**Лекция № 1**

**Лекция № 2**

**Лекция № 3**

**Лекция № 4**

**Реляционная модель и реляционная алгебра.**

Отношения удобно представлять в виде смежной таблицы с именованными столбцами. Каждая строка представляет один кортеж.

Отношение первой степени называется унарным, второй – бинарным, третей – тернарным. Nой степени – Nарной. Более строгим эквивалентным определением отношения является его определение через понятие декартового произведения.

Декартовым произведением ( D1 x D2 x D3 … Dn ) – является множество всех возможных упорядоченных кортежей ((d1, d2 … dn), di принадлежит Di), если f является подмножеством декартовых произведений D1 x D2 x D3 … Dn.

Строго говоря порядок кортежей не существенный, так как оно является множеством. Не следует это путать с упорядоченностью по определению, которое означает, что все кортежи в предложении должны различаться. То есть унарные отношения не должно содержать повторяющихся элементов. Перестановка элементов внутри кортежа формально приводит нас к другому отношению. Однако это требование можно ослабить идентифицировав(поименовав) каждую позицию кортежей, то есть столбец в табличном представлении отношений.

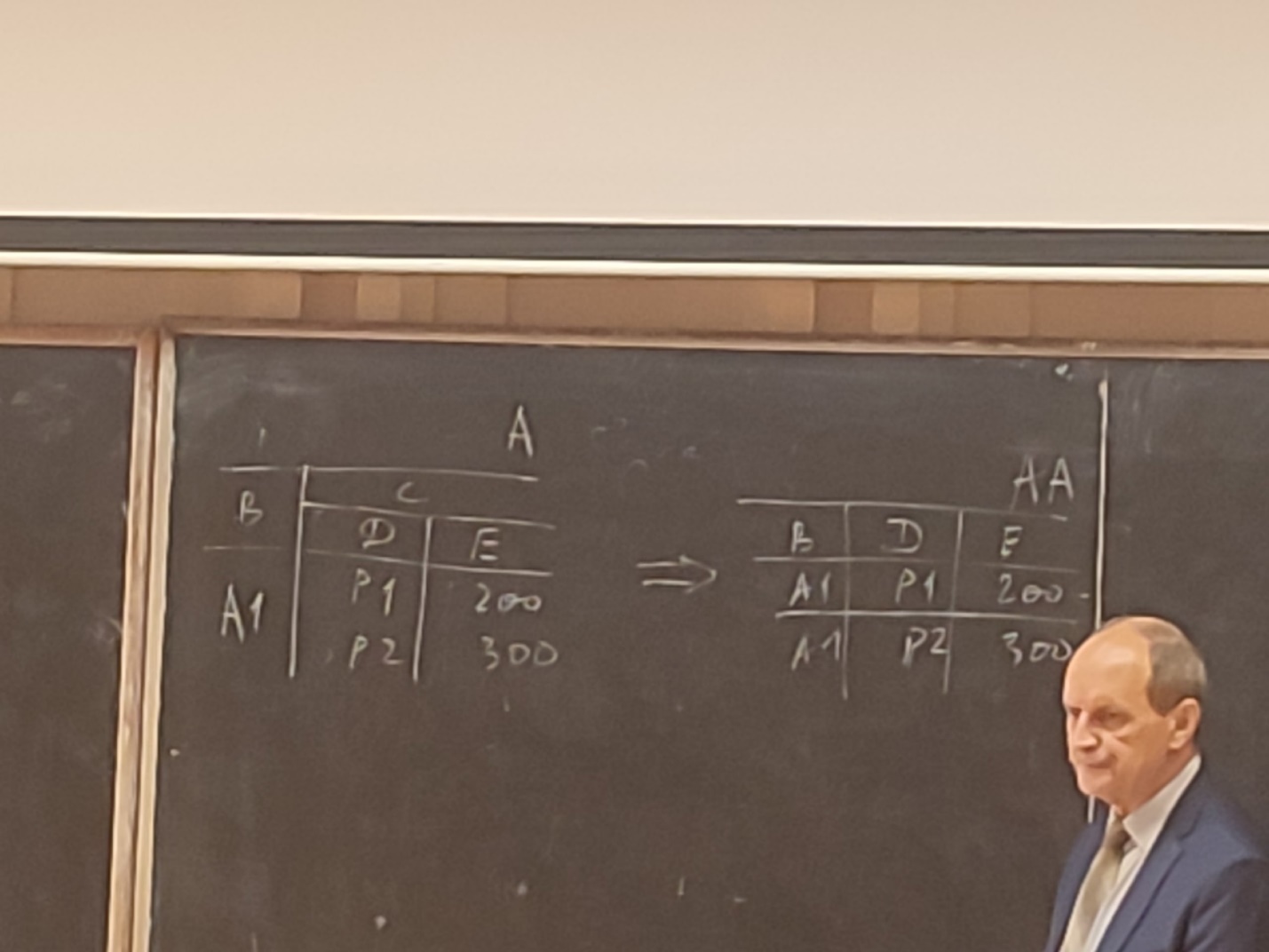
Важно понимать разницу между доменом и столбцами или атрибутами отношений, которые образуются из этого домена. Атрибут представляет использование домена внутри отношения. Возможно использование нескольких атрибутов из одного домена внутри некоторого отношения.



Это означает, что имена не всегда совпадают.

Не следует забывать, что отношение определено для множеств, каждое множество по определению содержит пустое множество. В общем случае отношение может содержать пустые значения.

Введем также понятие нормализации. Единственным отношением допустимым реляционным моделям, являются отношения значения каждого атрибута каждым кортежа которых являются атомарные(неделимые). То есть на пересечении любой строки и столбца в таблице должно быть только одно значение, а не множество. Такие значения называются нормализованными.



Важным элементом определений отношений является упорядоченность кортежей. С ней связано понятие ключа, первичного и внешнего. Предположим, что значение некоторого атрибута однозначно. Тогда этот атрибут может быть назван первичным ключом для данного отношения.

Не каждое отношение будет иметь первичный ключ в виде единственного атрибута, однако, каждое отношение по определению будет иметь некоторую комбинацию атрибутов, которые вместе взятые будут однозначно идентифицировать кортеж в отношениях. Таким образом, каждое отношение имеет (возможно составной) первичный ключ.

Возможно наличие нескольких атрибутов, имеющих уникальные значения. Это приводит к необходимости явного указания какой из них будут использоваться в качестве первичного ключа. Первичный ключ не должен иметь пустых значений. Атрибут отношения P1 – внешний ключ, если этот атрибут не первичный ключ P1, но его значение является значением первичного ключа некоторого отношения P2.

Ключи первичный и внешний представляют собой средства выражения связей между кортежами, хотя и не исчерпывают их.

Создать таблицу можно с помощью CREATE TABLE *name*(

elemName elemType

…

)

В каждой таблице для pg есть несколько столбцов неявно определенной для системы. Их имена нельзя использовать. А именно: tableid, xmin, xmax, cmin, cmin, did. Уже существующую таблицу можно изменить с помощью команды ALTEP TABLE. Добавить, удалить столбцы, добавить и удалить ограничения, изменять значение по умолчанию, изменять тип столбца.

Если таблица больше не нужна ее можно удалить выполнив команду DROP TABLE w. Исчезает и значения и таблица.

Самое простое понятие алгебры - это множество элементов и операций над ними, в результате выполнения которых получаются элементы того же множества. На последнее обратить внимание. Следовательно реляционная алгебра представляет собой совокупность операций над отношениями. Полная алгебра состоит из двух групп операций, традиционных операций на множества (объединение, пересечение, вычитание, декартовое произведение) и специальных операций (выборка, проекция), так как результатом операции реляционной алгебры является отношения, то выражения могут быть вложенными. Более того такая вложенность может достигать любой глубины.

Для добавления строк в таблицу используется:

INSERT INTO tablename VALUES('s', 12, 43, 3214, ’11.12.2003’);

Строки можно удалить из таблицы:

DELETE FROM tablename WHERE id=4;

DELETE FROM tablename; (все записи)

Для выполнения других операций используется оператор SELECT

SELECT \* FROM tablename;

WHERE id=4 – для ограничения множества.

Используется логическое выражение и выбираются только истинное выражение.

Можно получить результаты запроса в определенном порядке ORDER BY c;

Можно убрать дублирующиеся строки из результата запроса DISTINCT

Запросы могут обращаться сразу к нескольким таблицам или к той же таблице так, что одновременно будут обрабатываться разные ее свойства. Этим занимается JOIN.

Если имена столбцов в дух таблицах повторяется необходимо дополнить имена столбцов префиксом указывающим на отношение.

**Лекция № 5**

**Лекция № 6**

|  |  |  |  |
| --- | --- | --- | --- |
| н | ВН | ВК | Т |
| 1 | 9 | 10 | Б |
| 1 | 14 | 15 | С |
| 1 | 10 | 12 | ПВ |
| 1 | 15 | 18 | ПА |

Связи: (н, вн), (н, вк), (т, вн), (т,вк)

Однако существует зависимость Т => Н

В таком отношении по ошибке можно приписать Т = Б, хотя в таком отношении Н = 1

Делаем декомпозицию.

**4-нф**

Отношение находится в 4-нф, если оно находится в вк-нф и все многозначные зависимости являются функциональными зависимостями от возможных ключей.

R(A, B, C)

R.A => R.B

Существует только в том случае, если множество значений B соответствующее паре A и C зависит только от A и не зависит от C. Для предотвращения аномалий такое отношение необходимо разделить, разместив независимые факты в разные отношения.

Отношение безопасности определяется на 2 подуровнях:

1. Доступ к модели БД
2. Действие осуществляемое в рамках этого доступа

Доступ к концептуальной модели определяется внешними моделями каждую из которых представляет свой обобщенный пользователь Пi.

Без потери общности мы можем представить БД одним отношением. С одной стороны все пользователи и имеют доступ к этой таблице с другой она представляет множество разных ячеек доступ к которым осуществляется свое подмножество пользователей. Следовательно из отношения T можно выделить множество отношений к Ty композиция которых эквивалентна исходному отношению.

Пусть, например для отношения T существуют 4 пользователя, которые имеют к ним доступ следующим образом:

Тогда отношение T можно представить в виде 4 взаимосвязанных таблиц T1-T4. Доступ к каждой из которых будет иметь только один пользователь T1-T4. Если в первом случае каждый пользователь имеет доступ видя всю базу, то во втором он ограничен отношениями своей внешней модели и может даже не догадываться о существовании в базе других отношений. Очевидно, что с точки зрения безопасности НСБ второй вариант представления отношений лучше первого.

Таким образом отношение доступа фиксирует пересечения внешних моделей данных, которые в общем случае являются значительно более сложными, в отличии от примера…

Рассмотрим пример, когда доступы пользователей пересекаются.

Как можем видеть из этого примера каждый пользователь имеет доступ к БД через 4 соответствующих отношений. С одной стороны и каждому отношению имеет доступ различные пользователи. Появляются группы доступа, которые подчиняют принятые структуры и концептуальные модели данных. В первоначальном отношении имеем одну группу доступа.

Действия пользователей в рамках отношений определяются матрицей доступа. Матрица доступа фиксирует результат декомпозиции.

Определить формально какая декомпозиция исходного отношения T является наилучшей с точки зрения обеспечения безопасности доступа затруднительно и ее решение следует руководствоваться следующими двумя принципами:

1. Пользователь получает доступ ко всему отношению
2. Структура концептуальной модели БД должна быть определена таким образом, чтобы избыточная для внешних моделей информация была минимальной.

В процессе модификации базы вновь возникающие внешние модели накладываются на существующую структуру БД и базовые группы доступа определенные на этапе ее создания. Однако, если принципы безопасности начинают нарушаться, следует провести реорганизацию БД, несмотря на трудоемкость этого процесса. До определенной степени реорганизации концептуальной модели можно избегать, повышая избыточность БД, за счет локальных внутренних копий требуемых отношений, сохраняя целостность базы.

Принятие подобных решений должно быть компенсировано требованиями безопасности доступа и отсутствием иных способов обеспечить. Существуют различные структуры концептуальной модели БД обеспечивающие ее внешние модели. Зависимости от принятой структуры концептуальной модели возникают группы доступа к базе, отображаемые матрицей доступа.

**Лекция № 7**

**Команда COPY в Postgres SQL.**

Синтаксис

Есть в PG. Позволяет копировать данные между файлом и таблицей БД.

COPY UT [(UC[, …])]

FROM {‘иф’ | PROGRAM ‘k’ | stdint}

[[with](П1)]

[WHERE усл.]

COPY {ит [(ис[,…])] | (запрос)}

TO {‘иф’| PROGRAK ‘k’ | SDOUT}

[[WITH](П1)]

Ит – имя таблицы

Ис – имя столбца

Иф – имя файла

К – команда

П1 – параметры with

Усл - условие

П1:

FORMAT - имя форм

DELIMITEP - разделитель

HEADER - [Boolean | MATCH]

ENCODING - имя кодировки

COPY перемещает данные между таблицами Postgres Pro и обычными файлами в файловой системе. COPY TO копирует содержимое таблицы в файл, а COPY FROM — из файла в таблицу (добавляет данные к тем, что уже содержались в таблице). COPY TO может также скопировать результаты запроса SELECT.

Если указывается список столбцов, COPY TO копирует в файл только данные указанных столбцов, а COPY FROM вставляет каждое поле из файла в соответствующий ему по порядку столбец из указанного списка. В случае отсутствия в этом списке каких-либо столбцов таблицы при COPY FROM они получают значения по умолчанию.

COPY с именем файла указывает серверу Postgres Pro читать или записывать непосредственно этот файл. Заданный файл должен быть доступен пользователю Postgres Pro (тому пользователю, от имени которого работает сервер), и путь к файлу должен задаваться с точки зрения сервера. Когда указывается параметр PROGRAM, сервер выполняет заданную команду и читает данные из стандартного вывода программы, либо записывает их в стандартный ввод. Команда должна определяться с точки зрения сервера и быть доступной для исполнения пользователю Postgres Pro. Когда указывается STDIN или STDOUT, данные передаются через соединение клиента с сервером.

Каждый процесс, выполняющий операцию COPY, будет выдавать информацию о ходе её выполнения, отображаемую в представлении pg\_stat\_progress\_copy.

Основные параметры команды:

имя\_таблицы - имя существующей таблицы (возможно, дополненное схемой).

имя\_столбца - необязательный список столбцов, данные которых будут копироваться. Если этот список отсутствует, копируются все столбцы таблицы, за исключением генерируемых.

запрос - команда SELECT, VALUES, INSERT, UPDATE или DELETE, результаты которой будут скопированы. Заметьте, что запрос должен заключаться в скобки.

Для запросов INSERT, UPDATE и DELETE должно задаваться предложение RETURNING и в целевом отношении не должно быть условного правила, правила ALSO или правила INSTEAD, разворачивающегося в несколько операторов.

имя\_файла - путь входного или выходного файла. Путь входного файла может быть абсолютным или относительным, но путь выходного должен быть только абсолютным. Пользователям Windows следует использовать формат E'' и продублировать каждую обратную черту в пути файла.

PROGRAM

Выполняемая команда. COPY FROM читает стандартный вывод команды, а COPY TO записывает в её стандартный ввод.

Команда запускается через командную оболочку, так что, если требуется передать какие-либо аргументы, поступающие из не доверенного источника, необходимо аккуратно избавиться от всех спецсимволов, имеющих особое значение в оболочке, либо экранировать их. По соображениям безопасности лучше ограничиться фиксированной строкой команды или как минимум не позволять пользователям вводить в неё произвольное содержимое.

STDIN

Указывает, что данные будут поступать из клиентского приложения.

STDOUT

Указывает, что данные будут выдаваться клиентскому приложению.

FORMAT

Выбирает формат чтения или записи данных: text (текстовый), csv (значения, разделённые запятыми, Comma Separated Values) или binary (двоичный). По умолчанию выбирается формат text.

DELIMITER

Задаёт символ, разделяющий столбцы в строках файла. По умолчанию это символ табуляции в текстовом формате и запятая в формате CSV. Задаваемый символ должен быть однобайтовым. Для формата binary этот параметр не допускается.

HEADER

Указывает, что файл содержит строку заголовка с именами каждого столбца в файле. При выводе первая строка будет содержать имена столбцов из таблицы. При вводе первая строка отбрасывается, если для этого параметра установлено значение true (или равнозначное логическое значение). Если для этого параметра установлено значение MATCH, имена столбцов (и их количество) в строке заголовка должны совпадать с фактическими именами столбцов таблицы по порядку, иначе возникнет ошибка. Этот параметр не допускается при использовании формата binary. Параметр MATCH действителен только для команд COPY FROM.

ENCODING

Указывает, что файл имеет кодировку имя\_кодировки. Если этот параметр опущен, выбирается текущая кодировка клиента.

Команду COPY TO можно использовать только с простыми таблицами, не представлениями, и при этом она не копирует строки из дочерних таблиц или секций. Команда COPY копирует те же строки, что выдаёт запрос SELECT \* FROM ONLY t1. Для выгрузки всех строк представления или таблицы с учётом иерархии наследования или секционирования можно применить COPY (SELECT \* FROM таблица) TO

COPY FROM можно применять с обычными, сторонними и секционированными таблицами или представлениями, в которых установлены триггеры INSTEAD OF INSERT.

В таблице, данные которой читает команда COPY TO, требуется иметь право на выборку данных, а в таблице, куда вставляет значения COPY FROM, требуется право на добавление.

Если для таблицы включена защита на уровне строк, соответствующие политики SELECT будут применяться и к операторам COPY таблица TO. Операторы COPY FROM для таблиц с защитой строк в настоящее время не поддерживаются. Вместо них следует использовать равнозначные операторы INSERT.

Файлы, указанные в команде COPY, читаются или записываются непосредственно сервером, не клиентским приложением. Поэтому они должны располагаться на сервере или быть доступными серверу, а не клиенту. Они должны быть доступны на чтение или запись пользователю, от имени которого работает сервер, не клиенту. Аналогично, команда, указанная параметром PROGRAM, выполняется непосредственно сервером, а не клиентским приложением, и должна быть доступна на выполнение пользователю. Выполнять COPY с указанием файла или внешней команды разрешено только суперпользователям базы данных или членам ролей pg\_read\_server\_files, pg\_write\_server\_files или pg\_execute\_server\_program, так как это позволяет читать/записывать любые файлы и запускать любые программы, к которым имеет доступ сервер.

Не путайте команду COPY с \copy. \copy вызывает COPY FROM STDIN, а затем работает с данными в файле, доступном клиенту psql. Таким образом, когда применяется команда \copy, доступность файла и права доступа зависят от клиента, а не от сервера.

Путь файла, указываемый в COPY, рекомендуется всегда задавать как абсолютный, а не относительный. Это обязательное условие для команды COPY TO, но COPY FROM позволяет прочитать файл, заданный и относительным путём. Такой путь будет интерпретироваться относительно рабочего каталога серверного процесса (обычно это каталог данных кластера), а не рабочего каталога клиента.

COPY FROM вызывает все триггеры и обрабатывает все ограничения-проверки в целевой таблице. Однако правила при загрузке данных не вызываются.

Входные данные интерпретируются согласно кодировке, заданной параметром ENCODING, или текущей кодировке клиента, а выходные кодируются в кодировке ENCODING или текущей кодировке клиента, даже если данные не проходят через клиента, а считываются или записываются в файл непосредственно сервером.

COPY прекращает операцию при первой ошибке. Это не должно приводить к проблемам в случае с COPY TO, но после COPY FROM в целевой таблице остаются ранее полученные строки. Эти строки не будут видимыми и доступными, но будут занимать место на диске. Если сбой происходит при копировании большого объёма данных, это может приводить к значительным потерям дискового пространства. При желании вернуть потерянный объём, это можно сделать с помощью команды VACUUM.

Форматы файлов:

Текстовый формат

Когда применяется формат text, читаемые или записываемые данные представляют собой текстовый файл, строка в котором соответствует строке таблицы. Столбцы в строке разделяются символом-разделителем. Значения самих столбцов — текстовые строки, выдаваемые функцией вывода, либо воспринимаемые функцией ввода, соответствующей типу данных столбца. Заданный маркер NULL выводится и считывается вместо столбцов со значением NULL. COPY FROM выдаёт ошибку, если в любой из строк во входном файле оказывается больше или меньше столбцов, чем ожидается.

Конец данных может обозначаться одной строкой, содержащей только обратную косую и точку (\.). Маркер конца данных не требуется при чтении из файла, так как его роль вполне выполняет конец файла.

Символы обратной косой черты (\) в данных COPY позволяют экранировать символы данных, которые без них считались бы разделителями строк или столбцов. В частности, предваряться обратной косой должны следующие символы, когда они оказываются в значении столбца: сама обратная косая черта, перевод строки, возврат каретки и текущий разделитель.

Команда COPY FROM распознаёт следующие спецпоследовательности: \b, \f, \n, \r, \t, \v.

COPY TO никогда не выводит спецпоследовательности с восьмеричными или шестнадцатеричными кодами, однако выводит другие вышеперечисленные спецпоследовательности вместо управляющих символов.

Любой другой символ после обратной косой, будет представлять себя.

В приложениях, генерирующих данные для COPY, настоятельно рекомендуется преобразовать символы новой строки и возврата каретки в последовательности \n и \r, соответственно.

Все последовательности с обратной косой чертой обрабатываются после преобразования кодировки. Байты, заданные в таких последовательностях восьмеричными или шестнадцатеричными цифрами, должны представлять допустимые символы в кодировке базы данных.

COPY TO завершает каждую строку символом новой строки («\n»). Серверы, работающие в Microsoft Windows, вместо этого выводят символы возврат каретки/новая строка («\r\n»), но только при выводе COPY в файл на сервере.

CSV

Этот формат применяется для импорта и экспорта данных в виде списка значений, разделённых запятыми (CSV), с которым могут работать многие другие программы, например электронные таблицы.

В формате CSV отсутствует стандартный способ отличить значение NULL от пустой строки. Команда COPY решает это с помощью кавычек. Значение NULL выводится в виде строки, задаваемой параметром NULL, и не заключается в кавычки, тогда как значение не NULL, со строкой, задаваемой параметром NULL, заключается. Например, с параметрами по умолчанию NULL записывается в виде пустой строки без кавычек, тогда как пустая строка записывается в двойных кавычках (""). При чтении значений действуют похожие правила.

Двоичный формат:

При выборе формата binary все данные сохраняются/считываются в двоичном, а не текстовом виде. Иногда этот формат обрабатывается быстрее, чем текстовый и CSV. Кроме того, двоичный формат сильно зависит от типов данных; например, он не позволяет вывести данные из столбца smallint, а затем прочитать их в столбец integer, хотя с текстовым форматом это вполне возможно.

**Лекция № 8**

**Разрешения доступа**

Структурная декомпозиция КМ БД отношение внешних моделей порождает группы доступа.

ВМ определяется множеством запросов образованных эту модель. ВМ-и представляют обобщенных пользователей названных текущих СУБД ролями.

Главным и единственным объектом реляционной базы данных является отношения или таблицы. Они могут быть пустыми, то есть не содержать записей.

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, USAGE, SET и ALTER SYSTEM. Набор прав, применимых к определённому объекту, зависит от типа объекта (таблица, функция и т. д.). Более подробно назначение этих прав описывается ниже.

Право изменять или удалять объект является неотъемлемым правом владельца объекта, его нельзя лишиться или передать другому.

Объекту можно назначить нового владельца с помощью команды ALTER для соответствующего типа объекта, например:

ALTER TABLE имя\_таблицы OWNER TO новый\_владелец;

В pg 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, табличных пространств или параметров конфигурации 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.

Для каждого типа SQL-объекта показаны относящиеся к нему права, с использованием приведённых выше сокращений. Также в ней для каждого типа приведена команда psql, которая позволяет узнать, какие права назначены для объекта этого типа.

| **Тип объекта** | **Все права** | **Права**PUBLIC**по умолчанию** | **Команда psql** |
| --- | --- | --- | --- |
| DATABASE | CTc | Tc | \l |
| DOMAIN | U | U | \dD+ |
| FUNCTION или PROCEDURE | X | X | \df+ |
| FOREIGN DATA WRAPPER | U | нет | \dew+ |
| FOREIGN SERVER | U | нет | \des+ |
| LANGUAGE | U | U | \dL+ |
| LARGE OBJECT | rw | нет | \dl+ |
| PARAMETER | sA | нет | \dconfig+ |
| SCHEMA | UC | нет | \dn+ |
| SEQUENCE | rwU | нет | \dp |
| TABLE (и объекты, подобные таблицам) | arwdDxt | нет | \dp |
| Столбец таблицы | arwx | нет | \dp |
| TABLESPACE | C | нет | \db+ |
| TYPE | U | U | \dT+ |

Права, назначенные для определённого объекта, выводятся в виде элементов aclitem, где каждый aclitem отражает разрешения, которые были предоставлены субъекту определённым праводателем. Например, запись calvin=r\*w/hobbes означает, что роль calvin имеет право SELECT (r) с возможностью передачи (\*), а также непередаваемое право UPDATE (w), и оба эти права даны ему ролью hobbes. Если calvin имеет для этого объекта и другие права, предоставленные ему другим праводателем, они выводятся в отдельном элементе aclitem. Пустое поле правообладателя в aclitem соответствует роли PUBLIC.

**Лекция № 9**

**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.

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).

Если роль не задана, либо задано специальное имя пользователя PUBLIC, политика применяется ко всем пользователям в данной системе. Чтобы все пользователи могли обратиться только к собственной строке в таблице users, можно применить простую политику:

CREATE POLICY user\_policy ON users

USING (user\_name = current\_user);

Это работает подобно предыдущему примеру.

Чтобы определить для строк, добавляемых в таблицу, отдельную политику, отличную от политики, ограничивающей видимые строки, можно скомбинировать несколько политик.

CREATE POLICY user\_sel\_policy ON users

FOR SELECT

USING (true);

CREATE POLICY user\_mod\_policy ON users

USING (user\_name = current\_user);

Для команды SELECT эти две политики объединяются операцией OR, так что в итоге это позволяет выбирать все строки. Для команд других типов применяется только вторая политика, и эффект тот же, что и раньше.

На проверки ссылочной целостности, например, на ограничения уникальности и внешние ключи, защита строк никогда не распространяется, чтобы не нарушалась целостность данных. Поэтому организацию и политики защиты на уровне строк необходимо тщательно прорабатывать, чтобы не возникли «скрытые каналы» утечки информации через эти проверки.

В некоторых случаях важно, чтобы защита на уровне строк, наоборот, не действовала. Например, резервное копирование может оказаться провальным, если механизм защиты на уровне строк молча не даст скопировать какие-либо строки. В таком случае вы можете установить для параметра конфигурации row\_security значение off. Это само по себе не отключит защиту строк; при этом просто будет выдана ошибка, если результаты запроса отфильтруются политикой, с тем чтобы можно было изучить причину ошибки и устранить её.

В приведённых выше примерах выражения политики учитывали только текущие значения в запрашиваемой или изменяемой строке. Это самый простой и наиболее эффективный по скорости вариант; по возможности реализацию защиты строк следует проектировать именно так. Если же для принятия решения о доступе необходимо обращаться к другим строкам или другим таблицам, это можно осуществить, применяя в выражениях политик вложенные SELECT или функции, содержащие SELECT. Однако учтите, что при такой реализации возможны условия гонки, что чревато утечкой информации, если не принять меры предосторожности. Например, рассмотрим следующую конструкцию таблиц:

-- определение групп привилегий

CREATE TABLE groups (group\_id int PRIMARY KEY,

group\_name text NOT NULL);

INSERT INTO groups VALUES

(1, 'low'),

(2, 'medium'),

(5, 'high');

GRANT ALL ON groups TO alice; -- alice является администратором

GRANT SELECT ON groups TO public;

-- определение уровней привилегий для пользователей

CREATE TABLE users (user\_name text PRIMARY KEY,

group\_id int NOT NULL REFERENCES groups);

INSERT INTO users VALUES

('alice', 5),

('bob', 2),

('mallory', 2);

GRANT ALL ON users TO alice;

GRANT SELECT ON users TO public;

-- таблица, содержащая защищаемую информацию

CREATE TABLE information (info text,

group\_id int NOT NULL REFERENCES groups);

INSERT INTO information VALUES

('barely secret', 1),

('slightly secret', 2),

('very secret', 5);

ALTER TABLE information ENABLE ROW LEVEL SECURITY;

-- строка должна быть доступна для чтения/изменения пользователям с group\_id,

-- большим или равным group\_id данной строки

CREATE POLICY fp\_s ON information FOR SELECT

USING (group\_id <= (SELECT group\_id FROM users WHERE user\_name = current\_user));

CREATE POLICY fp\_u ON information FOR UPDATE

USING (group\_id <= (SELECT group\_id FROM users WHERE user\_name = current\_user));

-- мы защищаем таблицу с информацией, полагаясь только на RLS

GRANT ALL ON information TO public;

BEGIN;

UPDATE users SET group\_id = 1 WHERE user\_name = 'mallory';

UPDATE information SET info = 'secret from mallory' WHERE group\_id = 2;

COMMIT;

Здесь возможно условие гонки. Если Мэллори (роль mallory) параллельно выполняет, скажем:

SELECT \* FROM information WHERE group\_id = 2 FOR UPDATE;

и её транзакция в режиме READ COMMITTED, она сможет увидеть «secret from mallory». Это произойдёт, если её транзакция дойдёт до строки information сразу после того, как эту строку изменит Алиса (роль alice). Она заблокируется, ожидая фиксирования транзакции Алисы, а затем прочитает изменённое содержимое строки благодаря предложению FOR UPDATE. Однако при этом изменённое содержимое users не будет прочитано неявным запросом SELECT, так как этот вложенный SELECT выполняется без указания FOR UPDATE; вместо этого строка users читается из снимка, полученного в начале запроса. Таким образом, выражение политики проверяет старое значение уровня привилегий пользователя mallory и позволяет ей видеть изменённую строку.

Обойти эту проблему можно несколькими способами. Первое простое решение заключается в использовании SELECT ... FOR SHARE во вложенных запросах SELECT в политиках защиты строк. Однако для этого потребуется давать затронутым пользователям право UPDATE в целевой таблице (здесь users), что может быть нежелательно. (Хотя можно применить ещё одну политику защиты строк, чтобы они не могли практически воспользоваться этим правилом; либо поместить вложенный SELECT в функцию, определяющую контекст безопасности.) Кроме этого, активное использование блокировок строк в целевой таблице может повлечь проблемы с производительностью, особенно при частых изменениях. Другое решение, практичное, если целевая таблица изменяется нечасто, заключается в блокировке целевой таблицы в режиме ACCESS EXCLUSIVE при изменении, чтобы никакие параллельные транзакции не видели старые значения строк. Либо можно просто дождаться завершения всех параллельных транзакций после изменения в целевой таблице, прежде чем вносить изменения, рассчитанные на новые условия безопасности.

**Лекция № 10**

**Роли базы данных**

Роли можно рассматривать как пользователя бд или группу пользователей.

Роли могут владеть объектами бд и выдавать другим ролям разрешения на доступ. Можно предоставить одной роли членство другой, тогда она может использовать права других ролей. Концепция ролей включает в себя концепцию пользователей. Любая роль может использоваться в качестве пользователя, группы или того и другого. Роли бд концептуально отличаются от пользователей ОС. Роли бд являются глобальными для всей установки кластера базы, а не отдельной базы. Для создания, удаления и изменения роли используются команды CREATE, DROP, ALTER (ROLE имя).

Для удобства поставляются программы createuser и dropuser , которые являются обёртками для этих команд SQL и вызываются из командной строки оболочки ОС.

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;

Статус суперпользователя

Суперпользователь базы данных обходит все проверки прав доступа, за исключением права на вход в систему. Это опасная привилегия и она не должна использоваться небрежно. Лучше всего выполнять большую часть работы не как суперпользователь. Для создания нового суперпользователя используется CREATE ROLE имя SUPERUSER. Это нужно выполнить из под роли, которая также является суперпользователем.

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 изменять и удалять роли, которые были назначены пользователю с правами CREATEROLE и ADMIN OPTION. Такое назначение ролей происходит автоматически, когда пользователь с правом CREATEROLE, не являющийся суперпользователем, создаёт новую роль. Поэтому по умолчанию пользователь с правом CREATEROLE может изменять и удалять созданные им роли. Изменение роли включает большинство действий команды ALTER ROLE, например смену пароля, а также действия команд COMMENT и SECURITY LABEL в отношении ролей.

Однако, имея право CREATEROLE, нельзя ни создавать, ни каким-либо образом влиять на роли SUPERUSER. Кроме того, с правом CREATEROLE нельзя ни создавать пользователей REPLICATION, ни предоставлять или отзывать право REPLICATION, ни изменять свойства ролей таких пользователей. Тем не менее, это право позволяет использовать команды ALTER ROLE ... SET и ALTER ROLE ... RENAME в отношении ролей REPLICATION, а также COMMENT ON ROLE, SECURITY LABEL ON ROLE и DROP ROLE. Наконец, право CREATEROLE не даёт возможности предоставлять или отзывать право BYPASSRLS.

Запуск репликации

Роль должна иметь явное разрешение на запуск потоковой репликации. 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'.

Наследование прав

По умолчанию роль наследует права ролей, членом которых она является. Однако можно создать роль, которая не будет наследовать права по умолчанию. Для этого выполните CREATE ROLE имя NOINHERIT. Как вариант, атрибуты наследования отдельных прав можно переопределить, указав WITH INHERIT TRUE или WITH INHERIT FALSE.

Игнорирование защиты на уровне строк

Разрешение обходить все политики защиты на уровне строк (RLS) нужно давать роли явно. Создать роль с таким разрешением можно, выполнив CREATE ROLE имя BYPASSRLS от имени суперпользователя.

Ограничение соединений

Для роли можно ограничить число одновременных соединений. -1 (по умолчанию) означает отсутствие ограничения.

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

CREATE ROLE имя;

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.

Члены групповой роли могут использовать её права двумя способами. Во-первых, 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. Во-вторых, роли, имеющие атрибут INHERIT, автоматически используют права всех ролей, членами которых они являются, в том числе и унаследованные этими ролями права.

CREATE ROLE joe LOGIN INHERIT;

CREATE ROLE admin NOINHERIT;

CREATE ROLE wheel NOINHERIT;

GRANT admin TO joe;

GRANT wheel TO admin;

После подключения с ролью joe сеанс базы данных будет использовать права, выданные напрямую joe, и права, выданные роли admin, так как joe «наследует» права admin. Однако права, выданные wheel, не будут доступны, потому что, хотя joe неявно и является членом wheel, это членство получено через роль admin, которая имеет атрибут NOINHERIT. После выполнения команды:

SET ROLE admin;

сеанс будет использовать только права, назначенные admin, а права, назначенные роли joe, не будут доступны. После выполнения команды:

SET ROLE wheel;

сеанс будет использовать только права, выданные wheel, а права joe и admin не будут доступны. Начальный набор прав можно получить любой из команд:

SET ROLE joe;

SET ROLE NONE;

RESET ROLE;

Особенности удаления ролей:

Так как роли могут владеть объектами баз данных и иметь права доступа к объектам других, удаление роли не сводится к немедленному действию DROP ROLE. Сначала должны быть удалены и переданы другим владельцами 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, принадлежащими удаляемой роли, можно применить команду REASSIGN OWNED. Так как REASSIGN OWNED не может обращаться к объектам в других базах данных, её необходимо выполнить в каждой базе, которая содержит объекты, принадлежащие этой роли.