МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

ЛЕКЦИОННЫЕ МАТЕРИАЛЫ

Технологии хранения в системах кибербезопасности

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень	бакалавриат
	(бакалавриат, магистратура, специалитет)
Форма обучения	очная
	(очная, очно-заочная, заочная)
Направление(-я) подготовки	10.05.04 Информационно-аналитические системы безопасности
	(код(-ы) и наименование(-я))
Институт	Кибербезопасности и цифровых технологий (ИКБ)
	(полное и краткое наименование)
Кафедра	КБ-2 «Прикладные информационные технологии»
	(полное и краткое наименование кафедры, реализующей дисциплину (модуль))
Лектор	к.т.н., Селин Андрей Александрович
	(сокращенно – ученая степень, ученое звание; полностью – ФИО)

Используются в данной редакции с учебного года 2024/2025

(учебный год цифрами)

Проверено и согласовано «____» _____2024 г. _____ А.А. Бакаев

(подпись директора Института/Филиала с расшифровкой)

Москва 2024 г.

Технологии хранения в системах кибербезопасности

2024 год

Лекция 5.
Система баз данных PostgreSQL

Учебные вопросы лекции:

1. Основы организации СБД PostgreSQL
2. Основы PL/pgSQL

1. Основы организации СБД PostgreSQL

PostgreSQL работает в операционных системах: Linux, Windows (XP и новее), FreeBSD, OpenBSD, NetBSD, macOS, AIX, HP/UX и Solaris. Прежде чем работать с БД, следует проинициализировать область хранения БД на диске – кластер БД, который представляет собой набор БД, управляемых одним экземпляром сервера. После инициализации кластер будет содержать БД postgres, предназначенную для использования по умолчанию. При инициализации в каждом кластере создается еще одна БД template1, которая применяется в качестве шаблона создаваемых БД.

С точки зрения файловой системы кластер БД представляет собой один каталог, в котором будут храниться все данные. Так как каталог данных содержит все данные БД, важно защитить его от неавторизованного доступа. Чтобы можно было обратиться к БД, следует сначала запустить сервер БД. Программа сервера называется postgres, которая запускается в операционной системе в виде фонового процесса.

1. Основы организации СБД PostgreSQL (продолжение)

Чтобы можно было обратиться к БД, следует сначала запустить сервер БД. Программа сервера называется postgres, которая запускается в операционной системе в виде фонового процесса.

На работу СБД оказывает влияние множество параметров конфигурации. Имена всех параметров я
значение одного
точкой или п
определени
данных.

Каждый параметр определяется в отдельной строке. Знак равенства в ней между именем и значением является необязательным. Пробельные символы в строке не играют роли (кроме значений, заключённых в апострофы), а пустые строки игнорируются. Знаки решётки (#) обозначают продолжение строки как комментарий. Значения параметров, не являющиеся простыми идентификаторами или числами, должны заключаться в апострофы. Если один и тот же параметр определяется в файле конфигурации неоднократно, действовать будет только последнее определение, остальные игнорируются.

1. Основы организации СБД PostgreSQL (продолжение)

В каталоге данных PostgreSQL содержится файл postgresql.auto.conf, который имеет тот же формат, что и postgresql.conf, но предназначен для автоматического изменения, а не для редактирования вручную. Этот файл содержит параметры, задаваемые командой Alter System. Параметры в postgresql.auto.conf переопределяют те, которые указаны в postgresql.conf. PostgreSQL предоставляет SQL-команды для управления параметрами конфигурации:

– команда SHOW позволяет узнать текущее значение параметров; соответствующая ей функция – current_setting (имя_параметра text).

– команда SET позволяет изменить текущее значение параметров, которые действуют локально в рамках сеанса; соответствующая ей функция – set_config (имя_параметра, новое_значение, локально).

При подключении к серверу БД, клиентское приложение указывает имя пользователя PostgreSQL. По имени пользователя определяется, какие у него есть права доступа к объектам БД. Аутентификация клиентов управляется конфигурационным файлом pg_hba.conf, который расположен в каталоге с данными кластера БД (host-based authentication – аутентификации по имени узла).

1. Основы организации СБД PostgreSQL

PostgreSQL использует концепцию ролей для управления доступом к объектам БД. Концепция ролей включает в себя концепцию пользователей и групп.

Система сразу после инициализации всегда содержит одну предопределенную роль postgres с правами суперпользователя. Для создания других ролей, вначале нужно подключиться с этой ролью. Каждое подключение к серверу БД выполняется под именем конкретной роли, и эта роль определяет начальные права доступа для команд, выполняемых в этом соединении.

PostgreSQL предлагает широкий набор методов аутентификации. Основные из них:
– trust – сервер доверяет пользователям, никак не проверяя их;
– md5 – метод аутентификации по паролю, предусматривает хранение паролей на сервере в зашифрованном виде.

1. Основы организации СБД PostgreSQL (продолжение)

Каждый экземпляр сервера PostgreSQL обслуживает одну или несколько БД. Некоторые объекты, включая роли, БД и табличные пространства, определяются на уровне кластера и сохраняются в табличном пространстве pg_global. Внутри кластера существуют БД, которые отделены друг от друга, но могут обращаться к объектам уровня кластера. Внутри каждой БД имеются схемы, содержащие такие объекты, как таблицы и функции. Таким образом, полная иерархия выглядит следующим образом: кластер, БД, схема, объект БД (таблица, функция и др.).

При подключении к серверу БД клиент должен указать имя БД. Обращаться к нескольким БД через одно подключение нельзя, однако клиенты могут открыть несколько подключений. Для создания БД сервер PostgreSQL должен быть развернут и запущен. БД создается SQL-командой Create Database: CREATE DATABASE имя; где имя БД подчиняется правилам именования идентификаторов SQL. БД удаляются командой Drop Database: DROP DATABASE имя; Лишь владелец БД или суперпользователь могут ее удалить. Удаление БД – необратимая операция. Нельзя выполнить команду Drop Database пока существует хотя бы одно подключение к ней.

1. Основы организации СБД PostgreSQL (продолжение)

Надёжность – важное свойство СУБД. Один из аспектов надежности PostgreSQL состоит в том, что все данные записываются в журнал предварительной записи (WAL), которые сохраняются в энергонезависимой области. Основная идея WAL состоит в том, что изменения в файлах с данными (где находятся таблицы и индексы) должны записываться только после того, как эти изменения были занесены в журнал, т. е. после того как записи журнала, описывающие данные изменения, будут сохранены на внешнюю память. Результатом использования WAL является значительное уменьшение количества операций записи на диск, потому что для гарантии, что транзакция подтверждена, в записи на диск нуждается только файл журнала, а не каждый файл данных измененный в результате транзакции. Журналы WAL хранятся в виде набора файлов в каталоге pg_wal. Имеет смысл размещать журналы WAL на другом диске, отличном от того, где находятся основные файлы БД.

1. Основы организации СБД PostgreSQL (окончание)

БД не будет создана этой командой, её следует создать предварительно. Альтернативной стратегией резервного копирования является непосредственное копирование файлов, в которых PostgreSQL хранит содержимое БД. Ограничения, которые делают этот метод менее предпочтительным по сравнению с pg_dump: копирование файлов должно выполнятся при остановленном сервере БД; можно выполнить копирование всего кластера БД целиком.

Наличие журнала делает возможным использование третьей стратегии копирования БД: можно сочетать резервное копирование на уровне файлов с копированием файлов WAL. Если потребуется восстановить БД, можно восстановить ее копию из файлов, а затем воспроизвести журнал из скопированных файлов WAL, и таким образом привести систему в нужное состояние. Этот метод позволяет восстанавливать только весь кластер БД целиком.

2. Основы PL/pgSQL

Оператор DO позволяет выполнить анонимный блок кода на процедурном языке. Синтаксис:

DO [LANGUAGE имя] код

Код задается в виде текстовой строки (заключенной в апострофы или двойные знаки доллара). Имя определяет процедурный язык, на котором написан код (по умолчанию PL/pgSQL).

PL/pgSQL – процедурный язык СУБД PostgreSQL, который используется для создания функций и триггеров, добавляет управляющие и циклические структуры к языку SQL.

PL/pgSQL – блочно-структурированный язык.

Упрощенная структура блока:

```
[ <<метка>> ]
[ DECLARE
    объявление; [ ... ] ]
BEGIN
    оператор; [ ... ]
END [ метка ];
```

Каждое объявление и каждый оператор в блоке должны завершаться точкой с запятой. Блок может быть вложен в другой блок. Ключевые слова не чувствительны к регистру символов. Идентификаторы неявно преобразуются к нижнему регистру, если они не взяты в двойные кавычки. Комментарии в PL/pgSQL коде обозначаются так же, как и в SQL.

2. Основы PL/pgSQL(продолжение)

Выражение должно получить одно значение. Если тип данных результата выражения не соответствует типу данных переменной, это значение будет преобразовано к нужному типу с использованием приведения типов. Оператор RAISE предназначен для вывода сообщений (ошибок) на экран:

RAISE [уровень] 'строка' [, выражение [, ...]]

уровень – важность сообщения, возможные значения: DEBUG, LOG, INFO, NOTICE, WARNING и EXCEPTION (по умолчанию).

строка – текст выводимого сообщения, который может содержать местозаполнители в виде знака %;

выражение подставляется на местозаполнитель; количество выражений должно совпадать с числом местозаполнителей % в строке.

Условные операторы PL/pgSQL – IF и CASE – позволяют выполнять команды в зависимости от определенных условий. PL/pgSQL поддерживает три формы IF и две формы CASE. IF-THEN – простейшая форма оператора IF:

```
IF условие THEN операторы
END IF;
```

2. Основы PL/pgSQL (продолжение)

Операторы между THEN и END IF выполняются, если условие (логическое выражение) истинно. В противном случае они не выполняются. Форма IF-THEN-ELSE добавляет к IF-THEN возможность указать альтернативный набор операторов, которые будут выполнены, если условие не истинно (ложно или NULL):

```
IF условие THEN оператор
ELSE операторы
END IF;
```

В некоторых случаях д
обеспечивает способ прове

```
IF условие THEN оператор
[ ELSIF условие THEN опе
[ ELSIF условие THEN опе
[ ELSE операторы ]
END IF;
```

Вместо ключевого слова E

Условия в IF последовательно проверяются до тех пор, пока не будет найдено первое истинное. При этом операторы, относящиеся к этому условию, выполняются, и управление переходит к следующей после END IF команде (остальные условия не проверяются.) Если ни одно из условий IF не является истинным, то выполняются операторы секции ELSE. Если ни одно из условий IF не является истинным и отсутствует секция ELSE, то не будет выполнен ни один из операторов IF

2. Основы PL/pgSQL (продолжение)

Простая форма CASE реализует условное выполнение на основе сравнения переменной (выражения поиска) с набором значений (выражений):

```
CASE выражение-поиска
        WHEN выражение [, выражение [...] ] THEN операторы
        [ WHEN выражение [, выражение [...] ] THEN операторы ... ]
        [ ELSE операторы ]
END CASE;
```

Выражение поиска выч дым
выражением в услов тся
соответствующие опер ASE
команде (все последую е не
найдено, то выполняю но и
секция ELSE отсутству

Форма CASE с перебором условий реализует условное выполнение, основываясь на истинности логических условий:

```
CASE
        WHEN условие THEN операторы
        [ WHEN условие THEN операторы ... ]
        [ ELSE операторы ]
END CASE;
```

2. Основы PL/pgSQL (продолжение)

Если метка не указана, то завершается самый внутренний цикл, далее выполняется оператор, следующий за END LOOP. Если метка указана, то она должна относиться к текущему или внешнему циклу, или это может быть метка блока. При этом в именованном цикле/блоке выполнение прекращается, а управление переходит к следующему оператору после соответствующего END. При наличии WHEN цикл прекращается, только если условие (логическое выражение) истинно. В противном случае управление переходит к оператору, следующему за EXIT.
Когда EXIT используется для выхода из блока, управление переходит к следующему оператору после окончания блока. Для выхода из блока нужно обязательно указывать метку после EXIT.

бесконечности, пока не будет прекращен оператором EXIT.

```
[ <<метка>> ]
LOOP
        операторы
END LOOP [ метка ];
```

Оператор EXIT позволяет завершить цикл и выйти из него, а также выйти из блока. EXIT можно использовать со всеми типами циклов. Его синтаксис:

```
EXIT [ метка ] [WHEN условие];
```

2. Основы PL/pgSQL (продолжение)

Оператор CONTINUE позволяет пропустить итерацию цикла, т. е. опустить все оставшиеся операторы в теле цикла. CONTINUE можно использовать со всеми типами циклов. Его синтаксис:

CONTINUE [метка] [WHEN условие];

Если метка не указана, то начинается следующая итерация самого внутреннего цикла. Если имеется метка, то она указывает на цикл, выполнение которого будет продолжено. При наличии WHEN следующая итерация цикла начинается только тогда, когда условие истинно. В противном случае управление переходит к оператору

Переменная цикла автоматически определяется с типом INTEGER и существует внутри цикла (если уже существует переменная с таким именем, то внутри цикла она будет игнорироваться). Выражения для нижней и верхней границы диапазона чисел вычисляются один раз при входе в цикл. Если не указано BY, то шаг итерации 1, в противном случае используется значение в BY, которое вычисляется один раз при входе в цикл. Если указано REVERSE, то после каждой итерации величина шага вычитается, а не добавляется. Если нижняя граница цикла больше верхней границы (или меньше, в случае REVERSE), то тело цикла не выполняется вообще.

2. Основы PL/pgSQL (продолжение)

Форма цикла FOR по результатам запроса:
[<<метка>>]
FOR имя IN запрос LOOP

SQL-команда CREATE FUNCTION позволяет создать функцию. Ее упрощенный синтаксис:
CREATE [OR REPLACE] FUNCTION имя ([[режим_аргумента] [имя_аргумента] тип_аргумента [{ DEFAULT | = } выражение_по_умолчанию] [, ...]]) [RETURNS тип_результата | RETURNS TABLE (имя_столбца тип_столбца [, ...])] AS 'определение' LANGUAGE имя_языка

INSERT, UPDATE или DELETE с предложением RETURNING.

В PostgreSQL представлены функции следующих видов: на языке SQL, процедурном языке PL/pgSQL и внутренние функции. Функции любых видов могут принимать в качестве аргументов (параметров) базовые или составные типы и возвращать значения базового или составного типа.

2. Основы PL/pgSQL (продолжение)

Команда Create Or Replace Function позволяет заменить текущее определение существующей функции, но она не позволяет изменить имя или аргументы функции, а также тип результата существующей функции. Имя создаваемой функции должно отличаться от имен существующих функций с такими же типами аргументов в этой схеме. Однако функции с аргументами разных типов могут иметь одно имя (это называется перегрузкой). Режим аргумента: IN (входной, по умолчанию), OUT (выходной), INOUT (входной и выходной). Значения по умолчанию могут иметь только входные параметры, включая INOUT. Определение функции задается в виде строковой константы. Полезно заключать определение функции в знаки долларов, а не в апострофы, поскольку при использовании апострофов, все апострофы и обратные косые черты в определении функции приходится экранировать, дублируя их. Язык, на котором реализована функция, может быть SQL, C или процедурный язык, например,

Функции на языке запросов (SQL) выполняют список операторов SQL и возвращают результат последнего запроса в списке. В простом случае будет возвращена первая строка результата последнего запроса (первая строка в наборе с несколькими строками определена точно при использовании ORDER BY). Если последний запрос не вернет строки, будет возвращено NULL.

2. Основы PL/pgSQL (продолжение)

Кроме того, SQL-функцию можно объявить как возвращающую множество строк,

Альтернативный способ описать результаты функции – определить ее с выходными параметрами. Выходные параметры позволяют удобным способом определить функции, возвращающие несколько столбцов. Выходные параметры не включаются в список аргументов при вызове функции. PostgreSQL определяет сигнатуру вызова функции, рассматривая только входные параметры.

точкой с запятой. Если функция не объявлена как возвращающая VOID, последним оператором должен быть SELECT, либо INSERT, UPDATE или DELETE с предложением RETURNING. Любой набор команд на языке SQL можно скомпоновать вместе и обозначить как функцию.

К аргументам SQL-функции можно обращаться в теле функции по именам или номерам. Подход с именами позволяет обращаться к аргументам по 11 именам. Это возможно, если аргументу функции назначено имя. Можно дополнить имя аргумента именем функции функция.аргумент, чтобы сделать имя аргумента однозначным. Подход с нумерацией позволяет обращаться к аргументам, применяя запись вида $n ($1 – обращение к первому аргументу, $2 – ко второму и т. д.). Через номер можно обращаться и к именованным аргументам. Если аргумент имеет составной тип, то для обращения к его атрибутам можно использовать запись с точкой: аргумент.поле или $1.поле. Можно дополнить имя аргумента именем функции.

2. Основы PL/pgSQL (продолжение)

Переименовать функцию, сменить владельца или схему функции можно с помощью команды Alter Function:

ALTER FUNCTION имя [([[режим_аргумента] [имя_аргумента] тип_аргумента [, ...]])]
{ RENAME TO новое_имя |
OWNER TO новый_владелец |
SET SCHEMA новая_схема }

Удалить функцию можно с помощью команды DROP FUNCTION:

DROP FUNCTION [IF EXISTS] имя ([[режим_аргумента] [имя_аргумента] тип_аргумента [, ...]]) [, ...] [CASCADE | RESTRICT]

вызываются в предложении From.

Еще один способ объявить функцию, возвращающую множество, – использовать синтаксис Returns Table(столбцы). Запись Returns Table не позволяет явно указывать OUT и INOUT для параметров; все выходные столбцы необходимо записать в списке Table.

2. Основы PL/pgSQL (продолжение)

Функции на PL/pgSQL определяются командами CREATE FUNCTION. Текст тела

Триггер можно настроить так, чтобы он срабатывал до (Before) или (After) события. В качестве события допускаются одна или несколько операций INSERT, UPDATE, DELETE или TRUNCATE. Триггеры UPDATE можно настроить на изменение значений только заданных столбцов, для этого следует указать в качестве события: UPDATE OF имя_столбца [, ...]

результат выполнения запроса (выражения) к результату функции.

RETURN QUERY запрос;
RETURN NEXT выражение;

Триггер автоматически выполняет заданную функцию всякий раз, когда выполняется определенный тип операции с таблицей (представлением) БД.

Создать триггер можно с помощью SQL-команды Create Trigger:

CREATE TRIGGER имя { BEFORE | AFTER }
{ событие [OR ...] } ON имя_таблицы
[FOR [EACH] { ROW | STATEMENT }] [WHEN (условие)]
EXECUTE FUNCTION имя_функции (аргументы)

2. Основы PL/pgSQL (продолжение)

Таблица 1 – Переменные, доступные в триггерных функциях

Переменная	Тип данных	Описание
NEW	RECORD	содержит новую строку таблицы для команд INSERT и UPDATE в триггерах уровня строки; имеет значение NULL в триггерах уровня оператора и для команды DELETE
OLD	RECORD	содержит старую строку таблицы для команд UPDATE и DELETE в триггерах уровня строки; имеет значение NULL в триггерах уровня оператора и для команды INSERT
TG_NAME	name	содержит имя сработавшего триггера
TG_WHEN	text	строка, содержащая BEFORE, AFTER или INSTEAD OF, в зависимости от определения триггера
TG_LEVEL	text	строка, содержащая ROW или STATEMENT, в зависимости от определения триггера
TG_OP	text	строка, содержащая INSERT, UPDATE, DELETE или TRUNCATE, в зависимости от операции, для которой сработал триггер
TG_TABLE_NAME	name	содержит имя таблицы, для которой сработал триггер
TG_TABLE_SCHEMA	name	содержит имя схемы таблицы, для которой сработал триггер

2. Основы PL/pgSQL (окончание)

Триггерная функция должна вернуть либо NULL, либо строку, соответствующую структуре таблицы, для которой сработал триггер. Триггер BEFORE уровня строк для операций INSERT и UPDATE должен вернуть значение NEW, а для операций DELETE – значение OLD.

Переименовать триггер можно с помощью команды ALTER TRIGGER:

```
ALTER TRIGGER имя ON имя_таблицы RENAME TO новое_имя
```

Включить или отключить действие триггера можно с помощью команды ALTER TABLE:

```
ALTER TABLE { ENABLE | DISABLE } TRIGGER { имя_триггера | ALL }
```

Удалить триггер можно с помощью команды DROP TRIGGER:

```
DROP TRIGGER [ IF EXISTS ] имя ON имя_таблицы [ CASCADE | RESTRICT ]
```

Спасибо за внимание!