1. Инструкции языка описания данных, инструкции языка обработки данных, инструкции безопасности, инструкции управления транзакциями.

Язык DLL – Data Definition Lsanguage. Data Definition Language (DDL) – это группа операторов определения данных.

Ядро языка определения данных образуют три команды:

- Создать – CREATE. Позволяет определить и создать объект базы данных.
- Удалить – DROP. Служит для удаления существующего объекта базы данных.
- Изменить – ALTER. С её помощью можно изменить определение объекта базы данных.

DML (Data Manipulation Language) – инструкции языка манипулирования данными. Это группа операторов для манипуляции данными. К числу элементов языка обработки данных (DML) относятся инструкции:

- SELECT – осуществляет выборку данных.
- INSERT – добавляет новые данные.
- UPDATE – изменяет существующие данные.
- DELETE – удаляет данные.

Data Control Language (DCL) – группа операторов определения доступа к данным. Иными словами, это операторы для управления разрешениями, с помощью них мы можем разрешать или запрещать выполнение определенных операций над объектами базы данных.

Сюда входят:

- GRANT – предоставляет пользователю или группе разрешения на определённые операции с объектом.
- REVOKE – отзывает выданные разрешения.
- DENY– задаёт запрет, имеющий приоритет над разрешением.

Transaction Control Language (TCL) – группа операторов для управления транзакциями. Транзакция – это команда или блок команд (инструкций), которые успешно завершаются как единое целое, при этом в базе данных все внесенные изменения фиксируются на постоянной основе или отменяются, т.е. все изменения, внесенные любой командой, входящей в транзакцию, будут отменены.

Группа операторов TCL предназначена для реализации и управления транзакциями. Сюда можно отнести:

- BEGIN TRANSACTION – служит для определения начала транзакции.
- COMMIT TRANSACTION – применяет транзакцию.

- ROLLBACK TRANSACTION – откатывает все изменения, сделанные в контексте текущей транзакции.
- SAVE TRANSACTION – устанавливает промежуточную точку сохранения внутри транзакции.

2. Реляционная модель данных. Структурная, целостная, манипуляционная части. Реляционная алгебра. Исчисление кортежей.

Реляционная модель данных (РМД) — логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка.

На реляционной модели данных строятся реляционные базы данных.

Основа реляционной модели:

- Структурная часть – говорит о том, какие объекты есть в системе и из чего они состоят.
- Целостная часть – какие есть ограничения.
- Манипуляционная часть – обработка данных.

Структурная часть:

- Типы (int, varchar, …).
- Домен – ограничение над типом или доменом (целое положительное, целое неотрицательное).
- Атрибуты – пара имя - тип/домен.
- Заголовок отношения – множество всех атрибутов.
- Кортеж – одна строка. Множество упорядоченных пар типа *<атрибут, значение>*. Уникален.
- Отношения.

Пусть дана совокупность типов данных T1, T2, ..., Tn, называемых также доменами, не обязательно различных. Тогда n-арным отношением R, или отношением R степени n называют подмножество Декартова произведения множеств T1, T2, ..., Tn.

Отношение R состоит из заголовка (схемы) и тела. Заголовок представляет собой множество атрибутов (именованных вхождений домена в заголовок отношения), а тело — множество кортежей, соответствующих заголовку.

Непустое подмножество множества атрибутов схемы будет потенциальным ключом тогда и только тогда, когда оно будет обладать свойствами уникальности и неизбыточности. В реляционной модели один из потенциальных ключей должен быть выбран первичным.

Количество кортежей называют кардинальным числом отношения (кардинальностью), или мощностью отношения.

Количество атрибутов называют степенью, или «арностью» отношения.

{<a1, int>, <a2, float>} – заголовок отношения.

Целостная часть – целостность сущностей и целостность ссылок.

- Целостность сущностей – каждый кортеж любого отношения должен отличаться от любого другого кортежа этого отношения.
- Ссылочная целостность – для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

Манипуляционная часть:

- Реляционная алгебра
- Реляционные исчисления

Реляционная алгебра — замкнутая система операций над отношениями в реляционной модели данных (работа с множествами).

Стандартные операции реляционной алгебры:

1. Традиционные
    a. Объединение (UNION)
    b. Пересечение (INTERSECT)
    c. Вычитание (MINUS)
    d. Декартово произведение (TIMES)
2. Специальные
    a. Соединение (JOIN)
    b. Ограничение (WHERE)
    c. Проекция (PROJECT)
    d. Деление (DIVIDE BY)

Результат любой операции реляционной алгебры над отношениями – отношение.

Дополнительные операторы реляционной алгебры:

1. EXTEND – расширение.
2. SUMMARIZE – обобщение.
3. GROUP – группирование.
4. UNGROUP – разгруппирование.
5. Реляционные сравнения (>, =, IS_EMPTY <реляционное_выражение>).

Реляционные выражения:

1. Унарные – ограничение, проекция, переименование.
2. Бинарные – всё остальное.

Реляционные исчисления:

- Исчисление кортежей.
- Исчисление доменов.

Исчисление кортежей:

- Where, правильно построенная функция
- Объявление кортежной переменной: *RANGE OF переменная IS список_областей.*
- Область := отношения | реляционное выражение.
- Реляционное выражение: *(список целевых элементов) [where(wff)]*.
- Целевой элемент := переменная | атрибут [AS NAME].
- Правильно построенная функция – некое условие или его отрицание, либо соединение условия с другой ППФ, либо предикат FOR ALL/EXISTS.

✓ RANGE OF SX IS S

✓ RANGE OF SY IS (SX) WHERE SX.City = “Смоленск”, (SX) WHERE EXISTS SRX (SPX.Sno = SX.Sno AND SPX.Pno = 1)

✓ // поставщики, поставляющие деталь №2.
RANGE OF SX IS S
RANGE OF SPX IS SP
(SPX.Sno) WHERE SPX.Pno=2
(SX.Sname) WHERE EXISTS SPX (SPX.Sno = SX.Sno AND SPX.Pno=2)

3. Семантическое моделирование данных

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных).

ER-модель используется при высокоуровневом (концептуальном) проектировании баз данных. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями.

Основными понятиями ER-модели являются сущность, связь и атрибут.

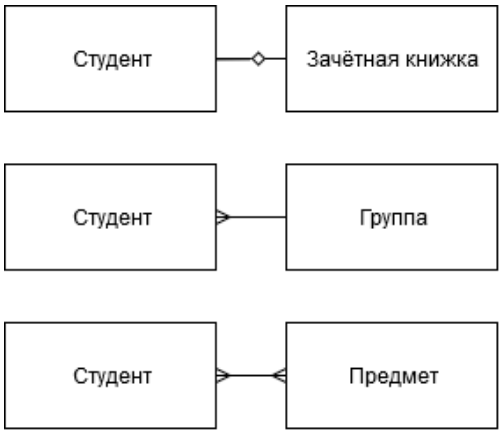
Сущность – это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности.

Связь – это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи). Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой.

Связи:

- 1 – n

- n – n
- 1 – 1



Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами.

Сильные сущности – ни от чего не зависят. Изображаются в прямоугольнике. Слабые сущности – зависят от другого объекта. Изображаются в двойном прямоугольнике.

3. Теория проектирования реляционных баз данных. Функциональные зависимости.

БД – самодокументируемое собрание интегрированных (хранящих информацию об объектах) записей.

Самодокументирование:

- Наличие метаданных (информации о хранимых данных).
- Журналирование.

СУБД – отвечает за доступ к данным и корректную работу с ними. Состоит из:

- Ядро СУБД – отвечает за способ работы с данными.
- Процессор языка БД.

- Оптимизатор.
- Специальные службы.

Требования к БД:

1. Безопасность
    a. Внешняя – от мошенников. Не должно быть доступа извне.
    b. Внутренняя – защита от пользователей. Различные права доступа.
2. Время отклика
3. Восстановление после сбоев
4. Однозначность / неизбыточность
5. Совместный доступ
6. Целостность
7. Независимость от прикладных программ.

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.

Основные задачи:

Обеспечение хранения в БД всей необходимой информации.

Обеспечение возможности получения данных по всем необходимым запросам.

Сокращение избыточности и дублирования данных.

Обеспечение целостности базы данных.

Основные этапы проектирования БД:

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации на какую-либо конкретную СУБД и модель данных. Обычно используются графические нотации, подобные ER-диаграммам.

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Физическое проектирование — создание схемы базы данных для конкретной СУБД.

Функциональная зависимость — бинарное отношение между множествами атрибутов данного отношения и является, по сути, связью типа «один ко многим».

Если даны два атрибута X и Y некоторого отношения, то говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X соответствует ровно одно значение Y.

Функциональная зависимость обозначается X -> Y. Отметим, что X и Y могут представлять собой не только единичные атрибуты, но и группы, составленные из нескольких атрибутов одного отношения.

Правила:

1. $(B \subseteq A) \Rightarrow A \longrightarrow B$
2. $A \longrightarrow B \Rightarrow AC \longrightarrow BC$
3. $A \longrightarrow B$ и $B \longrightarrow C \Rightarrow A \longrightarrow C$
4. $A \longrightarrow A$
5. $A \longrightarrow BC \Rightarrow A \longrightarrow B$ и $A \longrightarrow C$
6. $A \longrightarrow B$ и $A \longrightarrow C \Rightarrow A \longrightarrow BC$
7. $A \longrightarrow B$ и $C \longrightarrow D \Rightarrow AC \longrightarrow BD$
8. $A \longrightarrow B$ и $C \longrightarrow D \Rightarrow A(C - B) \longrightarrow BD$

Неприводимое (минимальное в нормальной форме) покрытие : правая часть всегда одноэлементна; левая часть неприводима; нет лишних зависимостей (минимальное – объединяем зависимости с одинаковой левой частью).

Функциональные зависимости являются ограничениями целостности и определяют семантику хранящихся в БД данных. При каждом обновлении СУБД должна проверять их соблюдение. Следовательно, наличие большого количества функциональных зависимостей нежелательно, иначе происходит замедление работы. Для упрощения задачи необходимо сократить набор функциональных зависимостей до минимально необходимого.

Если I является неприводимым покрытием исходного множества функциональных зависимостей S, то проверка выполнения функциональных зависимостей из I автоматически гарантирует выполнение всех функциональных зависимостей из S. Таким образом, задача поиска минимально необходимого набора сводится к отысканию неприводимого покрытия множества функциональных зависимостей, которое и будет использоваться вместо исходного множества.

5. Теория проектирования реляционных баз данных. Нормальные формы.

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности.

Основные задачи:

Обеспечение хранения в БД всей необходимой информации.

Обеспечение возможности получения данных по всем необходимым запросам.

Сокращение избыточности и дублирования данных.

Обеспечение целостности базы данных.

Основные этапы проектирования БД:

Концептуальное (инфологическое) проектирование — построение семантической модели предметной области, то есть информационной модели наиболее высокого уровня абстракции. Такая модель создаётся без ориентации

на какую-либо конкретную СУБД и модель данных. Обычно используются графические нотации, подобные ER-диаграммам.

Логическое (даталогическое) проектирование — создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели данных даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

Физическое проектирование — создание схемы базы данных для конкретной СУБД.

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Достоинства нормализации:

+ Избавление от аномалий
    - Аномалии:
        - аномалия обновления – проявляется в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.
        - аномалия удаления – при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.
        - аномалия вставки – возникает, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.
+ Уменьшение объема данных
+ Избавление от NULL

Недостатки нормализации:

– Время запроса увеличивается.

Нормализация – это процесс, в результате которого из нескольких больших таблиц создается много маленьких.
Цели нормализации:

1. Избавиться от аномалий
2. Уменьшить (минимизировать) объем хранимых данных.

1NF

- Все атрибуты должны быть атомарными.

Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ.

Исходная ненормализованная (то есть не являющаяся правильным представлением некоторого отношения) таблица:

Сотрудник	**Номер телефона**
Иванов И. И.	283-56-82 390-57-34
Петров П. П.	708-62-34

Таблица, приведённая к 1НФ, являющаяся правильным представлением некоторого отношения:

Сотрудник	**Номер телефона**
Иванов И. И.	283-56-82
Иванов И. И.	390-57-34
Петров П. П.	708-62-34

2NF

- 1NF
- Каждый полный атрибут функционально зависит от ключа

Пусть в следующем отношении первичный ключ образует пара атрибутов {*Филиал компании*, *Должность*}:

R

Филиал компании	**Должность**	**Зарплата**	**Наличие компьютера**
Филиал в Томске	Уборщик	20000	Нет
Филиал в Москве	Программист	40000	Есть
Филиал в Томске	Программист	25000	Есть

Допустим, что зарплата зависит от филиала и должности, а наличие компьютера зависит только от должности.

Существует функциональная зависимость *Должность* → *Наличие компьютера*, в которой левая часть (детерминант) является лишь частью первичного ключа, что нарушает условие второй нормальной формы.

Для приведения к 2NF исходное отношение следует декомпозировать на два отношения:

R1

Филиал компании	**Должность**	**Зарплата**
Филиал в Томске	Уборщик	20000
Филиал в Томске	Программист	25000
Филиал в Москве	Программист	40000

R2

Должность	**Наличие компьютера**
Уборщик	Нет
Программист	Есть

3NF

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

Рассмотрим в качестве примера переменную отношения R1:

R1

Сотрудник	Отдел	Телефон
Гришин	Бухгалтерия	11-22-33
Васильев	Бухгалтерия	11-22-33
Петров	Снабжение	44-55-66

Каждый сотрудник относится исключительно к одному отделу; каждый отдел имеет единственный телефон. Атрибут *Сотрудник* является первичным ключом. Личных телефонов у сотрудников нет, и телефон сотрудника зависит исключительно от отдела.

В примере существуют следующие функциональные зависимости: *Сотрудник* → *Отдел*, *Отдел* → *Телефон*, *Сотрудник* → *Телефон*.

Переменная отношения R1 находится во второй нормальной форме, поскольку каждый атрибут имеет неприводимую функциональную зависимость от потенциального ключа *Сотрудник*.

Зависимость *Сотрудник* → *Телефон* является транзитивной, следовательно, отношение не находится в третьей нормальной форме.

В результате разделения R1 получаются две переменные отношения, находящиеся в 3NF:

R2

Отдел	Телефон
Бухгалтерия	11-22-33
Снабжение	44-55-66

R3

Сотрудник	Отдел
Гришин	Бухгалтерия
Васильев	Бухгалтерия
Петров	Снабжение

OLTP – нормализация.
OLAP – денормализация.

6. Инструкции управления потоком (T-SQL).

Инструкция	Описание
BEGIN...END	Включает в себя последовательность инструкций языка Transact-SQL, позволяя выполнять группу инструкций Transact-SQL. Ключевые слова BEGIN и END относятся к языку потока управления.
BREAK	Выполняет выход из самого внутреннего цикла в инструкции WHILE или из инструкции IF...ELSE в цикле WHILE. Выполняется любая инструкция, находящаяся сразу после ключевого слова END, обозначающего конец цикла. Часто, но не всегда, ключевое слово BREAK встречается после проверки условия инструкции IF.
CONTINUE	Производит перезапуск цикла WHILE. Никакие инструкции после ключевого слова CONTINUE не выполняются. CONTINUE часто, но не всегда, предваряется проверкой IF.
GOTO label	Переводит поток выполнения на метку. Инструкции языка Transact-SQL или инструкции, следующие за инструкцией GOTO, пропускаются, и выполнение продолжается с метки. Инструкция GOTO и метки могут быть включены в процедуру, пакет или блок инструкций. Инструкции GOTO могут быть вложенными.
IF...ELSE	Задает условия для выполнения инструкции Transact-SQL. Инструкция языка Transact-SQL, следующая за ключевым словом IF и его условием выполняется только в том случае, если логическое выражение возвращает TRUE. Необязательное ключевое слово ELSE представляет другую инструкцию языка Transact-SQL,

	которая выполняется, если условие IF не удовлетворяется: логическое выражение возвращает FALSE.
RETURN	Служит для безусловного выхода из запроса или процедуры. Инструкция RETURN выполняется немедленно и полностью и может использоваться в любой точке для выхода из процедуры, пакета или блока инструкций. Инструкции, следующие после RETURN, не выполняются.
THROW	Вызывает исключение и передает выполнение блоку CATCH конструкции TRY...CATCH в SQL Server.
TRY...CATCH	Реализация обработчика ошибок на языке Transact-SQL похожа на обработку исключений в языках Microsoft Visual C# и Microsoft Visual C++. Группа инструкций на языке Transact-SQL может быть заключена в блок TRY. Если ошибка возникает в блоке TRY, управление передается следующей группе инструкций, заключенных в блок CATCH.
WAITFOR	Блокирует выполнение пакета, хранимой процедуры или транзакции до наступления указанного времени или интервала времени, либо заданная инструкция изменяет или возвращает, по крайней мере, одну строку.
WHILE	Ставит условие повторного выполнения SQL-инструкции или блока инструкций. Эти инструкции вызываются в цикле, пока указанное условие истинно. Вызовами инструкций в цикле WHILE можно контролировать из цикла с помощью ключевых слов BREAK и CONTINUE.

7. Системы типов данных языка SQL.

Типы данных SQL разделяются на три группы:

- Строковые.
- С плавающей точкой (дробные числа).
- Целые числа, дата и время.

1. **Типы данных SQL** строковые

Типы данных SQL	Описание
`CHAR(size)`	Строки фиксированной длиной (могут содержать буквы, цифры и специальные символы). Фиксированный размер указан в скобках. Можно записать до 255 символов
`VARCHAR(size)`	Может хранить не более 255 символов.
`TINYTEXT`	Может хранить не более 255 символов.
`TEXT`	Может хранить не более 65 535 символов.
`BLOB`	Может хранить не более 65 535 символов.
`MEDIUMTEXT`	Может хранить не более 16 777 215 символов.
`MEDIUMBLOB`	Может хранить не более 16 777 215 символов.
`LONGTEXT`	Может хранить не более 4 294 967 295 символов.
`LONGBLOB`	Может хранить не более 4 294 967 295 символов.
`ENUM(x,y,z,etc.)`	Позволяет вводить список допустимых значений. Можно ввести до 65535 значений в **SQL Тип данных ENUM** список. Если при вставке значения не будет присутствовать в списке *ENUM*, то мы получим пустое значение. Ввести возможные значения можно в таком формате: `ENUM ( 'X', 'Y', 'Z')`
`SET`	**SQL Тип данных SET** напоминает *ENUM* за исключением того, что *SET* может содержать до 64 значений.

2. **Типы данных SQL** с плавающей точкой (дробные числа) и целые числа

Типы данных SQL	Описание
`TINYINT(size)`	Может хранить числа от -128 до 127
`SMALLINT(size)`	Диапазон от -32 768 до 32 767
`MEDIUMINT(size)`	Диапазон от -8 388 608 до 8 388 607
`INT(size)`	Диапазон от -2 147 483 648 до 2 147 483 647
`BIGINT(size)`	Диапазон от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
`FLOAT(size,d)`	Число с плавающей точкой небольшой точности.
`DOUBLE(size,d)`	Число с плавающей точкой двойной точности.
`DECIMAL(size,d)`	Дробное число, хранящееся в виде строки.

3. **Типы данных SQL** — Дата и время

Типы данных SQL	Описание
`DATE()`	Дата в формате ГГГГ-ММ-ДД
`DATETIME()`	Дата и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`
`TIMESTAMP()`	Дата и время в формате timestamp. Однако при получении значения поля оно отображается не в формате timestamp, а в виде ГГГГ-ММ-ДД ЧЧ:ММ:СС
`TIME()`	Время в формате `ЧЧ:ММ:СС`
`YEAR()`	Год в двух значной или в четырехзначном формате.

Типы данных в PostgreSQL.

Имя	Псевдонимы	Описание
bigint	int8	знаковое целое из 8 байт
bigserial	serial8	восьмибайтное целое с автоувеличением
bit [(*n*)]		битовая строка фиксированной длины
bit varying [(*n*)]	varbit [(*n*)]	битовая строка переменной длины
boolean	bool	логическое значение (true/false)
box		прямоугольник в плоскости
bytea		двоичные данные («массив байт»)
character [(*n*)]	char [(*n*)]	символьная строка фиксированной длины
character varying [(*n*)]	varchar [(*n*)]	символьная строка переменной длины
cidr		сетевой адрес IPv4 или IPv6
circle		круг в плоскости
date		календарная дата (год, месяц, день)
double precision	float8	число двойной точности с плавающей точкой (8 байт)
inet		адрес узла IPv4 или IPv6
integer	int, int4	знаковое четырёхбайтное целое
interval [*поля*] [(*p*)]		интервал времени
json		текстовые данные JSON
jsonb		двоичные данные JSON, разобранные
line		прямая в плоскости
lseg		отрезок в плоскости
macaddr		MAC-адрес
macaddr8		Адрес MAC (Media Access Control) (в формате EUI-64)
money		денежная сумма
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	вещественное число заданной точности
path		геометрический путь в плоскости
pg_lsn		Последовательный номер в журнале Postgres Pro
point		геометрическая точка в плоскости
polygon		замкнутый геометрический путь в плоскости
real	float4	число одинарной точности с плавающей точкой (4 байта)
smallint	int2	знаковое двухбайтное целое
smallserial	serial2	двухбайтное целое с автоувеличением

Имя	Псевдонимы	Описание
serial	serial4	четырёхбайтное целое с автоувеличением
text		символьная строка переменной длины
time [(*p*)] [without time zone]		время суток (без часового пояса)
time [(*p*)] with time zone	timetz	время суток с учётом часового пояса
timestamp [(*p*)] [without time zone]		дата и время (без часового пояса)
timestamp [(*p*)] with time zone	timestamptz	дата и время с учётом часового пояса
tsquery		запрос текстового поиска
tsvector		документ для текстового поиска
txid_snapshot		снимок идентификатора транзакций
uuid		универсальный уникальный идентификатор
xml		XML-данные

8. Скалярные выражения.

Скалярное выражение – это выражение, вырабатывающее результат некоторого типа, специфицированного в стандарте. Скалярные выражения являются основой языка SQL, поскольку, хотя это реляционный язык, все условия, элементы списков выборки и т. д. базируются именно на скалярных выражениях. В SQL:1999 имеется несколько разновидностей скалярных выражений. К числу наиболее важных разновидностей относятся численные выражения; выражения со значениями-строками символов; выражения со значениями даты-времени; выражения со значениями-временными интервалами; булевские выражения.

Скалярными называют те функции, которые возвращают одно значение. Эти функции могут принимать множество параметров, выполнять вычисления, но в результате выдают одно значение. Эти функции могут использоваться в любых выражениях, даже участвующих в ограничениях проверки. Значение возвращается функцией с помощью оператора return — эта команда должна завершать скалярную функцию.

В скалярных пользовательских функциях не допускаются операции обновления базы данных, но в то же время они могут работать с локальными временными таблицами. Они не могут возвращать данные BLOB (двоичные большие объекты) таких типов, как text, image и ntext, равно как табличные переменные и курсоры.

```
CREATE FUNCTION add(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;
```

Функция увеличения целого числа на 1, использующая именованный аргумент, на языке PL/pgSQL:

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;
```

9. Приведение и преобразование типов данных.

SQL-операторы, намеренно или нет, требуют совмещать данные разных типов в одном выражении. Для вычисления подобных выражений со смешанными типами PostgreSQL предоставляет широкий набор возможностей.

Однако следует учитывать, что неявные преобразования, производимые PostgreSQL, могут влиять на результат запроса. Поэтому при необходимости нужные результаты можно получить, применив явное преобразование типов.

SQL — язык со строгой типизацией.

В SQL есть четыре фундаментальных фактора, определяющих правила преобразования типов для анализатора выражений PostgreSQL:

- **Вызовы функций**

Система типов PostgreSQL во многом построена как дополнение к богатым возможностям функций. Функции могут иметь один или несколько аргументов, и при этом PostgreSQL разрешает перегружать имена функций, так что имя функции само по себе не идентифицирует вызываемую функцию; анализатор выбирает правильную функцию в зависимости от типов переданных аргументов.

- **Операторы**

PostgreSQL позволяет использовать в выражениях префиксные и постфиксные операторы с одним аргументом, а также операторы с двумя аргументами. Как и функции, операторы можно перегружать, так что и с ними существует проблема выбора правильного оператора.

- **Сохранение значений**

SQL-операторы INSERT и UPDATE помещают результаты выражений в таблицы. При этом получаемые значения должны соответствовать типам целевых столбцов или, возможно, приводиться к ним.

- **UNION, CASE и связанные конструкции**

Так как все результаты запроса объединяющего оператора SELECT должны оказаться в одном наборе столбцов, результаты каждого подзапроса SELECT должны приводиться к одному набору типов. Подобным образом, результирующие выражения конструкции CASE должны приводиться к общему типу, так как выражение CASE в целом должно иметь определённый выходной тип. То же справедливо в отношении конструкций ARRAY и функций GREATEST и LEAST.

Информация о существующих преобразованиях или приведениях типов, для каких типов они определены и как их выполнять, хранится в системных каталогах. Пользователь также может добавить дополнительные преобразования с помощью команды CREATE CAST. Обычно это делается, когда определяются новые типы данных. Набор приведений для встроенных типов достаточно хорошо проработан, так что его лучше не менять.

```
CREATE CAST (исходный_тип AS целевой_тип)
    WITH FUNCTION имя_функции (тип_аргумента [, ...])
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (исходный_тип AS целевой_тип)
    WITHOUT FUNCTION
    [ AS ASSIGNMENT | AS IMPLICIT ]

CREATE CAST (исходный_тип AS целевой_тип)
    WITH INOUT
    [ AS ASSIGNMENT | AS IMPLICIT ]
```

Описание

`CREATE CAST` создаёт новое приведение. Приведение определяет, как выполнить преобразование из одного типа в другой. Например,

```
SELECT CAST(42 AS float8);
```

преобразует целочисленную константу 42 к типу `float8`, вызывая ранее определённую функцию, в данном случае `float8(int4)`. (Если подходящее приведение не определено, возникнет ошибка преобразования.)

10. Условие поиска.

Необязательное предложение WHERE имеет общую форму: *WHERE условие*, где условие — любое выражение, выдающее результат типа boolean. Любая строка, не удовлетворяющая этому условию, исключается из результата. Строка удовлетворяет условию, если оно возвращает true при подстановке вместо ссылок на переменные фактических значений из этой строки.

	Описание
<search_condition>	Задает условия для строк, возвращаемых в результирующем наборе инструкции SELECT, выражения запроса или вложенного запроса. Задает обновляемые строки для инструкции UPDATE. Задает удаляемые строки для инструкции DELETE. Количество предикатов, которое может содержаться в условии поиска для инструкции Transact-SQL, не ограничено.
NOT	Инвертирует логическое выражение, задаваемое предикатом
AND	Объединяет два условия и выдает значение TRUE, если оба условия имеют значение TRUE.
OR	Объединяет два условия и выдает значение TRUE, если хотя бы одно имеет значение TRUE.

	Описание
< предикат >	Выражение, возвращающее значения TRUE, FALSE или UNKNOWN.
expression	Может являться именем столбца, константой, функцией, переменной, скалярным вложенным запросом или любым сочетанием имен столбцов, констант и функций, связанных операторами или вложенным запросом. Также может содержать выражение CASE.
=	Оператор, используемый для проверки равенства двух выражений
<>	Оператор, используемый для проверки условий неравенства условий двух выражений.
!=	Оператор, используемый для проверки условий неравенства условий двух выражений.
>	Оператор, используемый для проверки превышения одного выражения над условием другого.
>=	Оператор, используемый для проверки превышения либо равенства двух выражений.
!>	Оператор, используемый для проверки того, что одно выражение не превышает другое выражение.
>	Оператор, используемый для проверки того, что одно выражение меньше другого.
<=	Оператор, используемый для проверки того, что одно выражение меньше или равно другому.
!<	Оператор, используемый для проверки того, что одно выражение не меньше другого.
string_expression	Строка обычных символов и символов-шаблонов
[NOT] LIKE	Показывает, что обрабатываемая строка должна использоваться при совпадении с шаблоном
ESCAPE 'escape_ character'	Позволяет найти сам символ-шаблон в строке (вместо того чтобы использовать его как шаблон). escape_character — это символ, который нужно поместить перед символом-шаблоном, чтобы указать данное специальное использование.
[NOT] BETWEEN	Задает включающий диапазон значений. Используйте оператор AND для разделения начальных и конечных значений
IS [NOT] NULL	Задает поиск значений NULL или значений, не являющихся значениями NULL, в зависимости от используемых ключевых слов. При обращении одного из операндов выражения с битовыми или арифметическими операторами в значение NULL указанное выражение также обращается в значение NULL
CONTAINS	Осуществляет поиск столбцов, содержащих символьные данные с заданной точностью (fuzzy), соответствующие заданным отдельным словам и фразам на основе похожести словам и точному расстоянию между словами, взвешенному совпадению. Этот параметр может быть использован только в инструкции SELECT
FREETEXT	Предоставляет простую форму естественного языка ввода запросов на осуществление поиска столбцов, содержащих символьные данные, совпадающие с содержанием предиката не точно, а по

	Описание
	смыслу. Этот параметр может быть использован только в инструкции SELECT
[NOT] IN	Задает поиск выражения, основанного на выражении, включенного или исключенного из списка. Выражение поиска может быть константой или именем столбца, а списком может быть набор констант или, что чаще, вложенный запрос. Список значений необходимо заключать в скобки
subquery	Может рассматриваться как ограниченная инструкция SELECT и являющаяся подобной на <query_expresssion> в инструкции SELECT. Использование предложения ORDER BY и ключевого слова INTO не допускается
ALL	Используется с оператором сравнения и вложенным запросом. Возвращает для <предиката> значение TRUE, если все получаемые для вложенного запроса значения удовлетворяют условию, и значение FALSE, если не все значения удовлетворяют условию или в случае, когда в результате выполнения вложенного запроса внешней инструкции не выдается ни одной строки
{ SOME | ANY }	Используется с оператором сравнения и вложенным запросом. Возвращает для <предиката> значение TRUE, если хотя бы одно получаемое для вложенного запроса значение удовлетворяет условию, и значение FALSE, если ни одно из значений не удовлетворяет условию или в случае, когда в результате выполнения вложенного запроса внешней инструкции не выдается ни одной строки. В противном случае результатом выражения является значение UNKNOWN
EXISTS	Используется во вложенном запросе для проверки существования строк, возвращенных вложенным запросом

11. Сценарии и пакеты.

Пакеты позволяют объединить процедуры и функции в один автономный модуль. Обычно пакеты состоят из двух компонентов: спецификации (интерфейс) и тела (реализации интерфейса).

Спецификация пакета содержит информацию о пакете, кроме того, в ней перечисляются все имеющиеся в пакете процедуры и функции. Обычно спецификация не содержит кода, код помещается в теле пакета. Процедуры и функции, перечисленные в спецификации доступны для просмотра, а реализация скрыта.

Создание спецификации пакета

```
CREATE [OR REPLACE] PACKAGE package_name
IS | AS
спецификация пакета
END имя_пакета;
```

Спецификация пакета – список доступных процедур и функций (вместе со всеми переменными, определениями типов и курсорами).

Создание тела пакета

```
CREATE [OR REPLACE] PACKAGE BODY имя_пакета
IS | AS
тело_пакета
END имя пакета
```

Вызов процедур и функций в пакете

При вызове входящих в пакет процедур и функций, следует включать в выхов имя пакета.

```
package_name.stored_procedure([parameters]);
```

Получение информации о процедурах и функциях из пакета

```
SELECT ojbect_name, aggregate, parallel
FROM user_procedures
WHERE objectt_name = 'package';
```

Удаление пакета

```
DROP PACKAGE  package_name;
```

Сценарий — это серия инструкций языка Transact-SQL, которая хранится в файле. Данные из этого файла можно использовать в качестве исходных для редактора кода среды Среда SQL Server Management Studio или программ sqlcmd и osql. Эти программы будут выполнять инструкции SQL из файла. Сценарии языка Transact-SQL содержат один или несколько пакетов. Команда GO означает конец пакета. Если сценарий языка Transact-SQL не содержит команд GO, то он выполняется как единый пакет. Сценарии языка Transact-SQL можно использовать следующим образом:

- Сохраните как механизм резервной копии постоянную копию шагов, которые выполнялись при создании и заполнении базы данных на сервере.
- Перенесите инструкции с одного компьютера на другой, когда появится соответствующее приглашение.
- Быстро обучите новых работников разбираться в коде, изменять код и находить в нем ошибки.

12. Массовый импорт и экспорт данных.

Массовый экспорт означает копирование данных из таблицы SQL Server в файл данных.

Массовый импорт означает загрузку данных из файла данных в таблицу SQL Server. Например, можно экспортировать данные из приложения Microsoft Excel в файл данных, а затем выполнить массовый импорт данных в таблицу SQL Server.

COPY перемещает данные между таблицами PostgreSQL и традиционными файлами. 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 колонок, COPY скопирует в или из файла только данные указанных колонок. Если в таблице есть колонки, отсутствующие в этом списке, COPY FROM заполнит эти колонки значениями по умолчанию.

COPY с именем файла указывает серверу PostgreSQL читать или записывать непосредственно этот файл. Заданный файл должен быть доступен пользователю PostgreSQL (тому пользователю, от имени которого работает сервер) и это имя должно быть определено 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 PostgreSQL. Когда указывается STDIN или STDOUT, данные передаются через соединение клиента с сервером.

Инструкция BULK INSERT загружает данные из файла данных в таблицу. В T-SQL выполняет импорт файла данных в таблицу или представление базы данных в формате, указанном пользователем, в SQL Server.

```
BULK INSERT Sales.Orders
FROM '\\SystemX\DiskZ\Sales\data\orders.dat';

BULK INSERT Sales.Orders
FROM '\\SystemX\DiskZ\Sales\data\orders.csv'
WITH ( FORMAT='CSV');
```

Инструкция INSERT ...SELECT * FROM OPENROWSET(BULK...). (импортирует)- Инструкция Transact-SQL, использующая поставщик больших наборов строк OPENROWSET для массового импорта данных в таблицу SQL Server с помощью функции OPENROWSET(BULK...), применяющейся для

выборки данных в предложение INSERT. Содержит все необходимые сведения о соединении, которые требуются для доступа к удаленным данным источника данных OLE DB. Это альтернативный метод для доступа к таблицам на связанном сервере и является однократным нерегламентированным методом соединения и удаленного доступа к данным с помощью OLE DB.

13. Представления, функции, хранимые триггеры (DDL, DML), курсоры.

Представление — виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления.

В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе. Содержимое представления динамически вычисляется на основании данных, находящихся в реальных таблицах. Изменение данных в реальной таблице базы данных немедленно отражается в содержимом всех представлений, построенных на основании этой таблицы.

Типичным способом создания представлений для СУБД, поддерживающих язык запросов SQL, является связывание представления с определённым SQL-запросом.

Представления используются в запросах к БД тем же образом, как и обычные таблицы.

Преимущества представлений:

- Представления скрывают от прикладной программы сложность запросов и саму структуру таблиц БД. Когда прикладной программе требуется таблица с определённым набором данных, она делает простейший запрос из подготовленного представления.
- Использование представлений позволяет отделить прикладную схему представления данных от схемы хранения. С точки зрения прикладной программы структура данных соответствует тем представлениям, из которых программа эти данные извлекает. В действительности данные могут храниться совершенно иным образом, достаточно лишь создать представления, отвечающие потребностям программы. Разделение позволяет независимо модифицировать прикладную программу и схему хранения данных: как при изменении структуры физических таблиц, так и при изменении программы достаточно изменить представления соответствующим образом. Изменение программы не затрагивает физические таблицы, а изменение физической структуры таблиц не требует корректировки программы.
- С помощью представлений обеспечивается ещё один уровень защиты данных. Пользователю могут предоставляться

права только на представление, благодаря чему он не будет иметь доступа к данным, находящимся в тех же таблицах, но не предназначенных для него.

- Поскольку SQL-запрос, выбирающий данные представления, зафиксирован на момент его создания, СУБД получает возможность применить к этому запросу оптимизацию или предварительную компиляцию, что положительно сказывается на скорости обращения к представлению, по сравнению с прямым выполнением того же запроса из прикладной программы.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW имя [ ( имя_столбца [, ...] ) ]
    [ WITH ( имя_параметра_представления [= значение_параметра_представления] [, ... ] ) ]
    AS запрос
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Для удаления представлений применяется оператор DROP VIEW.

Простые представления становятся изменяемыми автоматически: система позволит выполнять команды INSERT, UPDATE и DELETE с таким представлением так же, как и с обычной таблицей. Представление будет автоматически изменяемым, если оно удовлетворяют одновременно всем следующим условиям:

- Список FROM в запросе, определяющем представлении, должен содержать ровно один элемент, и это должна быть таблица или другое изменяемое представление.
- Определение представления не должно содержать предложения WITH, DISTINCT, GROUP BY, HAVING, LIMIT и OFFSET на верхнем уровне запроса.
- Определение представления не должно содержать операции с множествами (UNION, INTERSECT и EXCEPT) на верхнем уровне запроса.
- Список выборки в запросе не должен содержать агрегатные и оконные функции, а также функции, возвращающие множества.

Создание представления, содержащего все комедийные фильмы:

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Функции:

1. Скалярные
   CREATE FUNCTION count_specialities() RETURNS INTEGER AS $$
   DECLARE speciality_count integer := (SELECT COUNT(*) FROM (SELECT DISTINCT d.speciality FROM doctors d) AS AllSpec);
   BEGIN
   RETURN speciality_count;
   END;
   $$ LANGUAGE plpgsql;

   SELECT count_specialities();
2. Табличные
   CREATE FUNCTION avg_salary_high() RETURNS TABLE (speciality text, avg_salary numeric) AS $$
   (
   SELECT
   d.speciality,
   AVG(d.salary)
   FROM doctors d WHERE d.category= 'Высшая'
   GROUP BY d.speciality
)
   $$ LANGUAGE SQL;

   *SELECT * FROM avg_salary_high();*
3. Многооператроные

Любой набор команд на языке SQL можно скомпоновать вместе и обозначить как функцию. Помимо запросов SELECT, эти команды могут включать запросы, изменяющие данные (INSERT, UPDATE и DELETE), а также другие SQL-команды. (В SQL-функциях нельзя использовать команды управления транзакциями, например COMMIT, SAVEPOINT, и некоторые вспомогательные команды, в частности VACUUM.) Однако последней командой должна быть SELECT или команда с предложением RETURNING, возвращающая результат с типом возврата функции. Если же вы хотите определить функцию SQL, выполняющую действия, но не возвращающую полезное значение, вы можете объявить её как возвращающую тип void.

К аргументам SQL-функции можно обращаться в теле функции по именам или номерам.

Учтите два важных требования относительно определения функции:

- Порядок в списке выборки внутреннего запроса должен в точности совпадать с порядком следования колонок в таблице, связанной с составным типом. (Имена колонок, как показывает пример выше, для системы значения не имеют).

- Вы должны привести выражения в соответствие с определением составного типа, либо вы получите ошибки.

Триггер – набор действий, который выполняется при наступлении какого-либо события в БД. Триггер – объект БД.

- DML –реагируют на INSERT, DELETE, UPDATE.
- DDL – реагируют на CREATE, DROP.

Мягкое удаление – запись на самом деле не удаляется, а помечается как удаленная и остаётся в таблице.

В PL/pgSQL можно создавать триггерные процедуры, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная процедура создаётся командой CREATE FUNCTION, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event_trigger (для триггеров, срабатывающих при событиях в базе). Для триггеров автоматически определяются специальные локальные переменные с именами вида TG_имя, описывающие условие, повлёкшее вызов триггера.

CREATE TRIGGER — создать триггер. CREATE TRIGGER создаёт новый триггер. Триггер будет связан с указанной таблицей, представлением или сторонней таблицей и будет выполнять заданную функцию имя_функции при определённых событиях.

```
CREATE [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [ OR ... ] }
    ON имя_таблицы
    [ FROM ссылающаяся_таблица ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] [ INITIALLY IMMEDIATE | INITIALLY DEFERRED ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( условие ) ]
    EXECUTE PROCEDURE имя_функции ( аргументы )

Здесь допускается событие:

    INSERT
    UPDATE [ OF имя_столбца [, ... ] ]
    DELETE
    TRUNCATE
```

-- DML триггер INSTEAD OF
*CREATE VIEW doctors_view AS SELECT * FROM doctors;*

CREATE FUNCTION insteadof_doctors_delete() RETURNS TRIGGER AS $$
DECLARE d integer := OLD.id;
BEGIN
RAISE NOTICE 'Deleted id: %', d;
DELETE FROM doctors WHERE id = OLD.id;
RETURN NEW;
END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER BDeleteDoctors
INSTEAD OF DELETE
ON doctors_view FOR EACH ROW
EXECUTE PROCEDURE insteadof_doctors_delete();

Триггер с пометкой FOR EACH ROW вызывается один раз для каждой строки, изменяемой в процессе операции. Например, операция DELETE, удаляющая 10 строк, приведёт к срабатыванию всех триггеров ON DELETE в целевом отношении 10 раз подряд, по одному разу для каждой удаляемой строки. Триггер с пометкой FOR EACH STATEMENT, напротив, вызывается только один раз для конкретной операции, вне зависимости от того, как много строк она изменила.

Для удаления триггера применяется команда DROP TRIGGER.

Курсор – указатель на область памяти. Курсор — это поименованная область памяти, содержащая результирующий набор select запроса. Второе определение – это механизм обработки результирующего набора select запроса.

Вместо того чтобы сразу выполнять весь запрос, есть возможность настроить курсор, инкапсулирующий запрос, и затем получать результат запроса по нескольку строк за раз. Одна из причин так делать заключается в том, чтобы избежать переполнения памяти, когда результат содержит большое количество строк.

В PL/SQL поддерживаются два типа курсоров:

- явный — объявляется разработчиком;
- неявный — не требует объявления.

В соответствии со стандартом SQL при работе с курсорами можно выделить следующие основные действия:

- создание или объявление курсора;
- открытие курсора, т. е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- выборка из курсора и изменение с его помощью строк данных;
- закрытие курсора, после чего он становится недоступным для пользовательских программ;
- освобождение курсора, т. е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

Доступ к курсорам в PL/pgSQL осуществляется через курсорные переменные, которые всегда имеют специальный тип данных refcursor. Один из способов создать курсорную переменную, просто объявить её как переменную типа refcursor. Другой способ заключается в использовании синтаксиса объявления курсора, который в общем виде выглядит так:

имя [[NO] SCROLL] CURSOR [(аргументы)] FOR запрос;

С указанием SCROLL курсор можно будет прокручивать назад. При NO SCROLL прокрутка назад не разрешается.

Прежде чем получать строки из курсора, его нужно открыть. После того как курсор будет открыт, с ним можно работать при помощи операторов:

- FETCH
  FETCH [направление { FROM | IN }] курсор INTO цель;
  FETCH извлекает следующую строку из курсора в цель. В качестве цели может быть строковая переменная, переменная типа record, или разделённый запятыми список простых переменных, как и в SELECT INTO.
- MOVE
  MOVE [направление { FROM | IN }] курсор;
  MOVE перемещает курсор без извлечения данных.
- UPDATE/DELETE WHERE CURRENT OF
  UPDATE таблица SET ... WHERE CURRENT OF курсор;
  DELETE FROM таблица WHERE CURRENT OF курсор;
  Когда курсор позиционирован на строку таблицы, эту строку можно изменить или удалить при помощи курсора.
- CLOSE
  CLOSE курсор;
  CLOSE закрывает связанный с курсором портал. Используется для того, чтобы освободить ресурсы раньше, чем закончится транзакция, или чтобы освободить курсорную переменную для повторного открытия.

Курсоры можно возвращать из функции на PL/pgSQL. Это полезно, когда нужно вернуть множество строк и столбцов, особенно если выборки очень большие.

```
--Хранимая процедура с курсором
CREATE OR REPLACE PROCEDURE count_specialities(_cat _CATEGORY, _spec text) AS $$
    DECLARE
        high_doctors_cursor NO SCROLL CURSOR FOR SELECT d.id FROM doctors d WHERE d.category = _cat AND d.speciality = _spec;
        cnt integer := 0;
        rec_doctors RECORD;
    BEGIN
        OPEN high_doctors_cursor;
```

```
                LOOP
                    FETCH high_doctors_cursor INTO rec_doctors;
                    IF NOT FOUND THEN EXIT;END IF;
                    cnt = cnt + 1;
                END LOOP;
                RAISE NOTICE 'Врачей специальности "%" категории "%" всего: %', _spec, _cat, cnt;
                CLOSE high_doctors_cursor;
            END;
        $$ LANGUAGE plpgsql;

        CALL count_specialities('Высшая', 'Валеолог');
```

14. Метаданные и доступ к ним.

Метаданные — информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте. Метаданные раскрывают сведения о признаках и свойствах, характеризующих какие-либо сущности, позволяющие автоматически искать и управлять ими в больших информационных потоках.

Каждая СУБД сохраняет метаданные (данные о данных) - детальную информацию обо всех объектах системы. Примерами таких объектов могут служить таблицы, представления, ограничения целостности, триггеры, правила безопасности и т.д. В разных СУБД применяются даже разные названия для метаданных - системный каталог в DB2 или словарь данных (Oracle). Однако общим свойством всех современных реляционных СУБД является то, что каталог/словарь сам состоит из таблиц. В результате пользователь может обращаться к метаданным так же, как и к своим данным - используя оператор SQL SELECT. Изменения же в каталоге/словаре производятся автоматически при выполнении пользователем операторов SQL, изменяющих состояние объектов базы данных.

Системные каталоги — это место, где система управления реляционной базой данных хранит метаданные схемы, в частности информацию о таблицах и столбцах, а также служебные сведения.

Большинство системных каталогов копируются из базы-шаблона при создании базы данных и затем принадлежат этой базе. Но некоторые каталоги физически разделяются всеми базами данных в кластере.

Имя каталога	Предназначение
pg_aggregate	агрегатные функции
pg_am	индексные методы доступа
pg_amop	операторы методов доступа
pg_amproc	опорные процедуры методов доступа
pg_attrdef	значения столбцов по умолчанию
pg_attribute	столбцы таблиц («атрибуты»)
pg_authid	идентификаторы для авторизации (роли)

Имя каталога	Предназначение
pg_auth_members	отношения членства для объектов авторизации
pg_cast	приведения (преобразования типов данных)
pg_class	таблицы, индексы, последовательности, представления («отношения»)

В дополнение к системным каталогам, в PostgreSQL есть набор встроенных представлений. Некоторые системные представления содержат в себе некоторые популярные запросы к системным каталогам, а другие дают доступ к внутреннему состоянию сервера.

Имя представления	Предназначение
pg_available_extensions	доступные расширения
pg_available_extension_versions	доступные версии расширений
pg_config	параметры конфигурации времени компиляции
pg_cursors	открытые курсоры
pg_file_settings	сводка содержимого файла конфигурации
pg_group	группы пользователей баз данных
pg_indexes	индексы

Информационная схема состоит из набора представлений, содержащих информацию об объектах, определённых в текущей базе данных. Информационная схема описана в стандарте SQL и поэтому можно рассчитывать на её переносимость и стабильность — в отличие от системных каталогов, которые привязаны к PostgreSQL, и моделируются, отталкиваясь от реализации. Представления информационной схемы, однако, не содержат информацию о функциях, присущих исключительно PostgreSQL; чтобы получить информацию о них, необходимо обратиться к системным каталогам или другим специфическим представлениям PostgreSQL.

Информационная схема сама по себе — это схема с именем information_schema. Данная схема автоматически доступна во всех базах данных.

Столбцы в представлениях информационной схемы имеют специальные типы данных, определённые в информационной схеме. Они определены как простые домены поверх обычных встроенных типов. Задействовать эти типы вне информационной схемы не следует, но тем не менее, приложения, выбирающие данные из информационной схеме должны быть готовы работать с ними.

Таблица information_schema_catalog_name всегда содержит одну строку и один столбец с именем текущей базы данных (текущий каталог, в терминологии SQL).

Представление administrable_role_authorizations описывает все роли, для которых текущий пользователь является администратором.

Представление columns содержит информацию обо всех столбцах таблиц (или столбцах представлений) в базе данных. Системные столбцы (oid

и т. д.) в нём не отображаются. В нём показываются только те столбцы, к которым имеет доступ текущий пользователь (являясь владельцем или имея некоторые права).

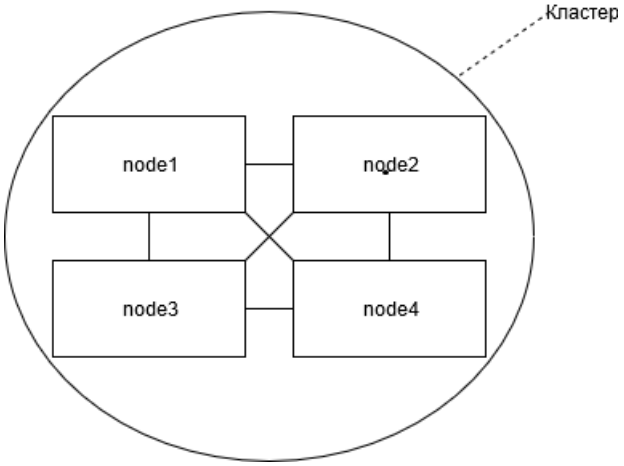
```
create or replace function indexes_info(_tablename text) returns table(index_name name, indexdef text) as $$ (
    SELECT indexname, indexdef FROM pg_indexes WHERE tablename = _tablename
)
$$ language sql;
select * from indexes_info('subjects');
```

15. Методы физического хранения данных на диске.

Основными единицами физического хранения являются блок данных, экстент, файл (либо раздел жесткого диска).

- Блок данных (block) или страница (page) является единицей обмена с внешней памятью. Размер страницы фиксирован для базы данных или для ее различных структур и устанавливается при создании. Очень важно сразу правильно выбрать размер блока: в работающей базе изменить его практически невозможно.
- Администратор отводит пространство для базы данных на внешних устройствах большими фрагментами: файлами и разделами диска. В первом случае доступ к диску осуществляется операционной системой, что дает определенные преимущества, например, работа с файлами средствами ОС. Во втором случае с внешним устройством работает сам сервер.
- Пространством внешней памяти, отведенным ему администратором, сервер управляет с помощью экстентов (extent), т.е. непрерывных последовательностей блоков. Информация о наличии экстентов для объекта схемы данных находится в специальных управляющих структурах, реализация которых зависит от СУБД. На управление экстентами (выделение пространства, освобождение, слияние) тратятся определенные ресурсы, поэтому для достижения эффективности нужно правильно определять их параметры. Уменьшение размера экстента будет способствовать более эффективному использованию памяти, однако при этом возрастают накладные расходы на управление большим количеством экстентов, что может замедлить операции вставки большого количества строк в таблицу.

Разложение по сегментам в MPP-системах



MyTable

id	Date	Item	Hash(d)
1	2018.01.01	A	1
2	2019.05.04	B	2
3	2016.03.08	C	3
4	2018.04.05	D	0
5		A	1
6		A	2
7		A	3
8		B	0
9		B	1
10		C	2
11		C	3
12		A	0
13		D	1
14		D	2
15		B	3
16		C	0

Node1

id	date	item
1	…	A
5	…	A
9	…	B
13	…	D

Node2

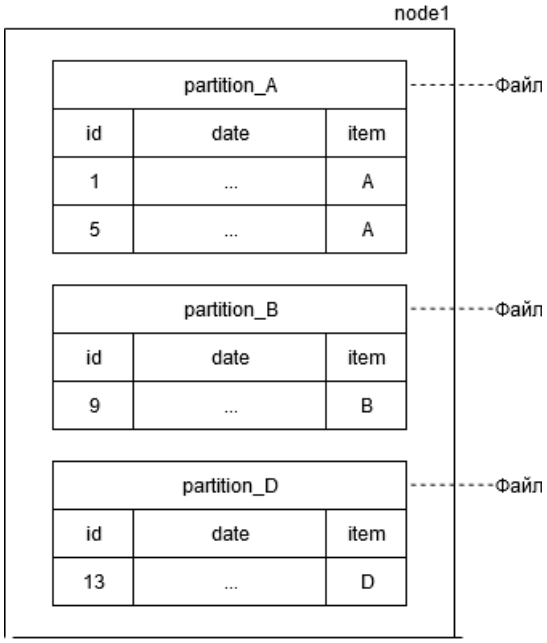
id	date	item
2	…	B
6	…	A
10	…	C
14	…	D

Node3

id	date	item
3	…	C
7	…	A
11	…	C
15	…	B

Node4

id	date	item
4	…	D
8	…	B
12	…	A
16	…	C



CREATE TABLE MyTable (id int, date date, item varchar(1))
SEGMENTED BY Hash(id)
PARTITION BY year(date);

16. Индексы.

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.

Индексы — это традиционное средство увеличения производительности БД. Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него. Однако с индексами связана дополнительная нагрузка на СУБД в целом, поэтому применять их следует обдуманно.

+ O(log n)
− Дополнительное место
− Перебалансировка при каждом INSERT и DELETE

Индексы:

- Кластеризованные – в листьях лежат данные.
- Некластеризованные – хранят ссылку на данные.

Между элементами дерева существует горизонтальное соединение.

Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице.

PostgreSQL поддерживает несколько типов индексов: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов. По умолчанию команда CREATE INDEX создаёт индексы типа B-дерево, эффективные в большинстве случаев.

CREATE INDEX — создать индекс.

CREATE INDEX index_name ON table_name (column_name);

Иногда индекс может выйти из строя и для нормальной работы его необходимо пересоздать.

REINDEX INDEX index_name; / Пересоздаст индекс index_name */*

REINDEX TABLE table_name; / Пересоздаст все индексы в таблице table_name */*

REINDEX DATABASE database_name; / Пересоздаст все индексы в базе database_name */*

Один индекс может поддерживать только одно правило сортировки для индексируемого столбца. Поэтому при необходимости применять разные правила сортировки могут потребоваться несколько индексов.

DROP INDEX удаляет существующий индекс из базы данных. Выполнить эту команду может только владелец индекса.

DROP INDEX title_idx;

17. Управление транзакциями.

Транзакция — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакции — это фундаментальное понятие во всех СУБД. Суть транзакции в том, что она объединяет последовательность действий в одну операцию "всё или ничего". Промежуточные состояния внутри последовательности не видны другим транзакциям, и если что-то помешает успешно завершить транзакцию, ни один из результатов этих действий не сохранится в базе данных.

В Postgres Pro транзакция определяется набором SQL-команд, окружённым командами BEGIN и COMMIT. Если в процессе выполнения транзакции мы решим, что не хотим фиксировать её изменения (например, потому что оказалось, что баланс Алисы стал отрицательным), мы можем выполнить команду ROLLBACK вместо COMMIT, и все наши изменения будут отменены.

Postgres Pro на самом деле отрабатывает каждый SQL-оператор как транзакцию. Если вы не вставите команду BEGIN, то каждый отдельный оператор будет неявно окружён командами BEGIN и COMMIT (в случае успешного завершения).

Операторами в транзакции можно также управлять на более детальном уровне, используя точки сохранения. Точки сохранения позволяют выборочно отменять некоторые части транзакции и фиксировать все остальные. Определив точку сохранения с помощью SAVEPOINT, при необходимости вы можете вернуться к ней с помощью команды ROLLBACK TO. Все изменения в базе данных, произошедшие после точки сохранения и до момента отката, отменяются, но изменения, произведённые ранее, сохраняются.

У любой транзакции есть четыре свойства (ACID):

1. A – атомарность – все действия в рамках транзакции выполняются вместе, как единое целое.
2. C – консистентность (согласованность) – каждая транзакция приводит БД в правильное логическое состояние.
3. I – изолированность – никакая транзакция извне не знает, что делает другая транзакция (пока она не подтвердила свои действия).
4. D – долговечность (устойчивость) – изменения в рамках подтвержденной транзакции сохраняются при любых обстоятельствах.

По умолчанию каждая инструкция – отдельная транзакция.

Виды транзакций:

- Автоматические
- Неявные
- Явные

Операции над транзакциями:

- Begin transaction
- Commit transaction
- Rollback transaction
- Save transaction

Управление транзакциями:

- Оптимистичное
- Пессимистичное

Пессимистичный подход работает на блокировках. В MS SQL Server каждые 5 минут запускается поиск блокировок.

Оптимистичный подход делает копии.

18-19. Неблагоприятные эффекты, вызванные параллельным выполнением транзакций, и их устранение.

Проблемы параллельных процессов:

1. Проблема последнего изменения (потерянное обновление) (level 0) – при одновременном изменении одного блока данных разными транзакциями теряются все изменения, кроме последнего.

Транзакция 1	Транзакция 2
`UPDATE tbl1 SET f2=f2+20 WHERE f1=1;`	`UPDATE tbl1 SET f2=f2+25 WHERE f1=1;`

2. Проблема грязного чтения (level 1) – чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится).

Транзакция 1	Транзакция 2
`UPDATE tbl1 SET f2=f2+1 WHERE f1=1;`	
	`SELECT f2 FROM tbl1 WHERE f1=1;`
`ROLLBACK WORK;`	

3. Проблема неповторного чтения (level 2) – при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными.

Транзакция 1	Транзакция 2
	SELECT f2 FROM tbl1 WHERE f1=1;
UPDATE tbl1 SET f2=f2+1 WHERE f1=1;	
COMMIT;	
	SELECT f2 FROM tbl1 WHERE f1=1;

4. Проблема чтения фантомов (level 3) – одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет или удаляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Транзакция 1	Транзакция 2
	SELECT SUM(f2) FROM tbl1;
INSERT INTO tbl1 (f1,f2) VALUES (15,20);	
COMMIT;	
	SELECT SUM(f2) FROM tbl1;

Уровень изолированности транзакций — условное значение, определяющее, в какой мере в результате выполнения логически параллельных транзакций в СУБД допускается получение несогласованных данных. Шкала уровней изолированности транзакций содержит ряд значений, проранжированных от наинизшего до наивысшего; более высокий уровень изолированности соответствует лучшей согласованности данных, но его использование может снижать количество физически параллельно выполняемых транзакций. И наоборот, более низкий уровень изолированности позволяет выполнять больше параллельных транзакций, но снижает точность данных.

Под «уровнем изоляции транзакций» понимается степень обеспечиваемой внутренними механизмами СУБД (то есть не требующей специального программирования) защиты от всех или некоторых видов вышеперечисленных несогласованности данных, возникающих при параллельном выполнении транзакций. Стандарт SQL-92 определяет шкалу из четырёх уровней изоляции: Read uncommitted, Read committed, Repeatable read, Serializable. Первый из них является самым слабым, последний — самым сильным, каждый последующий включает в себя все предыдущие.

Уровень изоляции	Грязное чтение	Неповторное чтение	Фантомное чтение
Read uncommited	-	-	-
Read committed	+	-	-
Repeatable read	+	+	-
Serializable (пессимистич.)/ snapshot(оптимистич.)	+	+	+

Read uncommitted (чтение незафиксированных данных) – низший (первый) уровень изоляции. Он гарантирует только отсутствие потерянных обновлений. Типичный способ реализации данного уровня изоляции — блокировка данных на время выполнения команды изменения, что гарантирует, что команды изменения одних и тех же строк, запущенные параллельно, фактически выполнятся последовательно, и ни одно из изменений не потеряется. Транзакции, выполняющие только чтение, при данном уровне изоляции никогда не блокируются.

Read committed (чтение фиксированных данных) – большинство промышленных СУБД, в частности, Microsoft SQL Server, PostgreSQL и Oracle, по умолчанию используют именно этот уровень. На этом уровне обеспечивается защита от «грязного» чтения, тем не менее. Реализация завершённого чтения может основываться на одном из двух подходов: блокировании или версионности.

Блокирование читаемых и изменяемых данных заключается в том, что пишущая транзакция блокирует изменяемые данные для читающих транзакций, работающих на уровне read committed или более высоком, до своего завершения, препятствуя, таким образом, «грязному» чтению, а данные, блокируемые читающей транзакцией, освобождаются сразу после завершения операции SELECT (таким образом, ситуация «неповторяющегося чтения» может возникать на данном уровне изоляции).

Сохранение нескольких версий параллельно изменяемых строк – при каждом изменении строки СУБД создаёт новую версию этой строки, с которой продолжает работать изменившая данные транзакция, в то время как любой другой «читающей» транзакции возвращается последняя зафиксированная версия. Преимущество такого подхода в том, что он обеспечивает большую скорость, так как предотвращает блокировки.

Repeatable read (повторяемость чтения) – уровень, при котором читающая транзакция «не видит» изменения данных, которые были ею ранее прочитаны. При этом никакая другая транзакция не может изменять данные, читаемые текущей транзакцией, пока та не окончена.

Serializable (упорядочиваемость) – самый высокий уровень изолированности; транзакции полностью изолируются друг от друга, каждая выполняется так, как будто параллельных транзакций не существует. Только на этом уровне параллельные транзакции не подвержены эффекту «фантомного чтения».

20. Блокировки.

Блокировка в СУБД — отметка о захвате объекта транзакцией в ограниченный или исключительный доступ с целью предотвращения коллизий и поддержания целостности данных.

Разновидности блокировок по типу блокируемых ресурсов
Блокируемые ресурсы:

- Конкретная строка в таблице
- Ключ (один или несколько ключей в индексе)
- Страница
- Экстент
- Таблица и все относящиеся к ней данные (индексы, ключи, данные)
- База (блокируется, когда меняется схема базы)

Эскалация блокировок – это процесс, который направлен на оптимизацию работы сервера, позволяющий заменять множество низкоуровневых блокировок одной или несколькими блокировками более высокого уровня. Например, если у нас создано множество блокировок уровня строки, и все строки принадлежат одной странице, то сервер, в целях экономии ресурсов, может заменить их все одной блокировкой уровня страницы.

Основной проблемой, касающейся укрупнения блокировок, является то обстоятельство, что решение, когда осуществлять укрупнение, принимает сервер баз данных, и это решение может не быть оптимальным для приложений, имеющих различные требования. Механизм укрупнения блокировок можно модифицировать с помощью инструкции ALTER TABLE.

Гранулярность блокировки определяет, какой ресурс блокируется в одной попытке блокировки. Компонент Database Engine может блокировать следующие ресурсы: строки, страницы, индексный ключ или диапазон индексных ключей, таблицы, экстент, саму базу данных. Система выбирает требуемую гранулярность блокировки автоматически.

Строка является наименьшим ресурсом, который можно заблокировать. Блокировка уровня строки также включает как строки данных, так и элементы индексов. Блокировка на уровне строки означает, что блокируется только строка, к которой обращается приложение. Поэтому все другие строки данной таблицы остаются свободными и их могут использовать другие приложения. Компонент Database Engine также может заблокировать страницу, на которой находится подлежащая блокировке строка.

В кластеризованных таблицах страницы данных хранятся на уровне узлов (кластеризованной) индексной структуры, и поэтому для них вместо блокировки строк применяется блокировка с индексными ключами.

Блокироваться также могут единицы дискового пространства, которые называются экстентами и имеют размер 64 Кбайт. Экстенты блокируются автоматически, и когда растет таблица или индекс, то для них требуется выделять дополнительное дисковое пространство.

Гранулярность блокировки оказывает влияние на одновременный конкурентный доступ. В общем, чем выше уровень гранулярности, тем больше

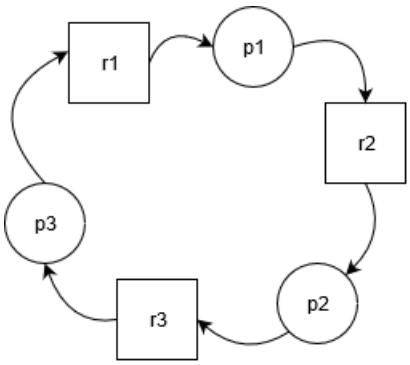
сокращается возможность совместного доступа к данным. Это означает, что блокировка уровня строк максимизирует одновременный конкурентный доступ, т.к. она блокирует всего лишь одну строку страницы, оставляя все другие строки доступными для других процессов. С другой стороны, низкий уровень блокировки увеличивает системные накладные расходы, поскольку для каждой отдельной строки требуется отдельная блокировка. Блокировка на уровне страниц и таблиц ограничивает уровень доступности данных, но также уменьшает системные накладные расходы.

Иерархия

Блокировка	**Описание**
Блокировка с намерением совмещаемого доступа (IS)	Защищает запрошенные или полученные совмещаемые блокировки на некоторых (но не на всех) ресурсах на более низком уровне иерархии.
Блокировка с намерением монопольного доступа (IX)	Режим IX является расширенным режимом IS, кроме того, он защищает запрос на совмещаемые блокировки на ресурсах более низкого уровня.
Совмещаемая блокировка с намерением монопольного доступа (SIX)	Защищает запрошенные или полученные совмещаемые блокировки на всех ресурсах более низкого уровня иерархии, а также блокировки с намерением на некоторых (но не всех) ресурсах более низкого уровня. На ресурсах верхнего уровня допускаются одновременные блокировки IS.
Блокировка с намерением обновления (IU)	Защищает запрошенные или полученные блокировки обновления на всех ресурсах более низкого уровня в иерархии. Блокировки IU применяются только на страничных ресурсах.
Совмещаемая блокировка с намерением обновления (SIU)	Сочетание блокировок S и IU в результате раздельного запрашивания этих блокировок и одновременного удержания их обеих.
Блокировка обновления с намерением монопольного доступа (UIX)	Сочетание блокировок U и IX в результате раздельного запрашивания этих блокировок, и одновременного удержания их обеих.

21. Взаимоблокировки, их обнаружение и устранение.

Взаимоблокировки (тупики) – это ситуация, возникающая в результате монопольного использования разделяемых ресурсов, когда один процесс, владея ресурсом, запрашивает другой ресурс, занятый непосредственно или через цепочку запросов другим процессом, ожидающим освобождения ресурса, занятого первым процессом.



Методы борьбы с тупиками:

1. Недопущение тупиков
2. Обход тупиков
3. Обнаружение тупиков

Обнаружение тупиков выполняется с помощью графовой модели Холта.

Монитор взаимоблокировок компонента SQL Server Database Engine периодически проверяет задачи на состояние взаимоблокировки. Если монитор обнаруживает цикличную зависимость, то выбирается одна задача, для которой транзакция будет завершена с ошибкой. Это позволяет другой задаче завершить свою транзакцию. Позднее приложение может повторно выполнить транзакцию, которая завершилась с ошибкой.

В качестве альтернативы пользователь может указать приоритет сеансов в ситуации взаимоблокировки, используя инструкцию SET DEADLOCK_PRIORITY. Если у двух сеансов имеются различные приоритеты в случае взаимоблокировки, то в качестве жертвы взаимоблокировки будет выбран сеанс с более низким приоритетом. Если у обоих сеансов установлен одинаковый приоритет в случае взаимоблокировки, то в качестве объекта взаимоблокировки будет выбран сеанс, откат которого потребует наименьших затрат. Если сеансы, вовлеченные в цикл взаимоблокировки, имеют один и тот же приоритет в случае взаимоблокировки и одинаковую стоимость, то жертва взаимоблокировки выбирается случайным образом.

Обнаружение взаимоблокировки выполняется потоком монитора блокировок, который периодически производит поиск по всем задачам в экземпляре компонента Database Engine.

- Значение интервала по умолчанию составляет 5 секунд.
- Если поток монитора блокировки находит взаимоблокировки, интервал обнаружения взаимоблокировок снижается с 5 секунд до 100 миллисекунд в зависимости от частоты взаимоблокировок.
- Если поток монитора блокировки прекращает поиск взаимоблокировок, Компонент Database Engine увеличивает интервал до 5 секунд.
- Если взаимоблокировка была только что найдена, предполагается, что следующие потоки, которые должны ожидать блокировки, входят в цикл взаимоблокировки. Первая пара элементов, ожидающих

блокировки, после того как взаимоблокировка была обнаружена, запускает поиск взаимоблокировок вместо того, чтобы ожидать следующий интервал обнаружения взаимоблокировки.

22. Режимы блокировок и совместимость блокировок.

Режимы

Блокировка	Описание
Блокировка с намерением совмещаемого доступа (IS)	Защищает запрошенные или полученные совмещаемые блокировки на некоторых (но не на всех) ресурсах на более низком уровне иерархии.
Блокировка с намерением монопольного доступа (IX)	Режим IX является расширенным режимом IS, кроме того, он защищает запрос на совмещаемые блокировки на ресурсах более низкого уровня.
Совмещаемая блокировка с намерением монопольного доступа (SIX)	Защищает запрошенные или полученные совмещаемые блокировки на всех ресурсах более низкого уровня иерархии, а также блокировки с намерением на некоторых (но не всех) ресурсах более низкого уровня. На ресурсах верхнего уровня допускаются одновременные блокировки IS.
Блокировка с намерением обновления (IU)	Защищает запрошенные или полученные блокировки обновления на всех ресурсах более низкого уровня в иерархии. Блокировки IU применяются только на страничных ресурсах.
Совмещаемая блокировка с намерением обновления (SIU)	Сочетание блокировок S и IU в результате раздельного запрашивания этих блокировок и одновременного удержания их обеих.
Блокировка обновления с намерением монопольного доступа (UIX)	Сочетание блокировок U и IX в результате раздельного запрашивания этих блокировок и одновременного удержания их обеих.
Блокировка обновления (U)	Применяется к тем ресурсам, которые могут быть обновлены. Предотвращает возникновение распространенной формы взаимоблокировки, возникающей тогда, когда несколько сеансов считывают, блокируют и затем, возможно, обновляют ресурс.
Монопольная блокировка (X)	Используется для операций модификации данных, таких как инструкции INSERT, UPDATE или DELETE. Гарантирует, что несколько обновлений не будет выполнено одновременно для одного ресурса.
Совмещаемая блокировка (S)	Используется для операций считывания, которые не меняют и не обновляют данные, такие как инструкция SELECT.

Совместимость блокировок определяет, могут ли несколько транзакций одновременно получить блокировку одного и того же ресурса. Если ресурс уже блокирован другой транзакцией, новая блокировка может быть предоставлена только в том случае, если режим запрошенной блокировки совместим с режимом существующей.

Запрашиваемый режим	IS	S	U	IX	SIX	X
Намеренная разделяемая (IS)	V	V	V	V	V	X
Совмещаемая блокировка (S)	V	V	V	X	X	X
Блокировка обновления (U)	V	V	X	X	X	X
С намерением монопольного доступа (IX)	V	X	X	V	X	X
Совмещаемая с намерением монопольного доступа (SIX)	V	X	X	X	X	X
Монопольная (X)	-	X	X	X	X	X

23. Обработка ошибок в транзакциях.

Реализация обработчика ошибок на языке Transact-SQL похожа на обработку исключений в языках Microsoft Visual C# и Microsoft Visual C++. Группа инструкций на языке Transact-SQL может быть заключена в блок TRY. Если ошибка возникает в блоке TRY, управление передается следующей группе инструкций, заключенных в блок CATCH.

BEGIN TRY
{ sql_statement | statement_block }
END TRY
BEGIN CATCH
[{ sql_statement | statement_block }]
END CATCH
[;]

sql_statement – любая из инструкций языка Transact-SQL.
statement_block – любая группа инструкций языка Transact-SQL в пакете или заключенная в блок BEGIN...END.

Конструкция TRY...CATCH перехватывает все ошибки исполнения с кодом серьезности, большим чем 10, которые не закрывают подключение к базе данных.

За блоком TRY сразу же должен следовать блок CATCH. Размещение каких-либо инструкций между инструкциями END TRY и BEGIN CATCH вызовет синтаксическую ошибку.

Конструкция TRY...CATCH не может охватывать несколько пакетов. Конструкция TRY...CATCH не может охватывать множество блоков инструкций на языке Transact-SQL. Например: конструктор TRY...CATCH не может охватывать два блока BEGIN...END из инструкций на языке Transact-SQL и не может охватывать конструкцию IF...ELSE.

Если ошибки в блоке TRY не возникают, то после выполнения последней инструкции в блоке TRY управление передается инструкции, расположенной сразу после инструкции END CATCH. Если же в коде, заключенном в блоке TRY, происходит ошибка, управление передается первой инструкции в соответствующем блоке CATCH. Если инструкция END CATCH является последней инструкцией хранимой процедуры или триггера, управление передается обратно инструкции, вызвавшей эту хранимую процедуру или триггер.

В области блока CATCH для получения сведений об ошибке, приведшей к выполнению данного блока CATCH, можно использовать следующие системные функции:

- функция ERROR_NUMBER() возвращает номер ошибки;
- функция ERROR_SEVERITY() возвращает степень серьезности ошибки;
- функция ERROR_STATE() возвращает код состояния ошибки;
- функция ERROR_PROCEDURE() возвращает имя хранимой процедуры или триггера, в котором произошла ошибка;
- функция ERROR_LINE() возвращает номер строки, которая вызвала ошибку, внутри подпрограммы;
- функция ERROR_MESSAGE() возвращает полный текст сообщения об ошибке. Текст содержит значения подставляемых параметров, таких как длина, имена объектов или время.

Эти функции возвращают значение NULL, если их вызов происходит вне области блока CATCH.

Конструкции TRY...CATCH не обрабатывают следующие условия.

- Предупреждения и информационные сообщения с уровнем серьезности 10 или ниже.
- Ошибки с уровнем серьезности 20 или выше, которые приводят к завершению обработки задачи компонентом Компонент SQL Server Database Engine для сеанса. Если возникла ошибка с уровнем серьезности 20 или выше, а подключение к базе данных не разорвано, конструкция TRY...CATCH обработает эту ошибку.
- Такие запросы, как прерывания от клиента или разрыв соединения, вызванный с клиента.
- Завершение сеанса системным администратором с помощью инструкции KILL.

24. Логическая и физическая архитектуры журнала транзакций.

Логическая

Каждая база данных SQL Server имеет журнал транзакций, в котором фиксируются все изменения данных, произведенные в каждой из транзакций. Журнал транзакций является критическим компонентом базы данных и в случае системного сбоя может потребоваться для приведения базы данных в согласованное состояние.

Логически журнал транзакций SQL Server работает так, как если бы он являлся последовательностью записей в журнале. Каждая запись журнала идентифицируется регистрационным номером транзакции (номер LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи. Записи журнала сохраняются в серийной последовательности по мере их создания, таким

образом если LSN2 больше, чем LSN1, то изменение, описанное записью журнала, на которую ссылается LSN2, произошло после изменения, описанного записью журнала LSN1. Каждая запись журнала содержит идентификатор транзакции, к которой она относится. Все записи журнала, связанные с определенной транзакцией, с помощью обратных указателей связаны в цепочку, которая предназначена для ускорения отката транзакции.

Записи журнала для изменения данных содержат либо выполненную логическую операцию, либо исходный и результирующий образ измененных данных. Исходный образ записи — это копия данных до выполнения операции, а результирующий образ — копия данных после ее выполнения.

В журнал транзакций записываются различные типы операций, например:

- начало и конец каждой транзакции;
- любые изменения данных (вставка, обновление или удаление), включая изменения в любой таблице (в том числе и в системных таблицах), производимые системными хранимыми процедурами или инструкциями языка DDL;
- любое выделение и освобождение страниц и экстентов;
- создание и удаление таблиц и индексов.

Кроме того, регистрируются операции отката. Каждая транзакция резервирует в журнале транзакций место, чтобы при выполнении инструкции отката или возникновения ошибки в журнале было достаточно места для регистрации отката.

Раздел файла журнала из первой записи, который должен присутствовать для успешного отката всей базы данных к последней зарегистрированной записи называется активной частью журнала, активным журналом или заключительным фрагментом журнала. Этот раздел журнала необходим для полного восстановления базы данных. Ни одна часть активного журнала не может быть усечена. Регистрационный номер транзакции в журнале (LSN) этой первой записи называется минимальным номером LSN восстановления (MinLSN).

Физическая

Журнал транзакций в базе данных сопоставляет один или несколько физических файлов. Физически последовательность записей журнала эффективно хранится в наборе физических файлов, которые образуют журнал транзакций. Для каждой базы данных должен существовать хотя бы один файл журнала.

Внутри системы компонент Компонент SQL Server Database Engine делит каждый физический файл журнала на несколько виртуальных файлов журнала. Виртуальные файлы журнала не имеют фиксированных размеров. Не существует также и определенного числа виртуальных файлов журнала, приходящихся на один физический файл журнала. Компонент Компонент Database Engine динамически определяет размер виртуальных файлов журнала при создании или расширении файлов журнала.

Журнал транзакций является оборачиваемым файлом. Когда конец логического журнала достигнет конца физического файла журнала, новые записи журнала будут размещаться в начале физического файла журнала. Если в журнале содержится несколько физических файлов журнала, логический журнал будет продвигаться по всем физическим файлам журнала до тех пор, пока он не вернется на начало первого физического файла журнала.

25. Контрольные точки, активная часть журнала и усечение журнала транзакций.

Контрольные точки

Контрольная точка создает известную надежную точку, с которой Компонент SQL Server Database Engine может начать применение изменений, содержащихся в журнале, во время восстановления после непредвиденного отключения или аварии.

Компонент Database Engine поддерживает несколько типов контрольных точек: автоматические, косвенные, ручные и внутренние.

Усечение журнала

Усечение журнала необходимо для предотвращения переполнения журнала. При усечении журнала удаляются неактивные виртуальные файлы журнала из логического журнала транзакций базы данных SQL Server , что приводит к освобождению пространства в логическом журнале для повторного использования физическим журналом транзакций. Если усечение журнала транзакций не выполняется, со временем он заполняет все доступное место на диске, отведенное для файлов физического журнала. Однако перед усечением журнала должна быть выполнена операция создания контрольной точки.

За исключением тех случаев, когда усечение журнала по каким-то причинам задерживается, оно выполняется автоматически после наступления следующих событий.

- В простой модели восстановления — после достижения контрольной точки.
- В модели полного восстановления или в модели восстановления с неполным протоколированием — после создания резервной копии журналов, при условии, что со времени предыдущей операции резервного копирования была достигнута контрольная точка.

Усечение журнала может быть задержано из-за множества факторов. В случае большой задержки усечения журнала возможно заполнение журнала транзакций.

Контрольные точки и активная часть журнала

При достижении контрольных точек измененные страницы данных записываются из буферного кэша текущей базы данных на диск. Это сводит к минимуму активную часть журнала, которую приходится обрабатывать при полном восстановлении базы данных.

Контрольная точка выполняет в базе данных следующее.

- Записывает в файл журнала запись, отмечающую начало контрольной точки.
- Сохраняет данные, записанные для контрольной точки в цепи записей журнала контрольной точки.
- Одним из элементов данных, регистрируемых в записях контрольной точки, является номер LSN первой записи журнала, при отсутствии которой успешный откат в масштабе всей базы данных невозможен. Такой номер LSN называется минимальным номером LSN восстановления (MinLSN). Номер MinLSN является наименьшим значением из:
  - номера LSN начала контрольной точки;
  - номера LSN начала старейшей активной транзакции;
  - номера LSN начала старейшей транзакции репликации, которая еще не была доставлена базе данных распространителя.

Контрольные точки срабатывают в следующих ситуациях.

- При явном выполнении инструкции CHECKPOINT. Контрольная точка срабатывает в текущей базе данных соединения.
- При выполнении в базе данных операции с минимальной регистрацией, например при выполнении операции массового копирования для базы данных, на которую распространяется модель восстановления с неполным протоколированием.
- При добавлении или удалении файлов баз данных с использованием инструкции ALTER DATABASE.
- При остановке экземпляра SQL Server инструкцией SHUTDOWN или при остановке службы SQL Server (MSSQLSERVER). И в том, и в другом случае будет создана контрольная точка для каждой базы данных на экземпляре SQL Server.
- Экземпляр SQL Server периодически создает для каждой базы данных автоматические контрольные точки с целью сократить время восстановления базы данных.
- При создании резервной копии базы данных.
- При выполнении действия, требующего отключения базы данных. Примерами могут служить присвоение параметру AUTO_CLOSE значения ON и закрытие последнего соединения пользователя с базой данных или изменение параметра базы данных, требующее перезапуска базы данных.

Часть файла журнала, начинающаяся с номера MinLSN и заканчивающаяся последней зафиксированной записью, называется активной частью журнала или "активным журналом". Этот раздел журнала необходим для выполнения полного восстановления базы данных. Ни одна часть

активного журнала не может быть усечена. Все записи журнала до номера MinLSN должны быть удалены из частей журнала.

26. Управление участниками системы безопасности.

Участники — сущности, которые могут запрашивать ресурсы SQL Server.

Поддерживаются два вида учетных записей подключения к серверу:

- учетные записи сервера, создаваемые на основании учетных записей операционной системы, и
- учетные записи сервера, создаваемые для прямого подключения к серверу.

Защищаемые объекты – это ресурсы, доступ к которым регулируется системой авторизации. Некоторые защищаемые объекты могут храниться внутри других, создавая иерархии «областей», которые сами могут защищаться.

27. Управление разрешениями.

PostgreSQL использует концепцию ролей (roles) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того как роль настроена. Роли могут владеть объектами базы данных (например, таблицами) и выдавать другим ролям разрешения на доступ к этим объектам, управляя тем, кто имеет доступ и к каким объектам. Кроме того, можно предоставить одной роли членство в другой роли, таким образом одна роль может использовать привилегии других ролей.

Роли базы данных концептуально полностью отличаются от пользователей операционной системы. Роли базы данных являются глобальными для всей установки кластера базы данных (не для отдельной базы данных). Для создания роли используется команда SQL CREATE ROLE:

CREATE ROLE name;

Для удаления роли используется команда DROP ROLE :

DROP ROLE name;

Для получения списка существующих ролей, рассмотрите pg_roles системного каталога. Метакоманда \du программы psql также полезна для получения списка существующих ролей.

Для начальной настройки кластера базы данных система сразу после инициализации всегда содержит одну предопределённую роль. Эта роль является суперпользователем ("superuser") и по умолчанию (если не изменено при запуске initdb) имеет такое же имя, как и пользователь операционной системы, инициализирующий кластер баз данных. Обычно эта роль называется postgres. Для создания других ролей вначале нужно подключиться с этой ролью.

Каждое подключение к серверу базы данных выполняется под именем конкретной роли, и эта роль определяет начальные привилегии доступа для команд, выполняемых в этом соединении.

Роль базы данных может иметь атрибуты, определяющие её полномочия и взаимодействие с системой аутентификации клиентов:

- Право подключения
- Статус суперпользователя
- Создание базы данных
- Создание роли
- Запуск репликации
- Пароль

Атрибуты ролей могут быть изменены после создания командой ALTER ROLE.

Часто бывает удобным сгруппировать пользователей для упрощения администрирования привилегий: привилегии выдаются или отзываются на всю группу. Для настройки групповой роли сначала нужно создать саму роль. После того как групповая роль создана, в неё можно добавлять или удалять членов, используя команды GRANT и REVOKE.