- Для чего нужны роли?
    - Для обеспечения различного уровня доступа к объектам бд и к управлению этой бд для различных пользователей и групп.
- Что такое схема?
    - Пространство имен, которое содержит именованные объекты бд такие, как таблицы tables, views, indexes, data types, functions.
- Рассказать про директивы GRANT и REVOKE.
    - GRANT - определить права доступа
    - REVOKE - забрать права доступа
- Для чего нужна роль PUBLIC?
    - Для назначения права всем ролям (в том числе и пока не существующим) в системе можно использовать специальное имя «роли»: `PUBLIC`
- Как добавить нового пользователя в текущую базу данных?
    -
- Как позволить пользователю заходить на сервер?
    - Дать ему атрибут login
    - `CREATE ROLE имя LOGIN;`
    - `CREATE USER имя;`
    - ну или ALTER ROLE имя LOGIN; (Наверно должно работать)
- Какие существуют права?
    - `ALL PRIVILEGES (все привелигии)`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `REFERENCES`, `TRIGGER`, `CREATE`, `CONNECT`, `TEMPORARY`, `EXECUTE` и `USAGE`
- Сменить владельца базы данных.
    - REASSIGN OWNED
      `REASSIGN OWNED BY { старая_роль | CURRENT_USER | SESSION_USER } [, ...]`
      `TO { новая_роль | CURRENT_USER | SESSION_USER }`
- Сменить пароль для пользователя.
    - ALTER USER user_name WITH PASSWORD 'new_password';
- Определить роль с заданными правами.
    - `GRANT UPDATE ON accounts TO joe;`
      `NOINHERIT/при создании роли`
      `Вообще лучше открой`
      https://postgrespro.ru/docs/postgresql/11/role-membership
      https://postgrespro.ru/docs/postgrespro/11/app-createuser
      https://postgrespro.ru/docs/postgrespro/11/sql-grant
- Рассказать о CHECK OPTION.
    - `WITH [ CASCADED | LOCAL ] CHECK OPTION`
    - Это указание управляет поведением автоматически изменяемых представлений. Если оно присутствует, при выполнении операций

INSERT и UPDATE с этим представлением будет проверяться, удовлетворяют ли новые строки условию, определяющему представление (то есть, проверяется, будут ли новые строки видны через это представление). Если они не удовлетворяют условию, операция не будет выполнена. Если указание CHECK OPTION отсутствует, команды INSERT и UPDATE смогут создавать в этом представлении строки, которые не будут видны в нём. Поддерживаются следующие варианты проверки:

- LOCAL

  Новые строки проверяются только по условиям, определённым непосредственно в самом представлении. Любые условия, определённые в нижележащих базовых представлениях, не проверяются (если только в них нет указания CHECK OPTION).

- CASCADED

  Новые строки проверяются по условиям данного представления и всех нижележащих базовых. Если указано CHECK OPTION, а LOCAL и CASCADED опущено, подразумевается указание CASCADED.

- Нельзя использовать с рекурсивными представлениями

- Рассказать о модификации данных через представления/Рассказать о вставке данных через представления.
  - https://postgrespro.ru/docs/postgrespro/11/rules-views#id-1.8.6.7.6
  - Если подзапрос выбирает данные из одного базового отношения и он достаточно прост, механизм перезаписи может автоматически заменить его нижележащим базовым отношением, чтобы команды INSERT, UPDATE или DELETE обращались к базовому отношению. Представления, «достаточно простые» для этого, называются *автоматически изменяемыми*. Подробнее виды представлений, которые могут изменяться автоматически, описаны в CREATE VIEW.
  - Эту задачу также можно решить, создав триггер INSTEAD OF для представления. В этом случае перезапись будет работать немного по-другому. Для INSERT механизм перезаписи не делает с представлением ничего, оставляя его результирующим

отношением запроса. Для `UPDATE` и `DELETE` ему по-прежнему придётся разворачивать запрос представления, чтобы получить «старые» строки, которые эта команда попытается изменить или удалить. Поэтому представление разворачивается как обычно, но в запрос добавляется ещё один элемент списка отношений, указывающий на представление в роли результирующего отношения.

- Продемонстрировать изменение и вставку данных через представления.
    - Изменения прямо в коде есть
- Продемонстрировать полезность материализованного представления.
    - предоставить быстрый доступ к данным, получаемым с удаленной системы через обертку сторонних данных
        - Если периодическое обновление данных из другого источника в локальной базе данных вас устраивает, этот подход может дать значительный выигрыш в скорости.
        - Заметьте, что мы также использовали возможность добавить индекс в материализованное представление, тогда как `file_fdw` индексы не поддерживает; при других видах доступа к сторонним данным такого преимущества может не быть.
    - Если пользователям нужно быстро обработать исторические данные, возможно их интересуют только общие показатели