Лабораторная работа №1

Ознакомление с основами работы в среде PostgreSQL

Цель работы: Изучить базовые операции по работе с базой данных. Изучить синтаксис команд. Приобрести навыки создания баз данных, создания, заполнения и модификации таблиц в PostgreSQL.

Задание. Общая часть: Создать и заполнить базу данных своего варианта в PostgreSQL. Таблицы (минимум по 10 записей в каждой) связать между собой полями идентификаторов. Предусмотреть наличие связей типа: «*один ко многим»*. С помощью команд интерактивного терминала *psql* просмотреть структуру базы данных, структуру таблиц, просмотреть данные в них, изменить структуру таблиц, добавить столбцы, добавить данные, создать столбцы с пользовательскими типами данных. Предусмотреть наличие таблиц-справочников и таблиц, использующих справочники.

Вариант 1: Создать и заполнить базу данных игроков хоккейной команды, состоящую из пяти таблиц. Первая таблица должна содержать поля: идентификатор игрока, имя, фамилия, дата рождения, рост, вес, идентификатор города проживания игрока и другие поля при необходимости. Вторая: игровая позиция (вратарь, защитник, нападающий), игровой номер, размер коньков, хват клюшки (левый или правый) и другие поля при необходимости. Третья: оклад, премия (выплата за каждое очко по системе «гол+пас»), штраф (за каждую минуту штрафного времени в игре, начиная с 30% игрового времени), идентификатор стадиона и другие поля при необходимости. Четвертая: справочник городов. Пятая таблица: справочник стадионов. На основании созданных таблиц создать таблицу, содержащую, например, поля: фамилия игрока, наименование города проживания игрока, игровая позиция, игровой номер, премия, наименование стадиона.

Вариант 2: Создать и заполнить базу данных для учета работы продуктового магазина, состоящую из пяти таблиц. Первая таблица должна содержать поля: идентификатор товара, наименование товара,

идентификатор типа товара, идентификатор производителя товара, количество товара на складе, стоимость покупки (за единицу измерения) и другие поля при необходимости. Вторая: идентификатор производителя, наименование производителя, идентификатор города, где находится производитель, и другие поля при необходимости. Третья: цена продажи, проданное количество товара и другие поля при необходимости. Четвертая: справочник городов. Пятая: справочник типов товаров (например, бакалея, молочные продукты, кондитерские изделия). На основании созданных таблиц создать таблицу, содержащую, например, поля: наименование товара, тип товара, наименование производителя, наименование города, где находится производитель, цена продажи.

Вариант 3: Создать и заполнить базу данных для службы такси, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор водителя, имя, фамилия, дата рождения, идентификатор автомобиля водителя, идентификатор города проживания водителя, водительский стаж и другие поля при необходимости. Вторая: идентификатор автомобиля, марка автомобиля, тариф за 1 км и другие поля при необходимости. Третья: идентификатор заказа, дата заказа, расстояние проезда, стоимость заказа и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: фамилия водителя, город проживания водителя, марка автомобиля, тариф, дата заказа, расстояние, стоимость заказа.

Вариант 4: Создать и заполнить базу данных для обработки данных по работе книжной лавки, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор жанра, наименование жанра и другие поля при необходимости. Вторая: идентификатор книги, название книги, автор книги, тираж, идентификатор жанра книги, идентификатор издательства, идентификатор города проживания автора и другие поля при необходимости. Третья: идентификатор издательства, наименование

издателя, адрес издательства и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: название книги, автор книги, город проживания автора, жанр, издательство.

Вариант 5: Создать и заполнить базу данных для осуществления учета работы автосалона, состоящую из пяти таблиц. Первая таблица должна содержать поля: идентификатор автомобиля, наименование автомобиля, идентификатор производителя, год выпуска, пробег, цена и другие поля при необходимости. Вторая: руль (правый или левый), привод (передний, задний или полный), трансмиссия (механическая, автомат), тип двигателя (вид топлива, способ впрыска топлива) и другие поля при необходимости. Третья: марка автомобиля (Toyota, Honda, ВАЗ и т.п.), идентификатор страны производителя автомобиля, тип кузова (седан, универсал, микроавтобус и т.д.) и другие поля при необходимости. Четвертая: справочник стран. Пятая: идентификатор производителя, наименование производителя и другие поля при необходимости. На основании созданных таблиц создать таблицу, содержащую, например, поля: марка автомобиля, наименование страны, наименование производителя, тип кузова, руль, привод, тип двигателя, стоимость.

Вариант 6: Создать и заполнить базу данных учета поступления товара на овощную базу, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор поставки, дата поставки, вес, цена за кг и другие поля при необходимости. Вторая: идентификатор продукта, наименование продукта, сорт и другие поля при необходимости. Третья: идентификатор поставщика, наименование поставщика, идентификатор города, где находится поставщик и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: наименование продукта, поставщик, наименование города, где находится поставщик, вес, цена.

Вариант 7: Создать и заполнить базу данных для учета работы типографии, состоящую из пяти таблиц. Первая таблица должна содержать поля: идентификатор продукции, идентификатор типа продукции (например, этикетка, коробка, блокнот и т.п.), идентификатор материала, количество, стоимость заказа и другие поля при необходимости. Вторая: идентификатор заказчика, наименование заказчика, идентификатор города, где находится заказчик, способ расчета (наличный, безналичный) и другие поля при необходимости. Третья: идентификатор материала, тип используемого материала, количество материала, стоимость расходных материалов и другие поля при необходимости. Четвертая: справочник городов. Пятая: справочник типов продукции. На основании созданных таблиц создать таблицу, содержащую, например, поля: наименование заказчика, город, где находится заказчик, тип продукции, тип используемого материала, стоимость заказа.

Вариант 8: Создать и заполнить базу данных жилищной управляющей компании, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор жильца, дата ремонта, идентификатор ремонта, стоимость ремонта, информация об оплате ремонта (оплачен полностью, не оплачен, оплачен частично) и другие поля при необходимости. Вторая: идентификатор ремонта, вид ремонта, идентификатор мастера, длительность ремонта и другие поля при необходимости. Третья: идентификатор мастера, имя, фамилия, дата рождения, идентификатор города проживания и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: фамилия жильца, наименование вида ремонта, фамилия мастера, наименование города проживания мастера, длительность ремонта, дата ремонта, стоимость ремонта.

Вариант 9: Создать и заполнить базу данных гарантийной мастерской по ремонту, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор заказа, дата заказа, идентификатор предмета для ремонта, идентификатор мастера, и другие поля при необходимости.

Вторая: идентификатор предмета, вид ремонтируемого предмета, стоимость ремонта и другие поля при необходимости. Третья: идентификатор мастера, имя, фамилия, дата рождения, идентификатор города проживания мастера и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: дата заказа, фамилия мастера, наименование города проживания мастера, вид ремонтируемого предмета, стоимость.

Вариант 10: Создать и заполнить базу данных туристического агентства, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор заказа, дата заказа, номер заказа, идентификатор тура, идентификатор руководителя тура, количество участников и другие поля при необходимости. Вторая: идентификатор тура, вид тура (автобусный, железнодорожный, авиа), стоимость тура, дата начала тура, идентификатор города путешествия и другие поля при необходимости. Третья: идентификатор руководителя тура, фамилия руководителя, дата рождения и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: дата заказа, фамилия руководителя тура, вид тура, наименование города путешествия, дата начала тура, стоимость тура.

Содержание отчета

Отчет должен содержать титульный лист, цель работы, задание, коды команд на каждом этапе выполнения работы, результаты выполнения команд (скриншоты), выводы и анализ результатов работы.

СОЗДАНИЕ ТАБЛИЦ В СУБД PostgreSQL

Создание пользователей и баз данных

Для создания нового пользователя в PostgreSQL используется команда *CREATE USER*. Команда *CREATE USER* имеет всего один обязательный параметр - имя нового пользователя. Ей также можно передать множество других параметров, в том числе пароль, системный идентификатор, группу и права, назначаемые новому пользователю.

Полный синтаксис команды *CREATE USER* выглядит так:

CREATE USER пользователь

[WITH [SYSID uid]

[PASSWORD 'пароль']]

[CREATEDB | NOCREATEDB]

[CREATEUSER | NOCREATEUSER]

[IN GROUP группа [...]]

[VALID UNTIL 'срок']

Имя создаваемого пользователя определяется параметром *пользователь*. В системе не допускается присутствие двух пользователей с одинаковыми именами.

За ключевым словом *WITH* следуют секции *SYSID* и/или *PASSWORD*.

Все остальные необязательные секции могут следовать в произвольном порядке (ключевое слово *WITH* для них не обязательно). Ниже перечислены все необязательные ключевые слова.

SYSID uid. Пользователю назначается системный идентификатор *uid*. Если значение не указано, выбирается некоторое уникальное число.

PASSWORD 'пароль'. Новому пользователю назначается заданный пароль. Если значение не указано, по умолчанию используется пароль NULL.

CREATEDB | NOCREATEDB. Ключевое слово *CREATEDB* предоставляет новому пользователю право создания баз данных, а также

право уничтожения принадлежащих ему баз данных. Ключевое слово *NOCREATEDB* явно указывает на отсутствие такого права (используется по умолчанию).

CREATEUSER | NOCREATEUSER. Ключевое слово *CREATEUSER* предоставляет пользователю право создания пользователей, наделяя его тем самым правами суперпользователя. Пользователь с правом создания других пользователей обладает всеми правами во всех базах данных (включая право создания баз данных, даже если было указано ключевое слово *NOCREATEDB*). Ключевое слово *NOCREATEUSER* явно указывает на отсутствие права создания новых пользователей.

IN GROUP группа [....]. Новый пользователь включается в группу с заданным именем. Допускается перечисление нескольких групп через запятую. Чтобы создание пользователя прошло успешно, перечисленные группы должны существовать.

VALID UNTIL 'срок'. Пароль пользователя становится недействительным в указанный момент, который задается в одном из поддерживаемых форматов времени. По истечении срока действия пароль необходимо сменить.

VALID UNTIL 'infinity'. Срок действия пароля не ограничивается.

При отсутствии ключевых слов *CREATEDB* и *CREATEUSER* создастся «обычный» пользователь, не обладающий особыми привилегиями. Он не может ни создавать, ни уничтожать базы данных и других пользователей. Обычные пользователи могут подключаться к базам данных PostgreSQL, но при этом они могут выполнять лишь те команды, выполнение которых им было разрешено.

Для создания базы данных необходимо воспользоваться командой *CREATE DATABASE*.

Синтаксис:

CREATE DATABASE база_данных

[WITH [LOCATION = {'каталог' | DEFAULT}]

[TEMPLATE - шаблон | DEFAULT]

[ENCODING - имя_кодировки | номер_кодировки | DEFAULT]]

Параметры:

база_данных. Имя создаваемой базы данных.

каталог. Каталог, в котором хранится база данных. С ключевым словом *DEFAULT* база сохраняется в каталоге по умолчанию, задаваемом переменной среды PGDATA (пли ключом -D, переданным postmaster).

шаблон. Имя шаблона, на основании которого создается новая база данных. Ключевое слово DEFAULT означает шаблон по умолчанию (обычно template1).

имя_кодировки | номер_кодировки. Расширенная кодировка, используемая базой данных. Задается в виде строкового литерала или в виде целочисленного номера, определяющего тип кодировки. Список типов расширенных кодировок в PostgreSQL приведен в приложении А. Список команд для работы с типом *date* приведен в приложении Б

DEFAULT. Ключевое слово *DEFAULT* явно задает кодировку по умолчанию (используемую при отсутствии параметра *ENCODING*).

Для входа в базу под созданным пользователем необходимо выйти из текущей учетной записи и набрать команду:

psql –U имя база

Здесь *имя* - имя созданного пользователя, *база* – имя базы данных, к которой происходит подключение. Если база не создана, подключиться к базе template1, создать новую базу и переподключиться.

Команды для работы с таблицами.
Создание, заполнение, модификация

Таблицы создаются командой *CREATE TABLE*. Эта команда создает пустую таблицу - таблицу без строк. Значения вводятся с помощью DML команды *INSERT*. Команда *CREATE TABLE,* в основном, определяет имя

таблицы в виде описания набора имен столбцов указанных в определенном порядке. Она также определяет типы данных (приложение В) и размеры столбцов. Каждая таблица должна иметь, по крайней мере, один столбец.

Синтаксис команды *CREATE TABLE*:

CREATE TABLE <table-name>

(<column name> <data type> [(<size>)],

<column name> <data type> [(<size>)] ...);

Так как пробелы используются для разделения частей команды SQL, они не могут быть частью имени таблицы. Подчеркивание (_) - обычно используется для разделения слов в именах таблиц.

Значение аргумента размера зависит от типа данных. Если вы его не указываете, ваша система сама будет назначать значение автоматически. Для числовых значений это - лучший выход, потому что в этом случае все ваши поля такого типа получат один и тот же размер, что освобождает вас от проблем их общей совместимости. Кроме того, использование аргумента размера с некоторыми числовыми наборами не совсем простой вопрос. Если вам нужно хранить большие числа, вам, несомненно, понадобятся гарантии, что поля достаточно велики, чтобы вместить их.

Один тип данных, для которого вы, в основном, должны назначать размер - CHAR. Аргумент размера - это целое число, которое определяет максимальное число символов, которое может вместить поле. Фактически, число символов поля может быть от нуля (если поле - NULL) до этого числа. По умолчанию, аргумент размера = 1, что означает, что поле может содержать только одну букву.

Таблицы принадлежат пользователю, который их создал, и имена всех таблиц принадлежащих данному пользователю должны отличаться друга от друга, как и имена всех столбцов внутри данной таблицы. Отдельные таблицы могут использовать одинаковые имена столбцов, даже если они принадлежат одному и тому же пользователю. Пользователи, не являющиеся

владельцами таблиц, могут ссылаться к этим таблицам с помощью имени владельца этих таблиц сопровождаемого точкой[3].

Эта команда будет создавать таблицу списка самолетов:

CREATE TABLE spisok

(Sname char(20), - *Название самолета*

country char (20), - *Страна-изготовитель*

Skor int, - *Крейсерская скорость*

vm int, - *Вместительность*

st int); - *Стоимость*

Порядок столбцов в таблице определяется порядком, в котором они указаны. Имя столбца не должно разделяться при переносе строки (что сделано для удобочитаемости), но отделяется запятыми.

Если вам не нужна далее созданная вами таблица или если вы планируете пересоздать её с другим набором полей, вы можете удалить ее, используя команду *DROP TABLE tablename*;

Для помещения записей в таблицу используется оператор *INSERT*:

INSERT INTO spisok VALUES (‘IL_96’, ‘Russia’, 870, 436, 10800000);

Обратите внимание, что все типы данных, используемые в команде, имеют соответствующие форматы. Константы, которые не являются простыми числовыми значениями обычно должны быть заключены в одинарные кавычки ('), как показано в примере.

Синтаксис, используемый здесь, требует, чтобы вы помнили порядок полей. Альтернативная форма записи позволяет вам перечислять поля явно:

INSERT INTO spisok (Sname, country, Skor, vm, st)

VALUES (‘IL_96’, ‘Russia’, 870, 436, 10800000);

Вы можете указать поля в другом порядке, если захотите это или даже опустить некоторые поля.

В большинстве современных РСУБД предусмотрена возможность модификации таблиц командой *ALTER TABLE*.

Для создания нового поля в команду *ALTER TABLE* включается секция *ADD COLUMN*.

Синтаксис команды *ALTER TABLE* с секцией *ADD COLUMN*:

ALTER TABLE таблица ADD [COLUMN] имя_поля тип_поля

• *таблица* - имя таблицы, в которой создается новое поле;

• *имя_поля* - имя создаваемого поля;

• *тип_поля* - тип создаваемого поля.

Ключевое слово *COLUMN* не является обязательным и включается в команду лишь для наглядности.

Несмотря на возможность включения новых полей в существующую таблицу, следует помнить, что в PostgreSQL не поддерживается удаление полей[2].

Существует два относительно простых способа реструктуризации существующих таблиц. Первый способ основан на использовании команды *CREATE TABLE AS*, а во втором способе команда *CREATE TABLE* объединяется с командой *INSERT INTO*.

Фактически, оба способа сводятся к созданию новой таблицы требуемой структуры, заполнению ее данными из существующей таблицы и переименованию. Таким образом, новая таблица занимает место старой.

Распространенная методика реструктуризации таблиц основана на использовании команды *CREATE TABLE* с секцией *AS* в сочетании с запросом SQL. Команда создает временную таблицу на основании существующей таблицы, после чего временная таблица переименовывается. Физическое создание новой таблицы может сопровождаться удалением полей и изменением порядка их следования с одновременным заполнением данными из исходной таблицы.

В приведенном ниже описании синтаксиса этой усеченной версии команды *CREATE TABLE запрос* представляет собой команду *SELECT* для выборки данных, переносимых в новую таблицу. Типы данных всех

создаваемых полей определяются типами данных соответствующих полей, выбранных в результате выполнения запроса.

CREATE [TEMPORARY | TEMP] TABLE таблица [(имя_поля [....])]

AS запрос

Преимущество такого подхода заключается в том, что создание таблицы и заполнение ее данными происходит в одной команде SQL. Самый заметный недостаток - отсутствие полноценных возможностей для установки ограничений в созданной таблице. После того как таблица создана, в нее можно добавлять только ограничения внешних ключей и проверки. После создания новой таблицы старую таблицу можно переименовать (или уничтожить) и присвоить новой таблице имя старой.

После того как записи сохранены в базе данных, вы можете обновить их поля командой SQL *UPDATE*. Новые значения полей задаются в виде констант, идентификаторов других баз данных или выражений. Допускается обновление как поля в целом, так и подмножества его значений в соответствии с заданными условиями.

Синтаксис команды *UPDATE*:

UPDATE [ONLY] таблица SET поле = выражение [....]

[FROM источник] [WHERE условие] UPDATE [ONLY] таблица.

Ключевое слово *ONLY* означает, что обновляется только заданная таблица, но не ее производные таблицы. Применяется лишь в том случае, если таблица использовалась в качестве базовой при наследовании.

SET поле = выражение [....]. Обязательная секция *SET* содержит перечисленные через запятую условия, определяющие новые значения обновляемых полей. Условия всегда имеют форму *поле = выражение*, где *поле* - имя обновляемого поля (не допускаются ни синонимы, ни точечная запись), а *выражение* описывает новое значение поля.

FROM источник. Секция *FROM* принадлежит к числу нестандартных расширений PostgreSQL и позволяет обновлять поля значениями, взятыми из других наборов.

WHERE условие. В секции *WHERE* задается критерий, по которому в таблице выбираются обновляемые записи. Если секция *WHERE* отсутствует, поле обновляется во всех записях. По аналогии с командой *SELECT* может использоваться для уточнения выборки из источников, перечисленных в секции *FROM*.

При отсутствии секции *WHERE* команда *UPDATE* обновляет заданное поле во всех записях таблицы. Обычно в этой ситуации новое значение поля задается выражением, а не константой. Выражение, указанное в секции *SET*, вычисляется заново для каждой записи, а новое значение поля определяется динамически.

Удаление записей из таблиц производится стандартной командой SQL *DELETE*. Вызов *DELETE* приводит к необратимым последствиям (исключение составляют тщательно спланированные транзакционные блоки), поэтому удаление данных из базы требует крайней осторожности.

Команда удаления одной или нескольких записей из базы имеет следующий синтаксис:

DELETE FROM [ONLY] таблица [WHERE условие]

WHERE условие. В секции *WHERE* задается критерий, по которому в таблице выбираются удаляемые записи. При отсутствии секции *WHERE* из таблицы удаляются все записи.

Секция *WHERE* почти всегда присутствует в команде *DELETE*. В ней определяются условия отбора удаляемых записей, выраженные в такой же синтаксической форме, как и при использовании команды *SELECT*.

Перед выполнением команды *DELETE* рекомендуется выполнить команду *SELECT* с соответствующей секцией *WHERE* и просмотреть удаляемые данные перед их фактическим уничтожением[2].

Посредством команды интерактивного терминала *\d таблица* можно просмотреть структуру таблицы с именем *таблица,* а также список всех индексов и ограничений таблицы. Список команд интерактивного терминала для справки и получения различной информации приведен в приложении Г.

В PostgreSQL поддерживается механизм создания объектно-реляционных связей, называемый наследованием. Таблица может наследовать некоторые атрибуты своих полей от одной или нескольких других таблиц, что приводит к созданию отношений типа «предок-потомок». В результате производные таблицы («потомки») обладают теми же полями и ограничениями, что и их базовые таблицы («предки»), а также дополняются собственными полями.

При составлении запроса к базовой таблице можно потребовать, чтобы запрос произвел выборку только из самой таблицы или же просмотрел как таблицу, так и ее производные таблицы. С другой стороны, в результаты запроса к производной таблице никогда не включаются записи из базовой таблицы.

Производная таблица создается командой SQL *CREATE TABLE*, в которую включается секция *INHERITS*. Секция состоит из ключевого слова *INHERITS* и имени базовой таблицы (или нескольких таблиц).

Часть команды *CREATE TABLE*, относящаяся к наследованию, выглядит так:

CREATE TABLE производная_таблица определение

INHERITS (базовая_таблица [....])

В этом определении *производная таблица* - имя создаваемой таблицы, *определение* — полное определение таблицы со всеми стандартными секциями команды *CREATE TABLE*, а *базовая таблица* - таблица, структура которой наследуется новой таблицей. Дополнительные имена базовых таблиц перечисляются через запятую.

Связь общих полей базовой и производной таблиц не ограничивается чисто косметическими удобствами. В запрос к базовой таблице можно включить ключевое слово *ONLY*, которое указывает, что данные производных таблиц исключаются из результатов запроса.

Данные базовых таблиц никогда не включаются в результаты запросов к производным таблицам. Таким образом, ключевое слово *ONLY* в запросе к

производной таблице действует лишь в том случае, если у производной таблицы есть собственный потомок, из-за чего она одновременно является и производной, и базовой.

Наследование может приводить к видимому нарушению ограничений. Например, значение поля, для которого установлено ограничение уникальности, может повторяться в данных производных таблиц. Применение наследования требует осторожности, поскольку производная таблица формально не нарушает ограничений, хотя при выборке из базовой таблицы без ключевого слова *ONLY* может показаться обратное.

Процесс включения данных в базовые и производные таблицы весьма прямолинеен. Вставка в производную таблицу приводит к кажущемуся появлению данных в базовой таблице, хотя сами данные по прежнему находятся в производной таблице. Вставка данных в базовую таблицу никак не отражается на производной таблице.

Процесс модификации данных в производной таблице достаточно прост и очевиден - изменяется только содержимое производной таблицы, а все данные из базовой таблицы остаются без изменений. Как говорилось выше, данные не являются общими в обычном смысле слова, а лишь видны в других таблицах. Выборка из базовой таблицы без ключевого слова *ONLY* выведет как записи базовой таблицы, так и модифицированные записи производной таблицы.

С модификацией записей в базовых таблицах дело обстоит сложнее. Команды *UPDATE* и *DELETE* по умолчанию работают не только с записями базовой таблицы, но и с записями всех производных таблиц, подходящих по заданному критерию.

Ключевое слово *ONLY* выполняет в командах *UPDATE* и *DELETE* те же функции, что и в команде *SELECT* - оно предотвращает каскадные модификации. Согласно правилам синтаксиса SQL ключевое слово *ONLY* всегда предшествует имени производной таблицы.

Центральное место в SQL занимает команда *SELECT*, предназначенная для построения запросов и выборки данных из таблиц и представлений. Данные, возвращаемые в результате запроса, называются итоговым набором; как и таблицы, они состоят из записей и полей.

Данные итогового набора не хранятся на диске в какой-либо постоянной форме. Итоговый набор является лишь временным представлением данных, полученных в результате запроса. Структура полей итогового набора может соответствовать структуре исходной таблицы, но может и радикально отличаться от нее. Итоговые наборы даже могут содержать поля, выбранные из других таблиц.

Из-за своей особой роли в PostgreSQL команда *SELECT* также является самой сложной командой, обладающей многочисленными секциями и параметрами. Ниже приведено общее определение синтаксиса *SELECT*.

SELECT [ALL | DISTINCT [ON (выражение [...])]]
цель [AS имя] [...] [FROM источник [...]]
[[NATURAL] тип_объединения источник
[ON условие | USING (список_полей)]] [...]
[WHERE условие] [GROUP BY критерий [...]]
[HAVING условие [...]]
[{UNION | INTERSECT | EXCEPT} [ALL] подзапрос]
[ORDER BY выражение [ASC | DESC | USING оператор] [...]]
[FOR UPDATE [OF таблица [...]]]
[LIMIT {число | ALL} [OFFSET начало]]

В этом описании *источник* представляет собой имя таблицы или подзапрос. Эти общие формы имеют следующий синтаксис:

FROM {[ONLY] таблица
[[AS] синоним [(синоним_поля[...])]]
(запрос) [AS] синоним [(синоним_поля [...])]}

ALL. Необязательное ключевое слово *ALL* указывает на то, что в выборку включаются все найденные записи.

DISTINCT [ON (выражение [...])]. Секция *DISTINCT* определяет поле или выражение, значения которого должны входить в итоговый набор не более одного раза.

Цель [AS имя] [...]. В качестве *цели* обычно указывается имя поля, хотя *цель* также может быть константой, идентификатором, функцией или общим выражением. Перечисляемые *цели* разделяются запятыми, существует возможность динамического назначения имен *целей* в секции *AS*. Звездочка (*) является сокращенным обозначением всех несистемных полей, вместе с ней в списке могут присутствовать и другие *цели*.

FROM источник [...]. В секции *FROM* указывается *источник*, в котором PostgreSQL ищет заданные *цели*. В данном случае *источник* является именем таблицы или подзапроса. Допускается перечисление нескольких *источников*, разделенных запятыми (примерный аналог перекрестного запроса). Синтаксис секции *FROM* подробно описан ниже.

[NATURAL] тип_объединения источник [ON условие | USING (список_полей)]. Источники *FROM* могут группироваться в секции *JOIN* с указанием типа объединения (*INNER, FULL, OUTER, CROSS*). В зависимости от типа объединения также может потребоваться уточняющее условие или список полей.

WHERE условие. Секция *WHERE* ограничивает итоговый набор заданными критериями. Условие должно возвращать простое логическое значение (true или false), но оно может состоять из нескольких внутренних условий, объединенных логическими операторами (например, AND или OR).

GROUP BY критерий [...]. Секция *GROUP BY* обеспечивает группировку записей по заданному *критерию*. Причем *критерий* может быть простым именем поля или произвольным выражением, примененным к значениям итогового набора.

HAVING условие [...]. Секция *HAVING* похожа на секцию *WHERE*, но условие проверяется на уровне целых групп, а не отдельных записей.

{UNION | INTERSECT | EXCEPT} [ALL] подзапрос. Выполнение одной из трех операций, в которых участвуют два запроса (исходный и дополнительный); итоговые данные возвращаются в виде набора с обобщенной структурой, из которого удаляются дубликаты записей (если не было задано ключевое слово *ALL*). *UNION* - объединение (записи, присутствующие в любом из двух наборов). *INTERSECT* - пересечение (записи, присутствующие одновременно в двух наборах). *EXCEPT* - исключение (записи, присутствующие в основном наборе *SELECT*, но не входящие в подзапрос).

ORDER BY выражение. Сортировка результатов команды *SELECT* по заданному *выражению*.

[ASC | DESC | USING оператор]. Порядок сортировки, определяемой секцией *ORDER BY выражение*: по возрастанию (ASC) или по убыванию (DESC). С ключевым словом *USING* может задаваться *оператор*, определяющий порядок сортировки (например, < или >).

FOR UPDATE [OF таблица [...]]. Возможность монопольной блокировки возвращаемых записей. В транзакционных блоках *FOR UPDATE* блокирует записи указанной таблицы до завершения транзакции. Заблокированные записи не могут обновляться другими транзакциями.

LIMIT {число | ALL}. Ограничение максимального количества возвращаемых записей или возвращение всей выборки (*ALL*).

OFFSET начало. Точка отсчета записей для секции *LIMIT*. Например, если в секции *LIMIT* установлено ограничение в 100 записей, а в секции *OFFSET* -50, запрос вернет записи с номерами 50-150 (если в итоговом наборе найдется столько записей).

Ниже описаны компоненты секции *FROM*.

[ONLY] таблица. Имя таблицы, используемой в качестве источника для команды *SELECT*. Ключевое слово *ONLY* исключает из запроса записи всех таблиц-потомков.

[AS] синоним. Источникам *FROM* могут назначаться необязательные псевдонимы, упрощающие запрос. Ключевое слово *AS* является необязательным.

(запрос) [AS] синоним. В круглых скобках находится любая синтаксически правильная команда *SELECT*. Итоговый набор, созданный запросом, используется в качестве источника *FROM* так, словно выборка производится из статической таблицы. При выборке из подзапроса обязательно должен назначаться синоним.

(синоним_поля [...]). Синонимы могут назначаться не только всему источнику, но и его отдельным полям. Перечисляемые синонимы нолей разделяются запятыми и группируются в круглых скобках за синонимом источника *FROM*. Синонимы перечисляются в порядке следования полей в таблице, к которой они относятся.

Целями команды *SELECT* могут быть не только простые поля, но и произвольные выражения (включающие вызовы функций или различные операции с идентификаторами) и константы. Синтаксис команды практически не изменяется, появляется лишь одно дополнительное требование - все самостоятельные выражения, идентификаторы и константы должны разделяться запятыми. В списке разрешены произвольные комбинации разнотипных целей.

Команда *SELECT* также может использоваться для простого вычисления и вывода результатов выражений и констант. В этом случае она не содержит секции *FROM* или имен столбцов.

Для каждой цели в списке может задаваться необязательная секция *AS*, которая назначает синоним (новое произвольное имя) для каждого поля в итоговом наборе. Имена синонимов подчиняются тем же правилам, что и имена обычных идентификаторов (в частности, они могут содержать внутренние пробелы или совпадать с ключевыми словами при условии заключения их в апострофы и т. д.)

Назначение синонима не влияет на исходное поле и действует лишь в контексте итогового набора, возвращаемого запросом. Секция *AS* особенно удобна при «выборке» выражений и констант, поскольку синонимы позволяют уточнить смысл неочевидных выражений или констант.

В секции *FROM* указывается источник данных - таблица или итоговый набор. Секция может содержать несколько источников, разделенных запятыми.

Использование нескольких источников данных в PostgreSQL требует осторожности. В результате выполнения команды *SELECT* для нескольких источников без секций *WHERE* и *JOIN*, уточняющих связи между источниками, возвращается полное декартово произведение источников. Иначе говоря, итоговый набор содержит все возможные комбинации записей из всех источников. Обычно для уточнения связей между источниками, перечисленными через запятую в секции *FROM*, используется секция *WHERE*.

Для предотвращения неоднозначности в «полные» имена столбцов включается имя таблицы. При этом используется специальный синтаксис, называемый точечной записью (название связано с тем, что имя таблицы отделяется от имени поля точкой).

Точечная запись обязательна только при наличии неоднозначности между наборами данных.

Если в качестве источника используется итоговый набор, созданный подзапросом, то весь подзапрос заключается в круглые скобки. По наличию круглых скобок PostgreSQL узнает о том, что команда *SELECT* интерпретируется как подзапрос, и выполняет ее перед выполнением внешней команды *SELECT*.

Источникам данных в секции *FROM* - таблицам, подзапросам и т. д. - можно назначать синонимы в секции *AS* (по аналогии с отдельными полями). Синонимы часто используются для упрощения точечной записи. Наличие синонима для набора данных позволяет обращаться к нему при помощи

точечной записи, что делает команды SQL более компактными и наглядными.

Синонимы можно назначать не только для источников данных в секции *FROM*, но и для отдельных полей внутри этих источников. Для этого за синонимом источника данных приводится список синонимов полей, разделенный запятыми и заключенный в круглые скобки. Список синонимов полей представляет собой последовательность идентификаторов, перечисленных в порядке следования полей в структуре таблицы (слева направо).

Список синонимов полей не обязан содержать данные обо всех полях. К тем полям, для которых не были заданы синонимы, можно обращаться по обычным именам. Если единственное поле, которому требуется назначить синоним, находится после других полей, вам придется включить в список все предшествующие поля (синоним поля может совпадать с его именем). В противном случае PostgreSQL не сможет определить, какому полю назначается синоним, и решит, что речь идет о первом поле таблицы.

Необязательное ключевое слово *DISTINCT* исключает дубликаты из итогового набора. Если ключевое слово *ON* отсутствует, из результатов запроса с ключевым словом *DISTINCT* исключаются записи с повторяющимися значениями целевых полей.

Проверяются только поля, входящие в целевой список *SELECT*.

В общем случае PostgreSQL выбирает записи, исключаемые из итогового набора при наличии секции *ON*, по своему усмотрению. Если в запрос вместе с *DISTINCT* входит секция *ORDER BY*, вы можете самостоятельно задать порядок выборки полей так, чтобы нужные записи оказались в начале.

Если вместо исключения всех дубликатов достаточно сгруппировать записи с повторяющимися значениями некоторого критерия, воспользуйтесь секцией *GROUP BY*.

В секции *WHERE* задается логическое условие, которому должны удовлетворять записи итогового набора. На практике команда *SELECT* почти всегда содержит как минимум одну уточняющую секцию *WHERE*.

Секция *WHERE* может содержать несколько условий, объединенных логическими операторами (например, AND или OR) и возвращающими одно логическое значение.

Количество условий, объединяемых в секции *WHERE*, не ограничено, хотя при наличии двух и более условий обычно выполняется группировка при помощи круглых скобок, наглядно демонстрирующая логическую связь между условиями.

Примеры команд:

- создать базу данных t1

create database t1;

- подключиться к базе данных t1

\c t1

- создать таблицу m1 с полями (идентификатор, имя, идентификатор_города)

create table m1(id int, name char(15), id_city int);

- создать таблицу m2 с полями (идентификатор, город)

create table m2(id int, city char(20));

- ввод данных в таблицу m1

insert into m1 values (3, 'Ivan', 1);

- ввод данных в таблицу m2

insert into m2 values (1, 'Omsk');

- вывести все записи из таблицы m1

select * from m1;

- создать таблицу m3 на основе таблиц m1 и m2

create table m3 as select m1.name, m2.city from m1 inner join m2 on m1.id_city=m2.id;

- вывести все записи из таблицы m3

select * from m3;

Выгрузка и загрузка базы данных

Архивация и восстановление данных занимают важное место в работе любого администратора базы данных. Ни одна система не застрахована от сбоев жесткого диска, неосторожности пользователей или других потенциальных несчастий, приводящих к порче данных PostgreSQL.

Приложение *pg_dump* запускается в режиме командной строки и строит серию команд SQL. Выполнение этих команд в указанном порядке позволяет полностью воссоздать базу данных.

Синтаксис приложения *pg_dump*:

pg_dump [параметры] база_данных

Параметр *база данных* определяет имя базы данных, для которой генерируются команды SQL. Строка параметров имеет такой же формат, как у других утилит управления базами данных (например, *createdb*). В ней чаще всего передается ключ -f для определения файла, в котором сохраняются сгенерированные команды.

Если флаг -f не указан, сгенерированные команды SQL вместо записи в файл выводятся в поток stdout.

Ниже приведен полный список ключей приложения *pg_dump*.

-a, --dataonly. Приложение генерирует только команды SQL *COPY* и *INSERT* (в зависимости от того, установлен ли ключ -d). В результате архивируются только данные, хранящиеся в базе, но не объекты базы данных. Если ключ - a указывается без ключа -d, сгенерированные команды COPY копируют все данные из stdin (то есть записи буквально сохраняются в выходном файле). В противном случае записи представляются последовательными командами *INSERT*.

-b, --blobs. Большие двоичные объекты архивируются наряду с обычными данными.Также должен быть установлен ключ -F с форматом t или c. По умолчанию данные больших двоичных объектов не архивируются.

-c, --clean. Командам SQL, создающим объекты базы данных, должны предшествовать команды удаления этих объектов. Ключ обычно используется при повторной инициализации существующей базы данных (вместо ее удаления и создания на пустом месте).

-C, --create. В выходные данные включается команда SQL для создания базы данных (CREATE DATABASE).

-d, --inserts. Для записей генерируются команды *INSERT* вместо используемых по умолчанию команд *COPY*. Этот вариант безопаснее, так как одна поврежденная запись приводит к сбою всей команды *COPY*, но процесс восстановления занимает гораздо больше времени.

-D, --attributeinserts. Ключ -D, как и -d, генерирует команды *INSERT*, но в каждую команду *INSERT* перед секцией *VALUES* включается список полей в круглыхскобках.

-f файл, --file=файл. Результаты работы *pg_dump* направляются в заданный файл вместо потока stdout. Пользователь, запускающий *pg_dump*, должен иметь

системные права записи в этот файл.

-F {c | t | p}, --format {c | t | p}. Формат выходного файла. c (сжатие gzip) – файл .tar, сжатый утилитой gzip (тоесть .tar.gzip). t – файл .tar. p (простой текст) – выходной файл генерируется в простом текстовом варианте (режим используется по умолчанию).

-h хост, --host=хосm. Хост, с которым устанавливается связь вместо хоста local host. Используется в тех случаях, когда архивируемая база данных находится на другом сервере.

-i, --ignore-version. Запрет сравнения версии *pg_dump* с текущей версией PostgreSQL. Ключ следует использовать лишь в крайних случаях, поскольку различия в структуре системных каталогов разных версий с

большой вероятностью приведут к возникновению ошибок. Обычно версия *pg_dump* должна соответствовать версии архивируемой базы данных.

-n, --noquotes. Идентификаторы заключаются в кавычки только при наличии

недопустимых символов (пробелов, символов верхнего регистра и т. д.).

-N, --quotes. Все идентификаторы обязательно заключаются в кавычки. Используется в *pg_dump* по умолчанию, начиная с PostgreSQL 6.4.

-o, --oid. Вместе с данными записей архивируются OID (идентификаторы объектов). Ключ очень важен в приложениях, которые так или иначе осмысленно

используют OID.

-O, --no-owner. При создании архива не учитывается принадлежность базы данных. Объекты, созданные в результате восстановления данных, будут принадлежать пользователю, выполняющему эту операцию.

-p *порт*, --Qort=*порт*. Порт, по которому должно производиться подключение к серверу, вместо порта по умолчанию (обычно 5432, хотя при компиляции PostgreSQL можно задать другой порт при помощи флага --with-gport).

-R, --no-reconnect. Подавляет все команды \connect, которые обычно обеспечивают сохранение прав владельцев при восстановлении из архива. На практике ключ аналогичен ключу -O, но он также исключает возможность использования ключа -C, поскольку после создания новой базы необходимо заново установить подключение.

-s, --schema-only. Генерируются только команды SQL для архивации таких объектов, как таблицы, последовательности, индексы и представления, а хранящиеся в таблицах данные игнорируются. Ключ может использоваться для копирования общей структуры базы данных с компьютера разработчика на компьютер, на котором база будет реально эксплуатироваться.

-t *таблица*, --table=*таблица*. В заданной базе данных архивируется только заданная таблица.

-u, --password. Запрос имени пользователя и пароля. Если пароль не задан (NULL), то в ответ на запрос можно просто нажать клавишу Enter.

-v, --verbose. Вывод функций *pg_dump* направляется в поток stderr, а не в stdout.

-x, --no-acl. Подавление команд *GRANT* и *REVOKE*, обычно используемых для сохранения прав, действующих на момент архивации. Ключ используется в том случае, если при восстановлении базы данных из архива не нужно восстанавливать существовавшие ранее права или ограничения.

-Z, --compress {0-9}. Уровень сжатия (0 - минимальный, 9 - максимальный)
при использовании с ключом -F c.

По умолчанию приложение *pg_dump* может запускаться любым системным
пользователем, но пользователь, подключающийся к PostgreSQL, должен обладать правом выборки для всех объектов в архивируемой базе данных.

Если архивный файл должен содержать большие двоичные объекты, воспользуйтесь форматом tar (t) или gzip (c), потому что текстовый формат не дает такой возможности. В остальных случаях обычных текстовых архивов бывает вполне достаточно.

Архивы в формате tar нередко более чем в два раза превышают по объему свои текстовые прототипы, даже если они не содержат двоичных объектов. Дело в том, что в формат tar включается иерархическое оглавление файлов .dat. В этом оглавлении хранится информация, необходимая для распаковки формата tar соответствующей командой *pg_restore*; дополнительные инструкции занимают лишнее место на диске. Поскольку tar архивирует файлы без сжатия, был предусмотрен формат c, обеспечивающий автоматическое сжатие tar-файлов в формат gzip.

Существует два способа восстановления базы данных из архива. Если архив представляет собой простой текстовый файл, его можно передать psql

в качестве входного файла. Если же был выбран другой формат архива (.tar или .tar.gz), следует использовать приложение *pg_restore*.

При восстановлении данные либо заносятся в пустую базу данных, либо база данных специально создается. Выбор зависит в основном от способа архивации (то есть от того, содержит ли архив только данные или же в него были включены команды создания базы данных).

Простой текстовый файл, созданный приложением *pg_dump*, можно передать psql в качестве входного файла. При этом будут последовательно выполнены все инструкции SQL, хранящиеся в архиве. В зависимости от режима архивации существует несколько вариантов вызова psql.

Если архив создавался с ключом -C, команда SQL для создания базы данных удалена, либо еще не создана в той системе, в которой она восстанавливается. Если база данных уже существует, возможно, ее придется удалить - но только в том случае, если вы твердо убеждены в актуальности данных архива.

С другой стороны, если ключ -C не использовался, придется создать базу данных перед подключением и восстановлением ее атрибутов и данных. Помните, что клиенту psql также необходимо передать параметры для подключения в качестве пользователя с правами создания базы данных.

Если файл был создан программой *pg_dump* в формате, отличном от простого текста, его можно восстановить из архива .tar или .tar.gz при помощи утилиты *pg_restore*.

Синтаксис команды *pg_restore*:

pg_restore [параметры] [файл]

Если файл не задан, *pg_restore* ожидает поступления данных из потока stdin.

Следовательно, при вызове *pg_restore* могут использоваться средства перенаправления ввода (<). Среди параметров особого внимания заслуживает ключ -d. Если он не задан, *pg_restore* вместо восстановления базы данных просто выводит команды в поток stdout (то есть на экран).

При использовании ключа создания базы данных -C все равно необходимо задать ключ -d с именем существующей базы данных для подключения - например,template1. Неважно, к какой базе данных вы при этом подключаетесь, это всего лишь временное подключение до момента создания новой базы данных.

Многие ключи *pg_restore* совпадают с аналогичными ключами команды *pg_dump*.

Иногда для достижения желаемой цели один ключ должен передаваться как при вызове *pg_dump*, так и *pg_restore*. Например, это относится к ключу -C. Если ключ передается только при вызове команды *pg_dump,* то команда *CREATE DATABASE* будет проигнорирована при восстановлении, несмотря на ее присутствие в архиве.

Контрольные вопросы

1. Каким образом можно исправить структуру таблицы?
2. Какой формат имеет команда, с помощью которой можно скопировать структуру таблицы?
3. Какую команду в PostgreSQL можно использовать для создания баз данных?
4. Каким образом можно просмотреть структуру базы данных?
5. Каким образом можно вызвать справку для конкретной команды?
6. Как можно удалить таблицу, базу данных?
7. Каким образом осуществляется добавление столбцов в таблицы?
8. Каким образом можно создать столбцы с пользовательскими типами данных?
9. Как осуществляется добавление данных в таблицу?
10. Каким образом можно отредактировать данные в таблице?
11. Какая команда осуществляет удаление записей из таблиц?
12. С помощью какой команды можно получить описание таблицы с указанием списка ее полей?

Список источников

1. Малыхина М.П. Базы данных: основы, проектирование, использование: учебное пособие для вузов по направлению подготовки "Информатика и вычислительная техника" / М. П. Малыхина. - СПб., 2006. - 517 с. : ил.
2. Хомоненко А.Д. Базы данных: учебник для вузов по техническим и экономическим специальностям / [Хомоненко А. Д., Цыганков В. М., Мальцев М. Г.] ; под ред. А. Д. Хомоненко. - М., 2006. - 736 с. : ил., табл.
3. Разработка приложений на C# с использованием СУБД PostgreSQL : учебное пособие / [И. А. Васюткина и др.] ; Новосиб. гос. техн. ун-т. - Новосибирск, 2015. - 141, [1] с. : ил., табл.. - Режим доступа: http://elibrary.nstu.ru/source?bib_id=vtls000220068
4. Дж Уорсли Дж., Дрейк Дж. PostgreSQL. Для профессионалов - СПб.: Питер, 2003. -496 с.
5. PostgreSQL (русский) [Электронный ресурс]: Документация по PostgreSQL. – Режим доступа: http://postgresql.ru.net . - загл. с экрана.
6. Википедия свободная энциклопедия (русский) [Электронный ресурс]: PostgreSQL. – Режим доступа: http://ru.wikipedia.org. - загл. с экрана.
7. Компьютерная документация (русский) [Электронный ресурс]: Функции для работы с датой и временем. – Режим доступа: http://www.hardline.ru/ . - загл. с экрана.
8. Linux и Windows: помощь админам и пользователям (русский) [Электронный ресурс]: 15 команд для управления PostgreSQL. – Режим доступа: http://www.guruadmin.ru . - загл. с экрана.

ПРИЛОЖЕНИЯ

Приложение А.

Таблица 1. Типы расширенных кодировок:

Константа	**Код**	**Описание**
SQL_ASCII	0	Простой формат ASCII
EUC_JP	1	Японская расширенная кодировка Unix
EUC_CN	2	Китайская расширенная кодировка Unix
EUC_KR	3	Корейская расширенная кодировка Unix
EUC_TW	4	Тайваньская расширенная кодировка Unix
UNICODE	5	Юникод UTF-8
MULE_INTERNAL	6	Внутренний тип Mule
LATIN1	7	ISO 8859-1 (английский язык, с поддержкой некоторых европейских)
LATIN2	8	ISO 8859-2 (английский язык, с поддержкой некоторых европейских)
LATIN3	9	ISO 8859-3 (английский язык, с поддержкой некоторых европейских)
LATIN4	10	ISO 8859-4 (английский язык, с поддержкой некоторых европейских)
LATIN5	11	ISO 8859-5 (английский язык, с поддержкой некоторых европейских)
KOI8	12	KOI8-R
WIN	13	Windows CP1251
ALT	14	Windows CP866

Приложение Б.

Таблица 2.1. Функции для работы с датой[6]

Функция	Описание
current date	Возвращает текущую дату в виде значения типа date
current time	Возвращает текущее время в виде значения типа time
current timestamp	Возвращает текущие дату и время в виде значения типа timestamp
date_part(s. t)	Выделяет из значения типа timestamp компонент даты или времени, определяемый строкой s
date_part(s. i)	Выделяет из значения типа interval компонент даты или времени, определяемый строкой s
date trunc(s, t)	Возвращает значение типа timestamp, усеченное до точности s
extract (k FROM t)	Выделяет из значения типа timestamp компонент даты или времени, определяемый ключевым словом k
extract (k FROM i)	Выделяет из значения типа interval компонент даты или времени, определяемый ключевым словом k
isfinite(t)	Возвращает true, если значение типа timestamp соответствует конечной величине (не invalid и не infinity)
isfinite(i)	Возвращает true, если значение типа interval соответствует конечной величине (не infinity)

now()	Возвращает текущие дату и время в виде значения типа timestamp. Эквивалент константы now
Timeofday()	Возвращает текущие дату и время в виде значения типа text

Табл.2.2. Варианты корректного ввода даты

Пример	**Описание**
January 8, 1999	значение является однозначным для любого `datestyle` режима ввода
1999-01-08	ISO 8601; 8 января в любом режиме (рекомендуемый формат)
1/8/1999	8 января в режиме `MDY`; 1 августа в режиме `DMY`
1/18/1999	18 января в режиме `MDY`; в других режимах значение будет отвергнуто
01/02/03	2 января, 2003 года в режиме `MDY`; 1 февраля, 2003 года в режиме `DMY`; 3 февраля, 2001 года в режиме `YMD`
1999-Jan-08	8 января в любом режиме
Jan-08-1999	8 января в любом режиме
08-Jan-1999	8 января в любом режиме

Пример	**Описание**
99-Jan-08	8 января в режиме `YMD`, иначе ошибка
08-Jan-99	8 января, за исключением режима `YMD`, в котором будет ошибка
Jan-08-99	8 января, за исключением режима `YMD`, в котором будет ошибка
19990108	ISO 8601; 8 января, 1999 года в любом режиме
990108	ISO 8601; 8 января, 1999 года в любом режиме
1999.008	год и номер дня в году
J2451187	день по Юлианскому календарю
January 8, 99 BC	99 год до Нашей Эры

Приложение В

Таблица 1.1. Типы данных:

Имя	Псевдонимы	Описание
bigint	int8	знаковое восьмибайтное целое число
bigserial	serial8	восьмибайтное целое число с автоинкрементом
bit [(*n*)]		битовая строка с фиксированной длиной
bit varying [(*n*)]	varbit	битовая строка с переменной длиной
boolean	bool	логическое значение (истина/ложь)
box		четырёхугольник на плоскости
bytea		двоичные данные ("массив байт")
character varying [(*n*)]	varchar [(*n*)]	строка с переменной длиной
character [(*n*)]	char [(*n*)]	строка с фиксированной длиной
cidr		адрес сети IPv4 или IPv6
circle		круг на плоскости

Имя	**Псевдонимы**	**Описание**
`date`		календарная дата (год, месяц, день)
`double precision`	`float8`	число с плавающей точкой двойной точности
`inet`		адрес узла IPv4 или IPv6
`integer`	`int`, `int4`	знаковое четырёхбайтовое целое
`interval [`*`fields`*`] [(`*`p`*`)]`		промежуток времени
`line`		бесконечная линия на плоскости
`lseg`		сегмент линии на плоскости
`macaddr`		MAC адрес
`money`		денежное значение (в валюте)
`numeric [(`*`p`*`, `*`s`*`)]`	`decimal [(`*`p`*`, `*`s`*`)]`	точное числовое значение с выбраной точностью
`path`		геометрический путь на плоскости (ломаная)

Имя	**Псевдонимы**	**Описание**
`point`		геометрическая точка на плоскости
`polygon`		закрытый геометрический путь на плоскости (полигон)
`real`	`float4`	число с плавающей точкой одинарной точности
`smallint`	`int2`	знаковое двухбайтное целое число
`serial`	`serial4`	четырёхбайтное целое число с автоинкрементом
`text`		строка символов перменной длины
`time [(`*p*`)] [without time zone]`		время дня
`time [(`*p*`)] with time zone`	`timetz`	время дня, включая часовой пояс
`timestamp [(`*p*`)] [without time zone]`		дата и время
`timestamp [(`*p*`)] with time zone`	`timestamptz`	дата и время, включая часовой пояс

Имя	Псевдонимы	Описание
`tsquery`		запрос текстового поиска
`tsvector`		документ текстового поиска
`txid_snapshot`		снимок ID транзакции уровня пользователя
`uuid`		универсальный уникальный идентификатор
`xml`		данные XML

Приложение Г.

\df получить список доступных функций

\c[onnect] [ИМЯБД|- [ПОЛЬЗОВАТЕЛЬ]] подсоединиться к новой базе данных

\h [ИМЯ] подсказка по синтаксису SQL команд; * для всех команд

\q выйти из psql

\timing переключить режим замера запросов (в данный момент: выкл.)

\d [ИМЯ] описать таблицу, индекс, последовательность или вид

\d{t|i|s|v|S} (добавьте "+" для более детальной информации) показать таблицы/индексы/последовательности/виды/системные таблицы

\du показать пользователей

\l показать все базы данных (добавьте "+" для более детальной информации)

\i «файл» Прочить текстовый «*файл*» и выполнить имеющиеся в нём команды. Удобно для нетривиальных операций. Имя файла с командами можно передать при запуске psql через ключик *-f*. В этом случае после чтения и исполнения всех команд psql автоматическипрекращает работу.

\o [«файл»] Сохранить результаты выполнения будущих команд в «*файл*». Если

аргумент отсутствует, то вывод переключается на терминал. psql имеет набор команд, которые позволяют сформировать вывод как угодно пользователю. Указать имя файла, в котором следует сохранить результаты, также можно передать при запуске psql с помощью ключика *-o*. Этот ключ удобно применять совместно с ключом *-f*. [9]