_роли | PUBLIC | CURRENT_USER |
SESSION_USER } [, ...] ]
    [ USING ( выражение_USING ) ]
    [ WITH CHECK ( выражение_CHECK ) ]
имя
```

Имя создаваемой политики. Оно должно отличаться от имён других политик для этой таблицы.

имя_таблицы

Имя (возможно, дополненное схемой) существующей таблицы (или представления), к которой применяется эта политика.

команда

Команда, к которой применяется политика. Допустимые варианты: ALL, SELECT, INSERT, UPDATE и DELETE. ALL (все) подразумевается по умолчанию. Особенности их применения описаны ниже.

имя_роли

Роль (роли), к которой применяется политика.

Выражение_USING

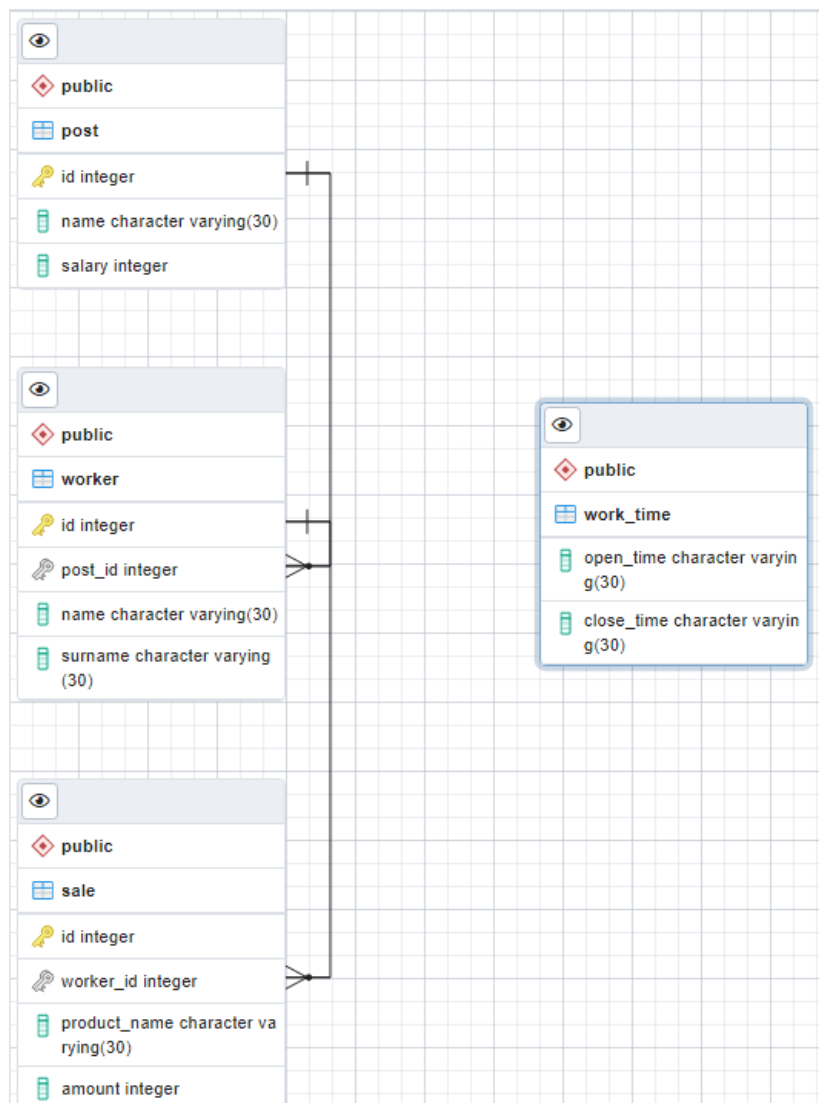
Произвольное условное выражение SQL. Все строки, для которых это выражение возвращает false или NULL, не будут видны пользователю (в запросе SELECT), и не будут доступны для модификации (запросами UPDATE или DELETE). Такая строка просто пропускается, ошибка при этом не выдаётся.

Выражение_CHECK

строки, которые могут быть созданы командами INSERT или UPDATE проверяются по выражению, указанному в WITH CHECK

Реализация:

Для демонстрации выполнения л/р была создана бд о продажах некого магазина. Она включает в себя 3 связанных между собой таблички должность(post), работник(worker) и продажа(sale). Так же имеется одна отдельная табличка work_time, хранящая информацию о времени рабочего дня некого магазина



Создадим некоторые роли:

Простой пользователь, директор, менеджер, продавец

```
CREATE ROLE base_user NOINHERIT;
```

```
CREATE ROLE director INHERIT;
```

```
CREATE ROLE manager INHERIT;
```

```
CREATE ROLE saler INHERIT;
```

base_user будет групповой ролью и не будет наследовать привилегии других ролей (пропишем NOINHERIT).

Добавим членство групповой роли:

```
GRANT base_user TO director, manager, saler, buyer;
```

Т.к. другие роли созданы с ключевым словом INHERIT, то они будут наследовать привилегии своей группы, т.е. у них будет возможность делать то, что разрешено групповой роли base_user.

Проверим:

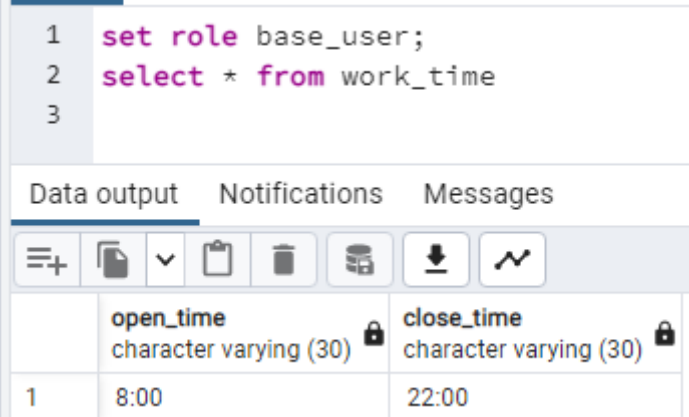
Разрешим базовому пользователю просматривать часы работы магазина:

```
GRANT SELECT ON work_time TO base_user;
```

Выполним запрос от базового пользователя

```
set role base_user;
```

```
select * from work_time
```



The screenshot shows a database client window with a SQL editor at the top containing two lines of code: `set role base_user;` and `select * from work_time`. Below the editor is a toolbar with icons for various actions. The main area displays the results of the SQL query in a table format. The table has two columns: `open_time` and `close_time`, both of type `character varying (30)`. The first row of data shows the values `8:00` and `22:00`.

	open_time character varying (30)	close_time character varying (30)
1	8:00	22:00

Запрос успешно выполнен.

Создадим новую роль, которая не будет входить в членство групповой роли базового пользователя и выполним аналогичный запрос

```
1 create role test
2 set role test
3 select * from work_time
4
```

Data output Notifications Messages

ERROR: ОШИБКА: нет доступа к таблице work_time

SQL state: 42501

Получена соответствующая ошибка т.к. у самой роли также нет привилегий на просмотр таблицы work_time

Выдадим роли директора право на удаление/изменение/правку/вставку данных во все таблицы:

GRANT SELECT, INSERT, UPDATE, DELETE

ON ALL TABLES IN SCHEMA public TO director;

Добавим информации в связанные таблицы от роли директора

set role director

insert into post values(default, 'saler', 20000),

(default, 'oldest_saler', 30000);

insert into worker values(default, 1, 'Aleksandr', 'Ivanov'),

(default, 2, 'Maksim', 'Petrov');

insert into sale values(default, 1, 'notebook', 1), (default, 1, 'mousepad', 3),

(default, 2, 'iphone 12', 5), (default, 2, 'apple airpods 2', 7);

Вставки были успешно произведены.

```
1 set role base_user
2
3 insert into post values(default, 'noinsert', 88888)
4
5
```

Data output Notifications Messages

ERROR: ОШИБКА: нет доступа к таблице post

SQL state: 42501

Теперь
удостоверимся, что у
обычного
пользователя не
появились такие же
привелегии

Получена ожидаемая ошибка

Добавим менеджеру возможность управлять табличками продаж и работников:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON sale, worker TO manager;
```

Теперь попробуем реализовать ролевую политику, так чтобы продавец мог смотреть и модифицировать информацию о продажах, которые совершил именно он.

Для начала создадим представление, которое будет выдавать id работников и сочетание имени фамилии в удобном формате (name_surname).

```
create or replace view workers_id_view as
select w.id, w.name || '_' || w.surname as name_surname
from sale as s join worker as w
on w.id = s.worker_id;
```

Оно будет выглядеть следующим образом:

	id integer	name_surname text
1	2	Maksim_Petrov
2	2	Maksim_Petrov
3	1	Aleksandr_Ivanov
4	1	Aleksandr_Ivanov

Выдадим права на модификацию таблицы продаж роли saler

```
grant SELECT, INSERT, UPDATE, DELETE on sale to saler;
```

Выдадим права на просмотр представления для роли saler

```
grant select on workers_id_view to saler
```

Создадим роль продавца, имя фамилия которого Aleksandr Ivanov

```
create role Aleksandr_Ivanov LOGIN;
```

Навесим на этого продавца роль saler

```
grant saler to Aleksandr_Ivanov;
```

Поставим защиту строк на таблицу sale. Без этой защиты политики не будут срабатывать.

```
alter table sale enable row level security;
```

Создадим политику. Для того, чтобы проверить текущий ли пользователь совершил продажу, мы будем обращаться к представлению. Будем проверять равен ли worker_id из таблицы sale айдишнику, который соответствует имени фамилии пользователя(то есть конкретному продавцу в нашем случае)

```
CREATE POLICY saler_policy ON sale FOR ALL TO saler
```

```
USING (worker_id in (SELECT id FROM workers_id_view WHERE LOWER(name_surname) = current_user))
```

```
WITH CHECK (worker_id in (SELECT id FROM workers_id_view WHERE LOWER(name_surname) = current_user));
```

Таблица sale выглядит следующим образом:

	id [PK] integer	worker_id integer	product_name character varying (30)	amount integer
1	1	1	notebook	2
2	2	1	mousepad	4
3	7	2	headphones	4
4	8	2	chair	1

id пользователя Aleksandr Ivanov = 1

	id [PK] integer	post_id integer	name character varying (30)	surname character varying (30)
1	1	1	Aleksandr	Ivanov
2	2	2	Maksim	Petrov

Попробуем посмотреть от таблицу sale с правами этого пользователя:


```
set role Aleksandr_Ivanov;
```

```
select * from sale;
```

	id [PK] integer	worker_id integer	product_name character varying (30)	amount integer
1	1	1	notebook	2
2	2	1	mousepad	4

В результате пришли только те строки, в которых worker_id равен id нашего пользователя. То есть строки о продажах которые совершил именно наш пользователь.

Попробуем изменить информацию а продажах от лица пользователя:

```
update sale set amount = amount + 1;
```

	id [PK] integer	worker_id integer	product_name character varying (30)	amount integer
1	1	1	notebook	3
2	2	1	mousepad	5
3	7	2	headphones	4
4	8	2	chair	1

В таблице sale инкрементировалось кол-во продаж только в тех строках, которые соответствуют продажам именно активного пользователя.

Все тесты прошли проверку, ролевая политика отрабатывает корректно

Роли: директор, менеджер, продавец

Директор делает все со всеми табличками

Менеджер делает все с табличками воркера и должностей

Воркер делает все с продажами, которые сделал он