РАБОТА В СУБД POSTGRESQL

Методы контроля качества данных

Индексы и оптимизация

Устранение дубликатов

- ✔ **Приведение информации к унифицированному виду** (типичный пример – написание названия одной страны разными способами, написание номеров телефонов, ввод в поля разным регистром)
- ✔ **Определение информации**, внесенной не в то поле
- ✔ **Разбор адресов** – разбиение полей, в которых закодирована информация о нескольких атрибутах

Строковые функции

UPPER – преобразует все символы строки в верхний регистр.

LOWER – преобразует все символы строки в нижний регистр.

INSTR – возвращает n-е вхождение подстроки в строке.

LENGTH – возвращает длину строки.

LTRIM – удаляет все указанные символы с левой стороны строки.

RTRIM – удаляет все указанные символы с правой стороны строки.

TRIM – удаляет все указанные

Строковые функции

SUBSTR – извлекает подстроку из строки

REPLACE – заменяет последовательность символов в строке другим набором символов

TRANSLATE – заменяет последовательность символов в строке другим набором символов (посимвольно)

Регулярные выражения

regexp_match

```
SELECT regexp_match('foobarbequebaz', 'bar.*que');
```

output | Messages | Explain | Notifications

regexp_match text[]
{barbeque}

regexp_matches

```
SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
```

output | Messages | Explain | Notifications

regexp_matches text[]
{bar,beque}
{bazil,barf}

regexp_replace

```
SELECT regexp_replace('foobarbaz', 'b..', 'X')
```

output | Messages | Explain | Notifications

regexp_replace text
fooXbaz

ы е

в ы р а ж е н и

я

regexp_split_to_table

```
SELECT foo FROM regexp_split_to_table('the quick brown fox jumps over the lazy dog', '\s+') AS foo;
```

a output | Messages | Explain × | Notifications

foo text
the
quick
brown
fox
jumps
over
the
lazy
dog

ы е

в ы р а ж е н и

я

split_part

```
select split_part('Текст1,Текст2',',',1)
```

output Messages Notifications

split_part text
Текст1

substring

```
select substring('Текст2','\d')
```

output Messages Notifications

substring text
2

Регулярные выражения

^ — начало строки;

$ — конец строки;

. — любой символ;

* - любое количество предыдущих символов;

+ - 1 или более предыдущих символов;

? - 0 или 1 предыдущих символов;

() - группировка конструкций;

| - оператор «ИЛИ»;

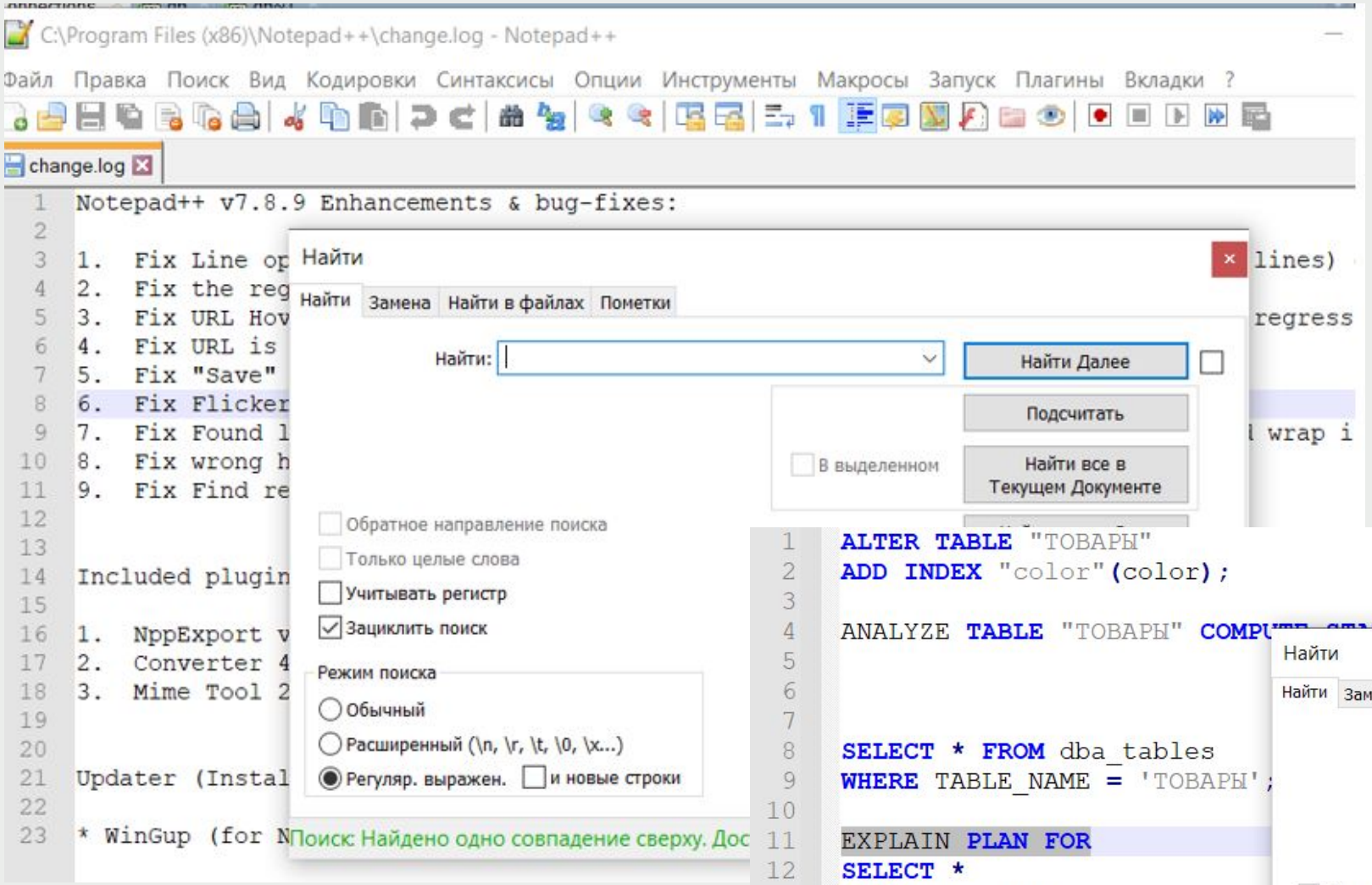
[] - любой из перечисленных символов, диапазон. Если первый символ в

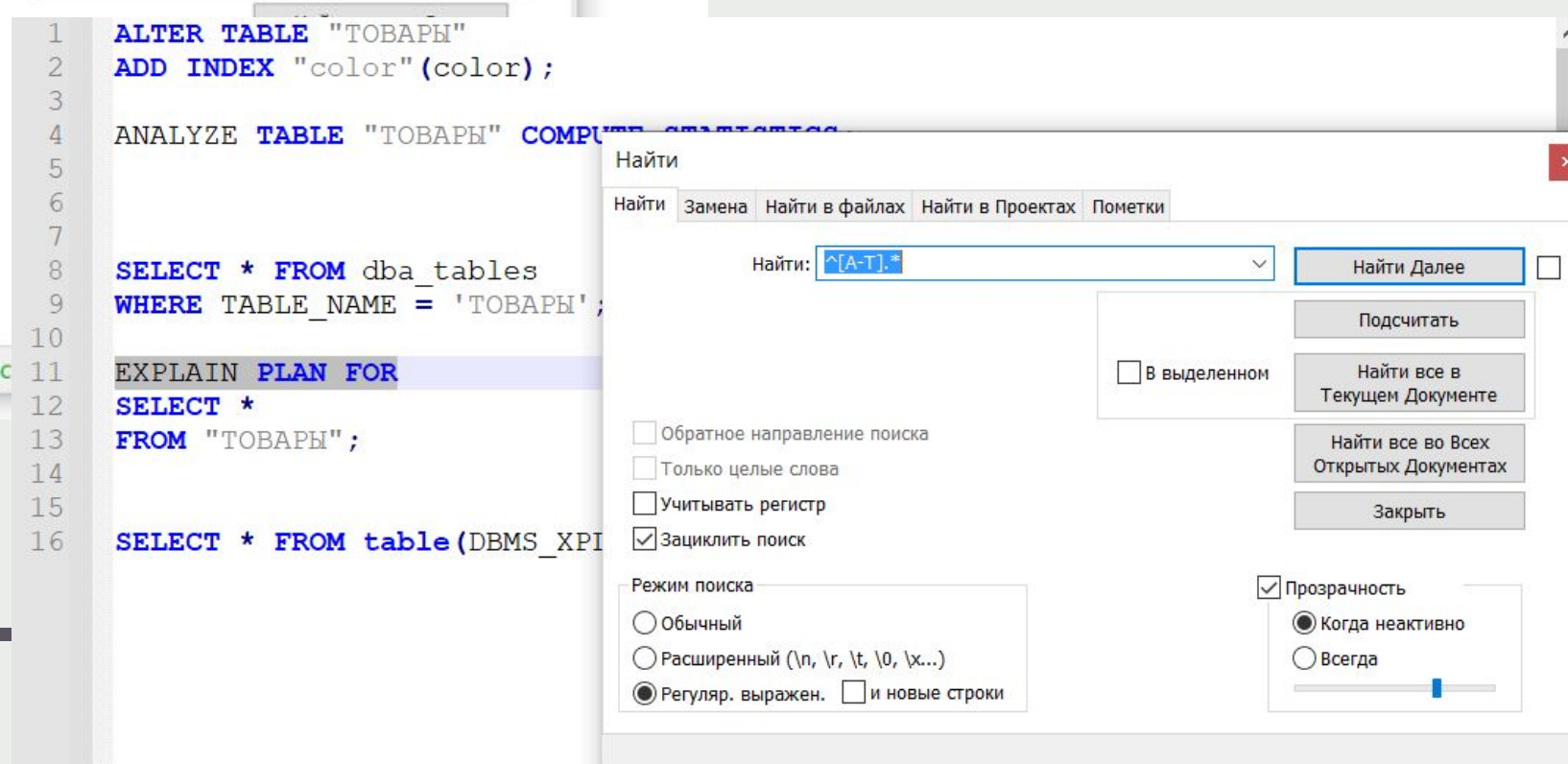
этой конструкции - «^», то массив работает наоборот - проверяемый

символ не должен совпадать с тем, что перечислено в скобках;

{} - повторение символа несколько раз;

\ - обратный слеш. Экранирование служебных символов.


C:\Program Files (x86)\Notepad++\change.log - Notepad++
Файл Правка Поиск Вид Кодировки Синтаксисы Опции Инструменты Макросы Запуск Плагины Вкладки ?
change.log
Notepad++ v7.8.9 Enhancements & bug-fixes:
Included plugin
Updater (Instal
* WinGup (for N
Найти
Найти Замена Найти в файлах Пометки
Найти:
Найти Далее
Подсчитать
В выделенном
Найти все в Текущем Документе
Обратное направление поиска
Только целые слова
Учитывать регистр
Зациклить поиск
Режим поиска
Обычный
Расширенный (\n, \r, \t, \0, \x...)
Регуляр. выражен.
и новые строки
Поиск: Найдено одно совпадение сверху. Дос


ALTER TABLE "ТОВАРЫ"
ADD INDEX "color"(color);
ANALYZE TABLE "ТОВАРЫ" COMPU
SELECT * FROM dba_tables
WHERE TABLE_NAME = 'ТОВАРЫ';
EXPLAIN PLAN FOR
SELECT *
FROM "ТОВАРЫ";
SELECT * FROM table(DBMS_XPI
Найти
Найти Замена Найти в файлах Найти в Проектах Пометки
Найти: ^[A-T].*
Найти Далее
Подсчитать
В выделенном
Найти все в Текущем Документе
Найти все во Всех Открытых Документах
Закрыть
Обратное направление поиска
Только целые слова
Учитывать регистр
Зациклить поиск
Режим поиска
Обычный
Расширенный (\n, \r, \t, \0, \x...)
Регуляр. выражен.
и новые строки
Прозрачность
Когда неактивно
Всегда

Регулярные выражения

Специальные метасимволы, ими можно заменить некоторые готовые конструкции:

\b — обозначает не символ, а границу между символами

\d — цифровой символ

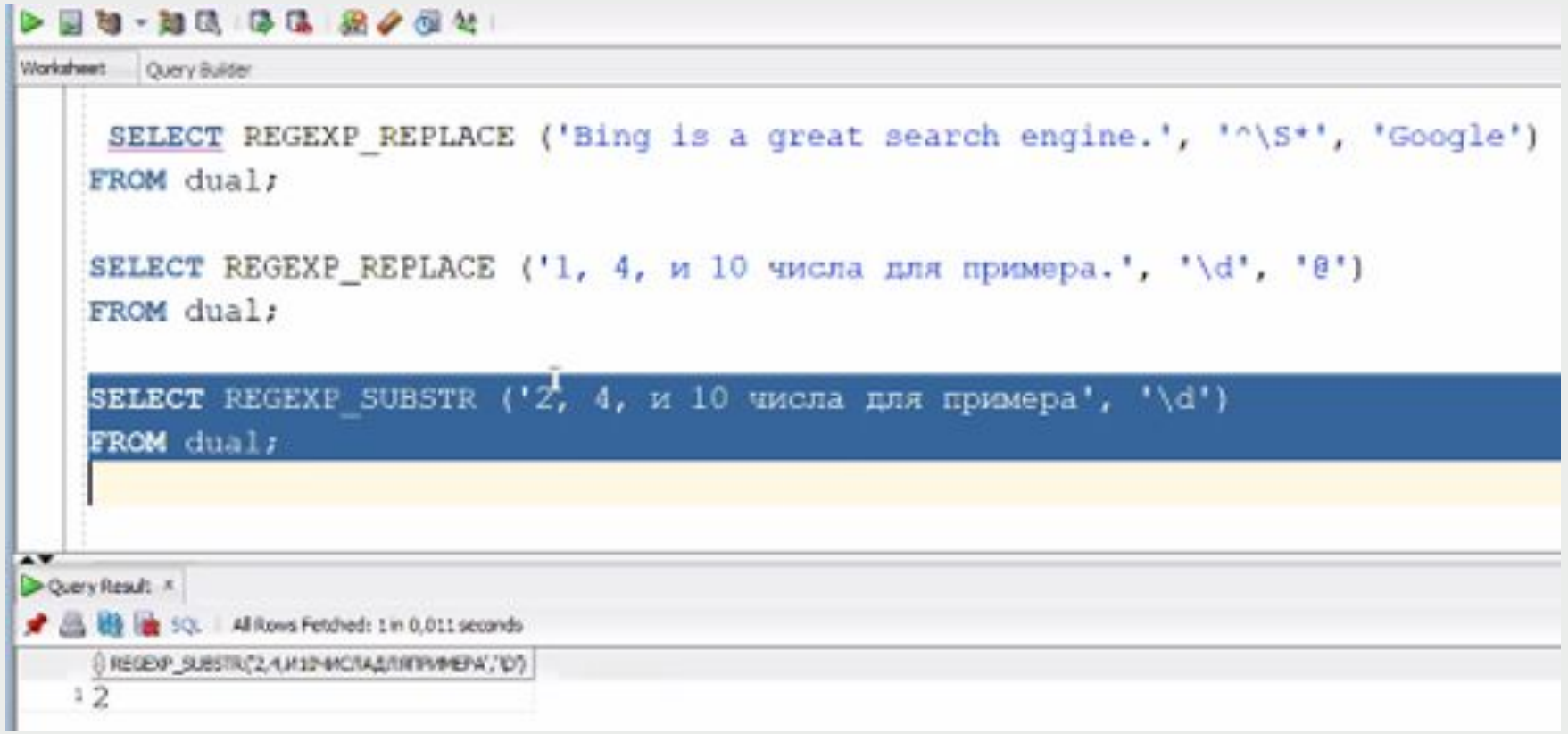
\D — нецифровой символ

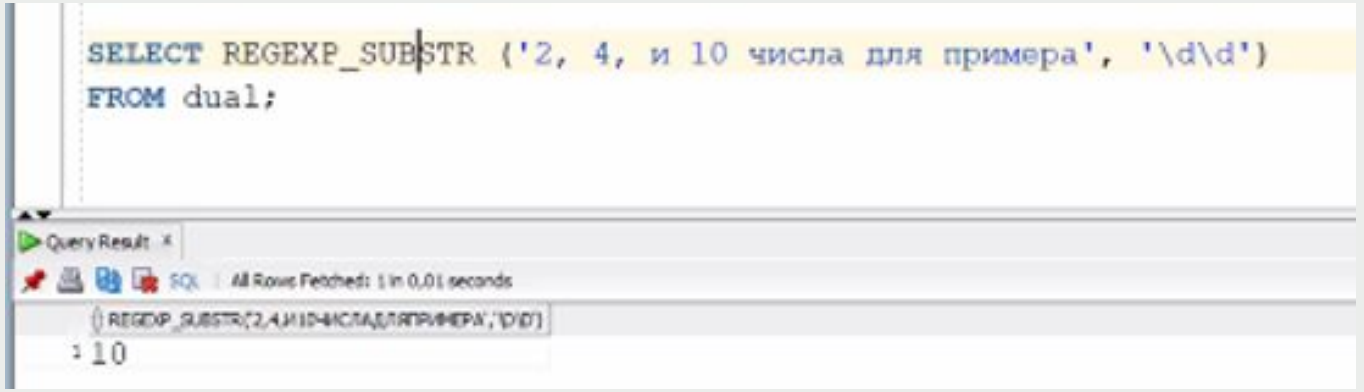
\s — пробельный символ

\S — непробельный символ

\w — буквенный или цифровой символ или знак подчеркивания

\W — любой символ, кроме буквенного или цифрового символа или знака подчеркивания


Worksheet
Query Builder
SELECT REGEXP_REPLACE ('Bing is a great search engine.', '^\S*', 'Google')
FROM dual;
SELECT REGEXP_REPLACE ('1, 4, и 10 числа для примера.', '\d', '@')
FROM dual;
SELECT REGEXP_SUBSTR ('2, 4, и 10 числа для примера', '\d')
FROM dual;
Query Result
All Rows Fetched: 1 in 0,011 seconds
2

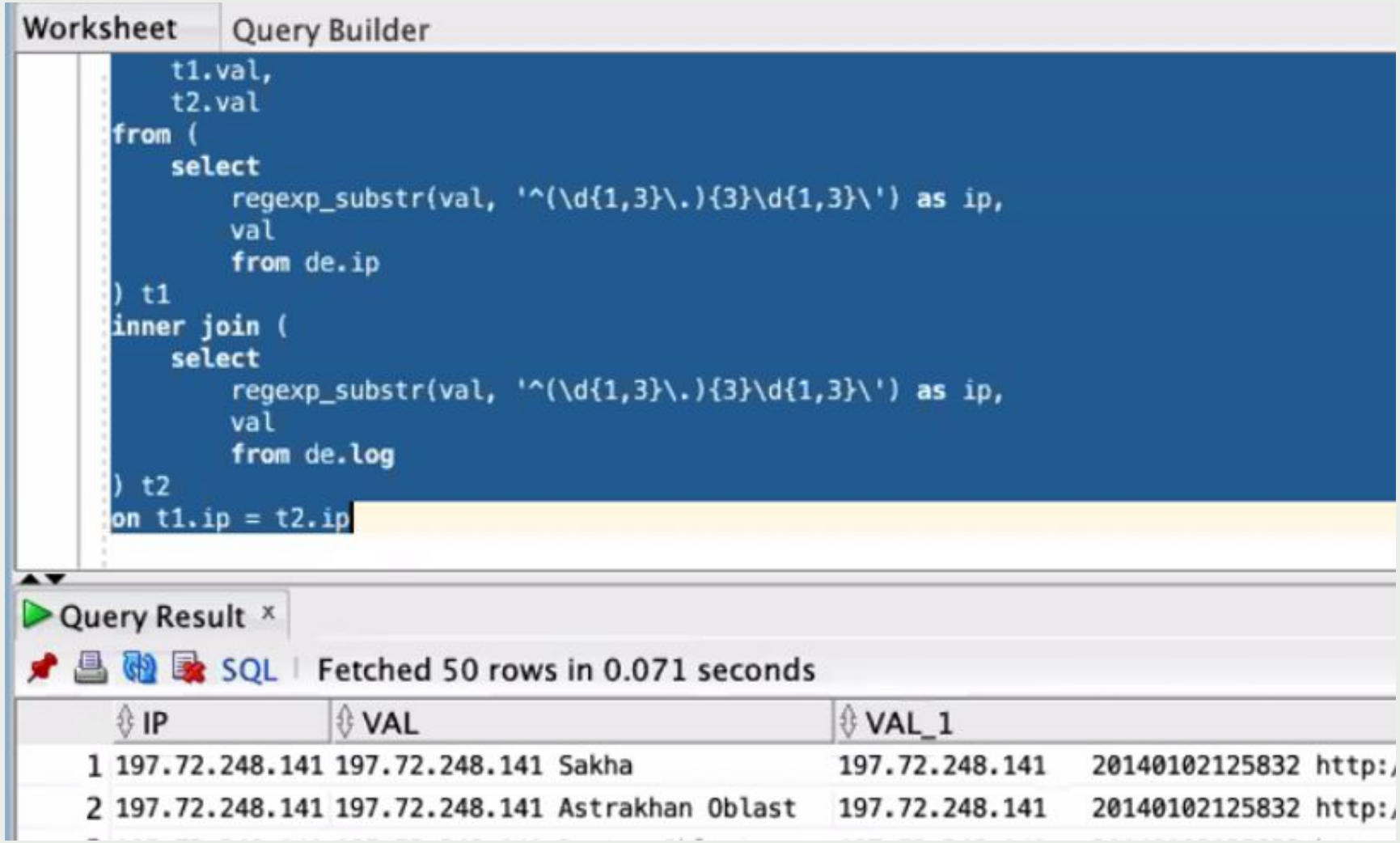

SELECT REGEXP_SUBSTR ('2, 4, и 10 числа для примера', '\d\d')
FROM dual;
Query Result
All Rows Fetched: 1 in 0,01 seconds
10

```
where
    regexp_like(product_name, '^[A-Ca-c].*')  ;
```

Query Result

SQL | All Rows Fetched: 50 in 0,31 seconds

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESCRIPTION
1	1787	CPU D300	Dual CPU @ 300Mhz. For light personal processing on
2	2439	CPU D400	Dual CPU @ 400Mhz. Good price/performance ratio; fo
3	1788	CPU D600	Dual CPU @ 600Mhz. State of the art, high clock spe
4	1742	CD-ROM 500/16x	CD drive, read only, speed 16x, maximum capacity 50
5	2402	CD-ROM 600/E/24x	600 MB external 24x speed CD-ROM drive (read only).


Worksheet
Query Builder
t1.val,
t2.val
from (
select
regexp_substr(val, '^(\d{1,3}\.){3}\d{1,3}\') as ip,
val
from de.ip
) t1
inner join (
select
regexp_substr(val, '^(\d{1,3}\.){3}\d{1,3}\') as ip,
val
from de.log
) t2
on t1.ip = t2.ip
Query Result
SQL
Fetched 50 rows in 0.071 seconds
IP
VAL
VAL_1
1 197.72.248.141 197.72.248.141 Sakha 197.72.248.141 20140102125832 http:/
2 197.72.248.141 197.72.248.141 Astrakhan Oblast 197.72.248.141 20140102125832 http:/

Пример

Время имеет формат часы: минуты. И часы, и минуты состоят из двух цифр, пример: 09:00.

Напишите регулярное выражение для поиска времени в строке: “Завтрак в 09:00”.

Учтите, что “37:98” – некорректное время

Worksheet Query Builder

```
select
    REGEXP_SUBSTR(time,'([0-1]\d|2[0-3]):[0-5]\d')
from (
    select 'Завтрак в (09:00)' as time from dual
    union all
    select 'Завтрак в (09:61)' as time from dual
    union all
    select 'Завтрак в (20:33)' as time from dual
    union all
    select 'Завтрак в (09:55)' as time from dual
    union all
    select 'Завтрак в (30:10)' as time from dual
)
```

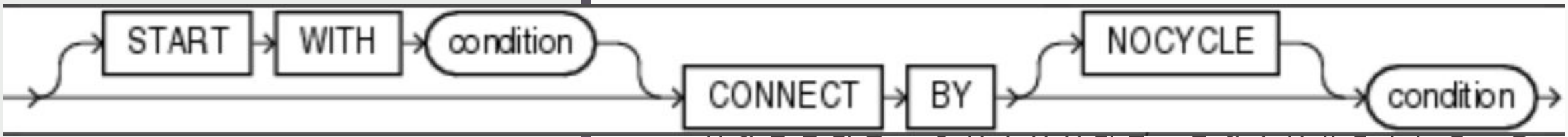
Query Result

SQL | All Rows Fetched: 5 in 0,264 seconds

	REGEXP_SUBSTR(TIME,'([0-1]\D|2[0-3]):[0-5]\D')
1	09:00
2	(null)
3	20:33
4	09:55
5	(null)

Пример выбора фамилии и города

```
select
        regexp_substr(t.dt,'[^,]+',1,1) f,
regexp_substr(t.dt,'[^,]+',1,3) city from
        regtest t
```



Иерархические запросы

Данные, которые находятся в таблице, могут быть иерархически упорядочены, например, иметь порядок Начальник-Подчинённый. Запросы, выводящие данные в иерархическом виде – называются иерархическими

```
create table test_table (
  id int,
  pid int,
  title varchar(256)
);

SELECT level, id, pid, title
FROM test_table
START WITH pid is null
CONNECT BY PRIOR id = pid;
```

Все дочерние строки оказываются под своими родителями

Script Output | Query Result | SQL | All Rows Fetched:

	LEVEL	ID	PID	TITLE
1	1	1	(null)	Россия
2	2	2	1	Воронеж
3	3	3	2	ООО ЛЛЛ
4	2	4	1	Москва
5	2	5	1	Лиски

```
SELECT lpad(' ', 3*level)||title as Tree
FROM test_table
START WITH pid is null
CONNECT BY PRIOR id = pid
ORDER SIBLINGS BY title;
```

Script Output | Query Result | SQL | All Rows Fetched: 5 in 0,019 secc

	TREE
1	Россия
2	Воронеж
3	ООО ЛЛЛ
4	Лиски
5	Москва

Отсортированные данные

Select [level] [список столбцов]

From т.1 join т.2 {условие соединения}

Start with [начало]

Connect by {условие подчинённости}

[Иерархические (рекурсивные) запросы / Хабр (habr.com)]

```
SELECT level, id, pid, title
FROM test_table
START WITH pid is null
CONNECT BY PRIOR id = pid;
```

Script Output | Query Result

SQL | All Rows Fetched: 5

	LEVEL	ID	PID	TITLE
1	1	1	(null)	Россия
2	2	2	1	Воронеж
3	3	3	2	ООО ЛЛЛ
4	2	4	1	Москва
5	2	5	1	Лиски

Порядок строк это хорошо, но нам было бы трудно понять, две строки рядом это родитель и его потомок или два брата-потомка одного родителя

Oracle предлагает в помощь дополнительный псевдостолбец **LEVEL**. Как легко догадаться, в нем записывается уровень записи по отношению к корневой

PRIOR. Это обычный унарный оператор, точно такой же как + или -. “Позвоните родителям” – говорит он, заставляя Оракл обратиться к предыдущей записи

```
SELECT SYS_CONNECT_BY_PATH(title, '/') as Path
FROM test_table
START WITH pid is null
CONNECT BY PRIOR id = pid;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0,019 seconds

	PATH
1	/Россия
2	/Россия/Воронеж
3	/Россия/Воронеж/ООО ЛЛЛ
4	/Россия/Москва
5	/Россия/Лиски

```
SELECT SYS_CONNECT_BY_PATH(title, '/') as Path
FROM test_table
WHERE id=3
START WITH pid is null
CONNECT BY PRIOR id = pid;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0,02 seconds

	PATH
1	/Россия/Воронеж/ООО ЛЛЛ

Файловые менеджеры обычно пишут путь к каталогу, в котором вы находитесь: /home/maovrn/documents/ и т.п.

Используем функцию **SYS_CONNECT_BY_PATH()**. Она принимает два параметра через запятую: название колонки и строку с символом-разделителем

```
SELECT id, pid, title, level,
 CONNECT_BY_ISLEAF as IsLeaf,
 PRIOR title as Parent,
 CONNECT_BY_ROOT title as Root
FROM test_table
START WITH pid is null
CONNECT BY PRIOR id = pid
ORDER SIBLINGS BY title;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0,02 seconds

	ID	PID	TITLE	LEVEL	ISLEAF	PARENT	ROOT
1	1	(null)	Россия	1	0	(null)	Россия
2	2	1	Воронеж	2	0	Россия	Россия
3	3	2	ООО ЛЛЛ	3	1	Воронеж	Россия
4	5	1	Лиски	2	1	Россия	Россия
5	4	1	Москва	2	1	Россия	Россия

Оператор PRIOR ссылался к родительской записи

Помимо него есть другой унарный оператор **CONNECT_BY_ROOT**, который ссылается на корневую запись, т.е. на самую первую в выборке

```
SELECT sq.rn
FROM (SELECT rownum as rn FROM dual
 CONNECT BY level <= (SELECT max(id) FROM test_table)) sq
WHERE sq.rn not in (SELECT id FROM test_table)
ORDER BY rn;
```

Воспользовавшись методом **regexp_substr** в сочетании с командой CONNECT by можно преобразовать каждую подстроку с разделителями в строку таблицы

```
select     regexp_substr('Петров,01011988,Москва','[^,]+',1,level) ll
from     dual
connect by level <=  REGEXP_COUNT('Петров,01011988,Москва',',')+1;
```

Query Result 26 | Query Result 27 | Explain Plan | Query Result 28

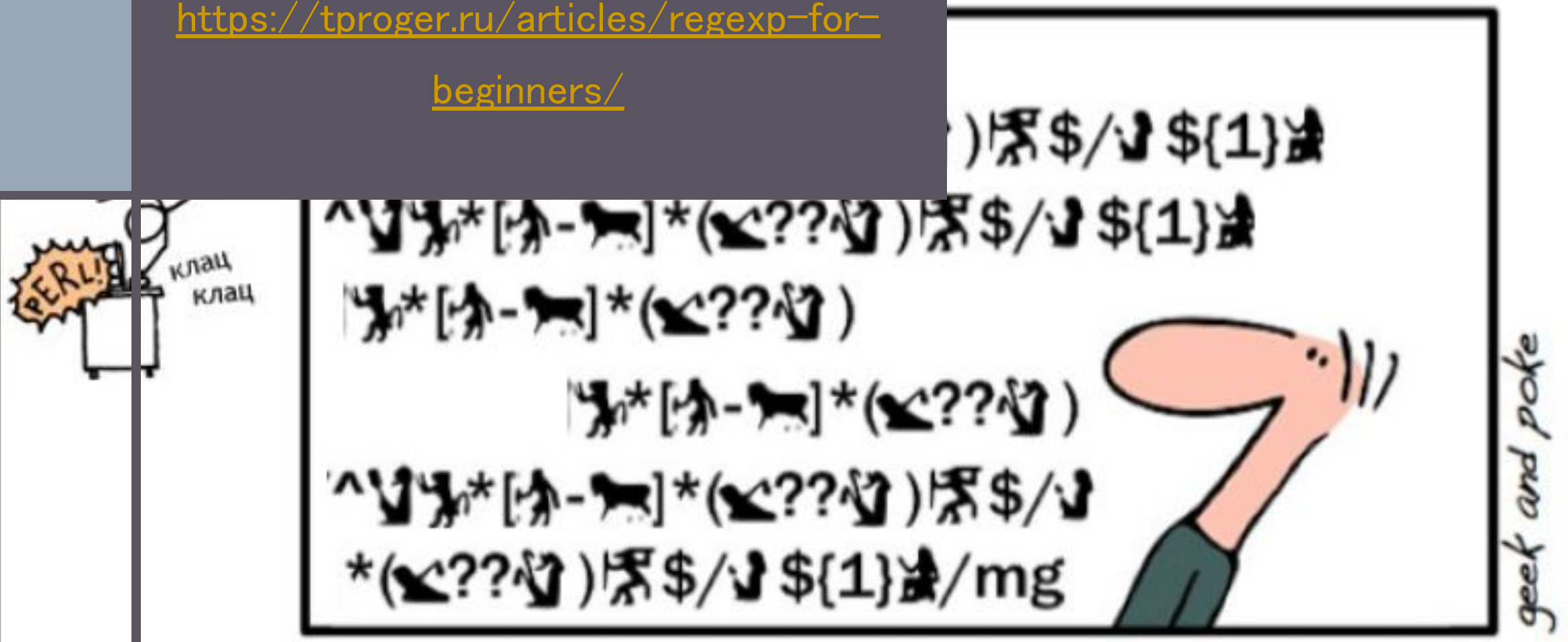
SQL | All Rows Fetched: 3 in 0,017 seconds

	LL
1	Петров
2	01011988
3	Москва

Регулярные выражения для новичков

ССЫЛКА НА САЙТ

https://tproger.ru/articles/regexp-for-beginners/



Задание

Перейти на сайт и составить краткий конспект по основам синтаксиса регулярных выражений

Использование регулярных выражений REGEXP в ORACLE SQL / Oracle SQL / Sql.ru

Задание для самостоятельного выполнения

1. Напишите регулярное выражение для поиска HTML-цвета, заданного как #ABCDEF, то есть # и содержит затем 6 шестнадцатеричных символов
2. Написать регулярное выражение для выбора IP адресов

Этапы выполнения запроса

Запрос, поступающий серверу на выполнение, проходит несколько этапов:

1. **Разбор**
2. **Трансформация**
3. **Планирование**
4. **Выполнение**

Разбор

Лексический анализатор разбирает текст запроса на *лексемы* (такие как ключевые слова, строковые и числовые литералы и т. п.)

Синтаксический анализатор убеждается, что полученный набор лексем соответствует грамматике языка

Query | Query History

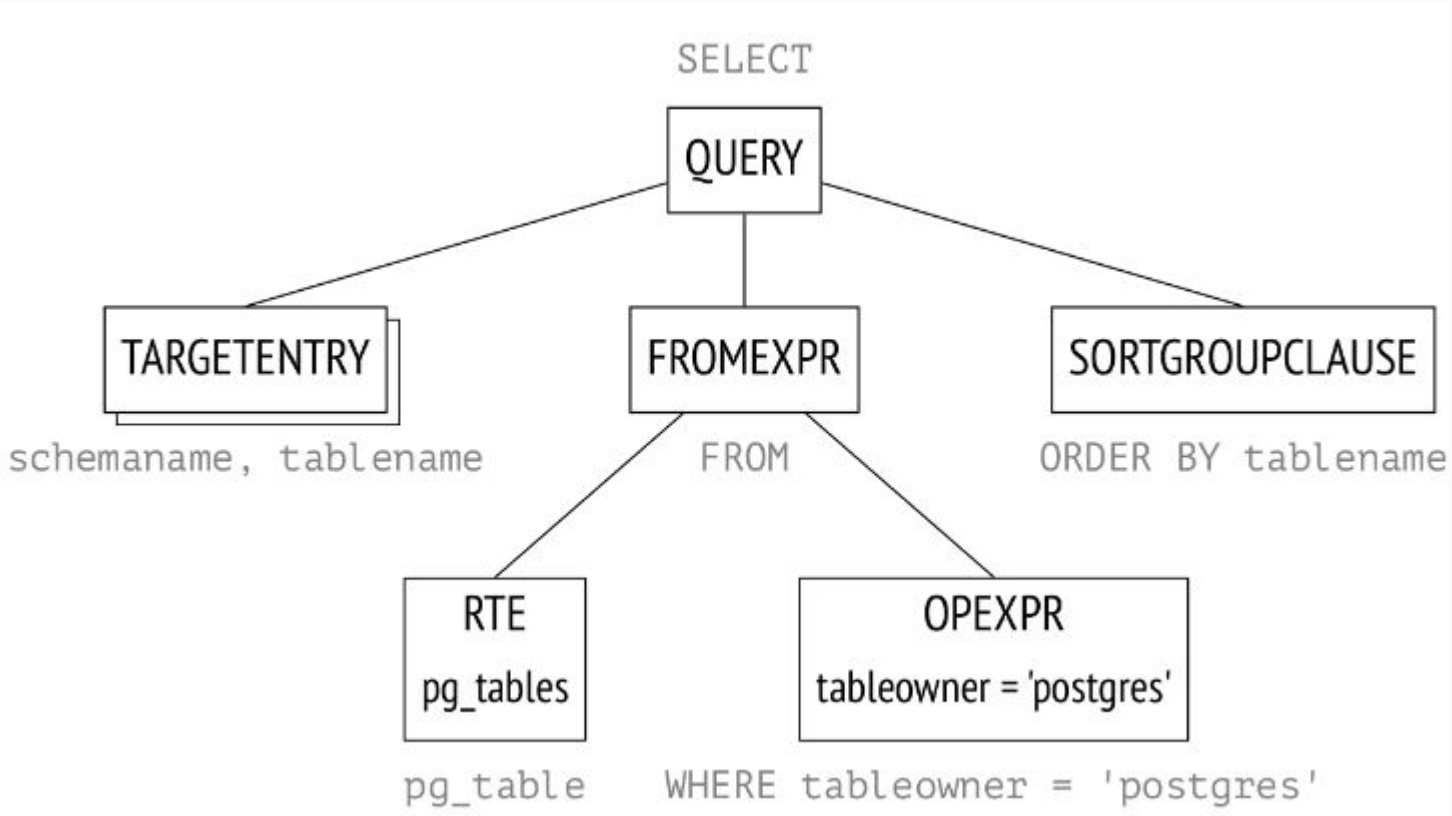
```
1  SELECT schemaname, tablename
2  FROM pg_tables
3  WHERE tableowner = 'postgres'
4  ORDER BY tablename;
```

Data output | Messages | Notifications

	schemaname name	tablename name
1	public	accounts
2	public	gruppa
3	public	passwd
4	pg_catalog	pg_aggreg...
5	pg_catalog	pg_am

Разобранный запрос представляется в виде абстрактного синтаксического дерева

Для него в памяти обслуживающего процесса будет построено дерево, показанное на рисунке в упрощенном виде. Рядом с узлами дерева подписаны части запроса, которые им соответствуют



Семантический разбор

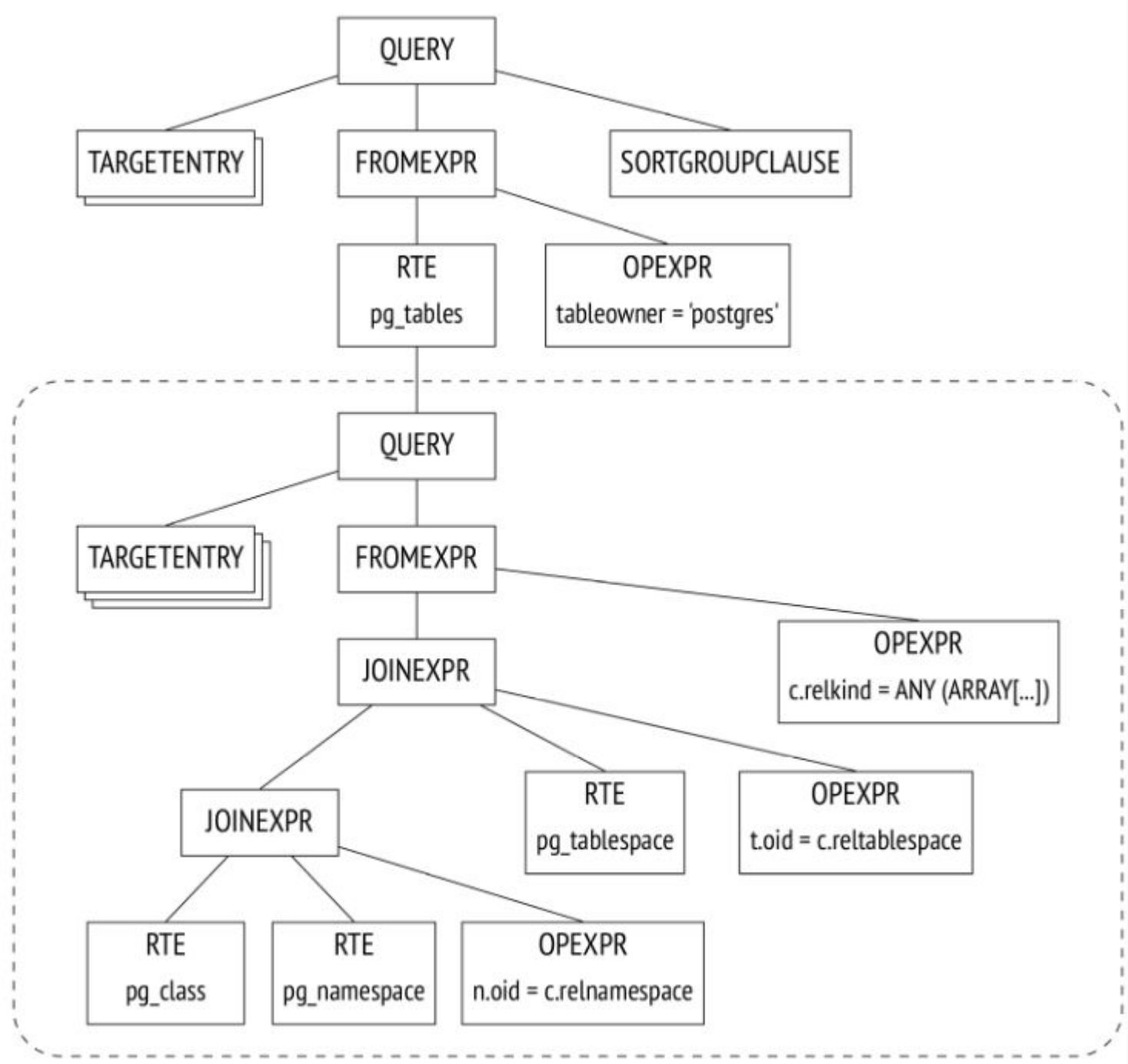
Задача *семантического анализа* — определить, есть ли в базе данных таблицы и другие объекты, на которые запрос ссылается по имени, и есть ли у пользователя право обращаться к этим объектам

Вся необходимая для семантического анализа информация хранится в системном каталоге

Трансформация

Далее запрос может *трансформироваться* (*переписываться*)

Трансформации используются ядром для нескольких целей, одна из них — заменять в дереве разбора имя представления на поддерево, соответствующее запросу этого представления – pg_tables — представление, и после трансформации дерево разбора примет следующий вид



Планирование

Разница во времени выполнения между неоптимальным и оптимальным планами может составлять многие и многие порядки,
поэтому *планировщик*, выполняющий *оптимизацию* разобранного запроса, является одним из самых сложных компонентов системы

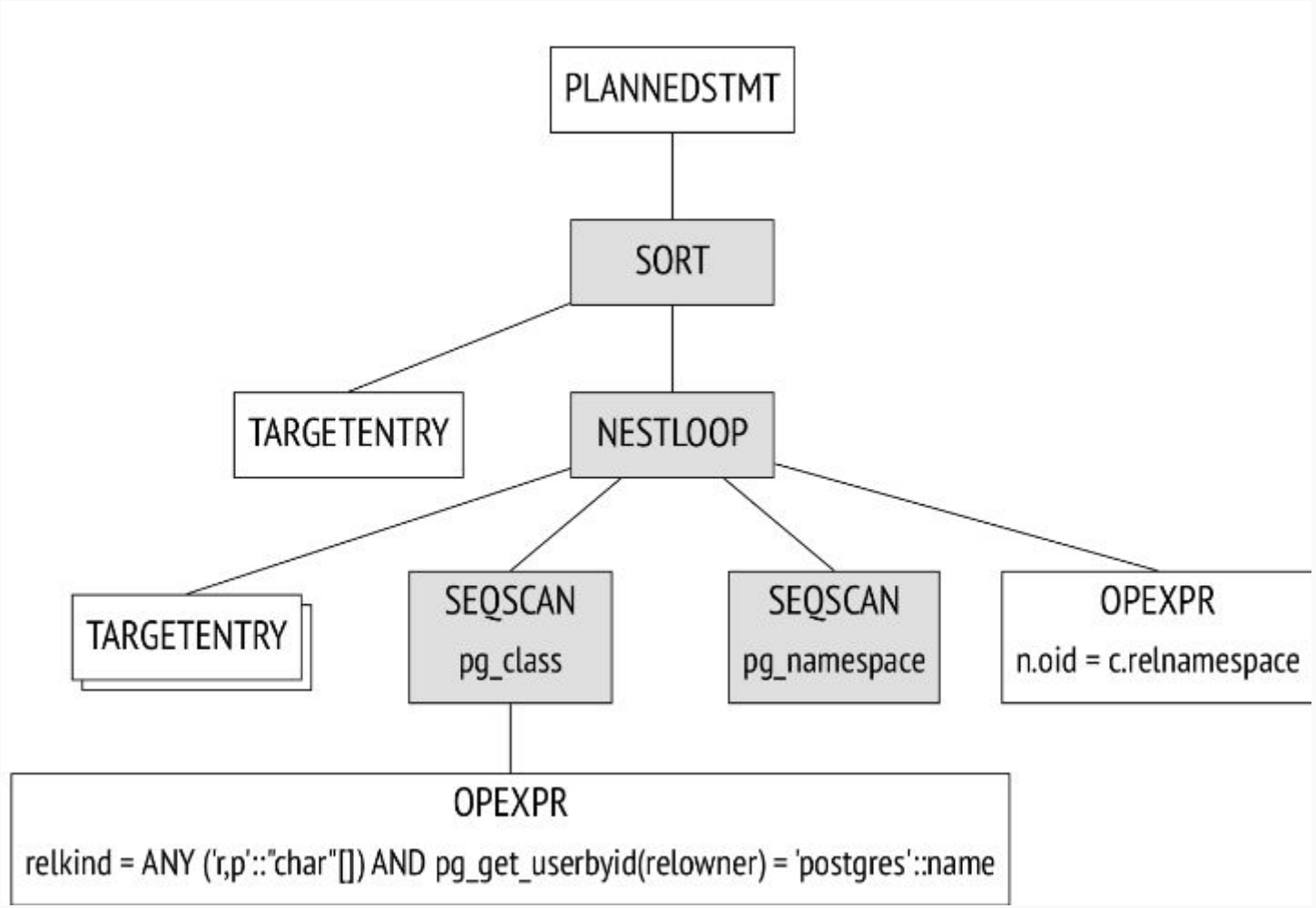
SQL — декларативный язык: запрос определяет, *какие* данные надо получить, но не говорит, *как именно* их получать

Любой запрос можно выполнить разными способами. Для каждой операции, представленной в дереве разбора, могут существовать разные способы ее выполнения:

- данные из таблицы можно получить, прочитав всю таблицу (и отбросив ненужное)
- можно найти подходящие строки с помощью индекса

Дерево плана

План выполнения также представляется в виде дерева, но его узлы содержат не логические, а физические операции над данными



Основные узлы дерева выведены на рисунке (слайд ранее)

В выводе команды EXPLAIN они отмечены стрелочками.

- узел Seq Scan в плане запроса соответствует чтению таблиц
- узел Nested Loop — соединению

Два момента:

- из трех таблиц запроса в дереве осталось только две: планировщик понял, что одна из таблиц не нужна для получения результата и ее можно удалить из дерева плана
- каждый узел дерева снабжен информацией о предполагаемом числе обрабатываемых строк (rows) и о стоимости (cost)

Текстовое представление плана выводит команда EXPLAIN

Query | Query History

```
1  EXPLAIN
2  SELECT schemaname, tablename
3  FROM pg_tables
4  WHERE tableowner = 'postgres'
5  ORDER BY tablename;
```

Data output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=21.78..21.78 rows=1 width=128)
2	Sort Key: c.relname
3	-> Nested Loop Left Join (cost=0.00..21.77 rows=1 width=128)
4	Join Filter: (n.oid = c.relnamespace)
5	-> Seq Scan on pg_class c (cost=0.00..20.63 rows=1 width=72)
6	Filter: ((relkind = ANY ('{r,p}'::"char"[])) AND (pg_get_userbyid(relowner) = 'postgres'::name))
7	-> Seq Scan on pg_namespace n (cost=0.00..1.06 rows=6 width=68)

Перебор планов

PostgreSQL использует ***стоимостной оптимизатор***

Оптимизатор рассматривает всевозможные планы и оценивает предполагаемое количество ресурсов, необходимых для выполнения (таких как операции ввода-вывода и такты процессора)

Такая оценка, приведенная к числовому виду, называется *стоимостью* плана

Из всех просмотренных планов выбирается план с **наименьшей** стоимостью

Количество возможных планов экспоненциально зависит от количества соединяемых таблиц, и просто перебрать один за другим все возможные варианты невозможно даже для относительно простых запросов

Для сокращения вариантов перебора

- Общие табличные выражения обычно оптимизируются отдельно от основного запроса; в версии 12 такое поведение гарантирует предложение **MATERIALIZE**.
- Запросы внутри функций, написанных на любом языке, кроме SQL, оптимизируются отдельно от основного запроса (тело функции на SQL в некоторых случаях может подставляться в запрос)
- Значение параметра *join_collapse_limit* в сочетании с явными предложениями JOIN, а также значение параметра *from_collapse_limit* в сочетании с подзапросами могут зафиксировать порядок некоторых соединений в соответствии с синтаксической структурой запроса

Оптимизатор

Один и тот же оператор SQL можно выполнить несколькими способами, и задача оптимизатора запросов состоит в выборе наиболее быстрого и эффективного пути выполнения каждого запроса к базе данных

План запроса

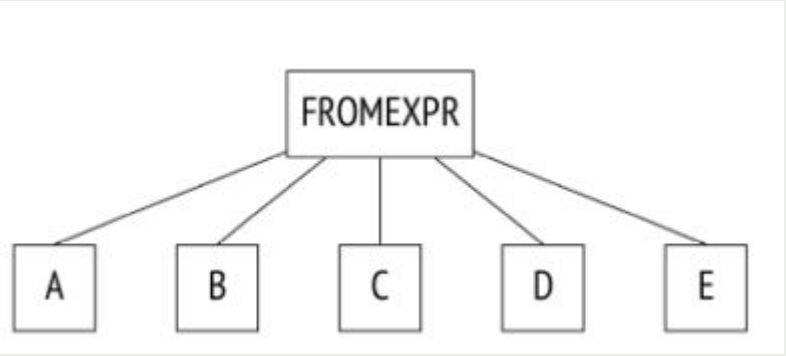
Чтобы разработать наилучший план выполнения любого оператора SQL, оптимизатор

1. **Оценивает возможные пути доступа**, порядки соединения (join orders) и т.п., и выбирает несколько подходящих планов выполнения
2. Вычисляет **стоимость альтернативных планов** на основе использования ими системы ввода-вывода, центрального процессора и памяти. На этом шаге оптимизатор использует статистику, которая включает информацию о распределении данных и характеристики хранения таблиц и индексов
3. **Сравнивает стоимости альтернативных планов** и выбирает план с минимальной стоимостью

Оптимизация запроса

действия:

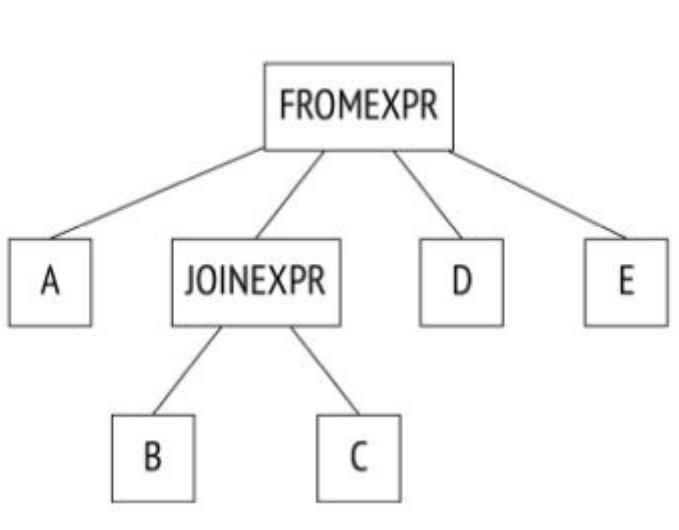
1. Проверить что запрос написан правильно(условия в WHERE, условия соединения)
2. Проверить актуальность статистики
3. Проверить как выполняется доступ к данным
4. Проверить как выполняются соединения
5. Сократить выборку
6. Использовать промежуточную материализацию
7. Применить партиционирование



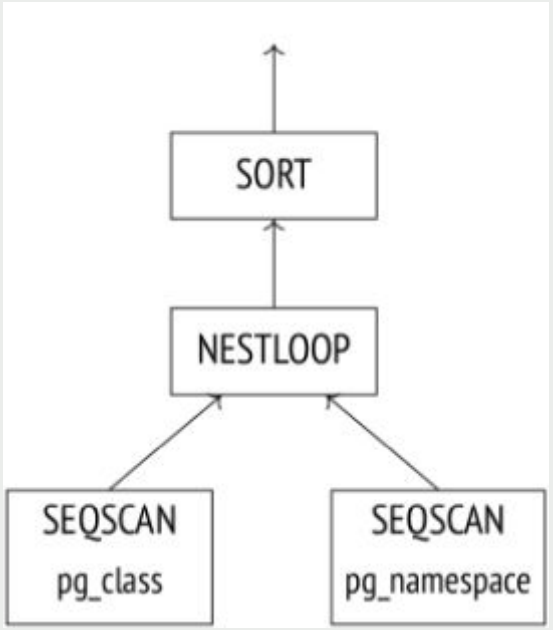
Для такого запроса планировщик будет рассматривать все возможные порядки соединения

SELECT ... **FROM** a, b **JOIN** c **ON** ..., d, e **WHERE** ...

Дерево разбора в данном случае



Выполнение



Некоторые узлы (как NESTLOOP на рисунке) соединяют данные, полученные из разных источников. Такой узел обращается за данными к двум дочерним узлам. Получив две строки, удовлетворяющие условию соединения, узел сразу же передает результирующую строку наверх (в отличие от сортировки, которая сначала вынуждена

Оптимизированный запрос *выполняется* в соответствии с планом

В памяти обслуживающего процесса создается *портал* — объект, хранящий состояние выполняющегося запроса

- Состояние представляется в виде дерева, повторяющего структуру дерева плана
- Фактически узлы дерева работают как конвейер, запрашивая и передавая друг другу строки

Выполнение начинается с корня

- Корневой узел (в примере это операция сортировки SORT) обращается за данными к дочернему узлу
- Получив все строки, узел выполняет сортировку и отдает данные выше, то есть клиенту

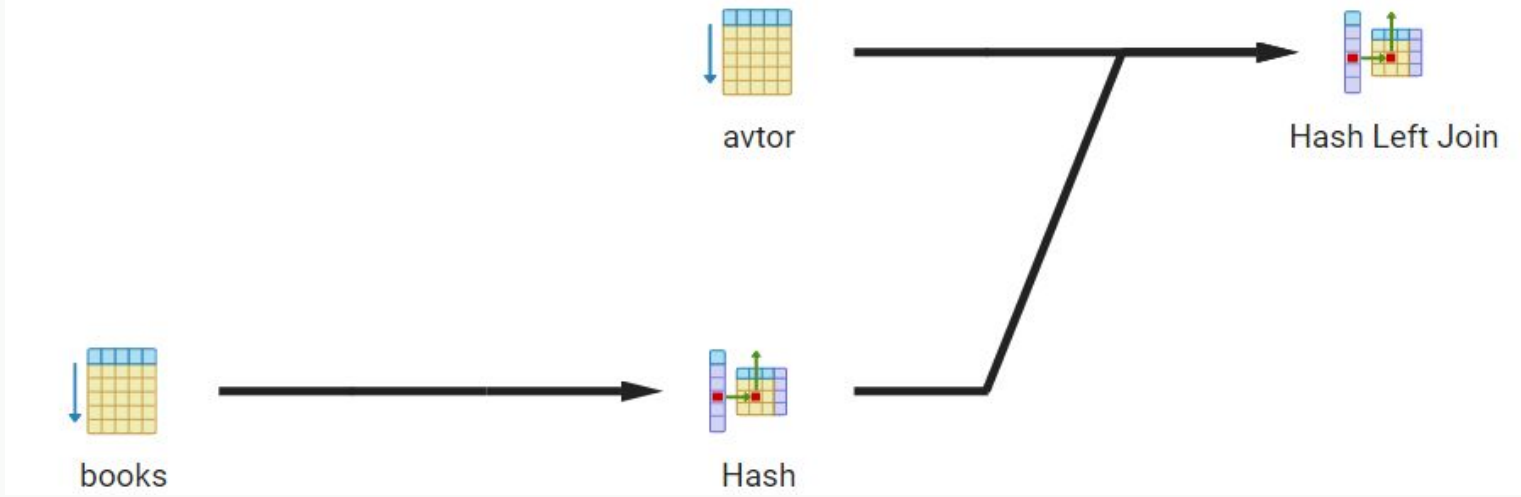
Пример

```
/*+ NestLoop(t1 t2) */
/*+ MergeJoin(t1 t2) */
/*+ Leading(t1 t2) */
```

```
/*+ HashJoin(av bk) */  select
    bk.nazv || ' ' || av.name as "Книга-автор"
from
     avtor av left join books bk on av.kod_avtor = bk.kod_avtor;
```

ata output | Messages | Graph Visualiser × | Explain × | Notifications

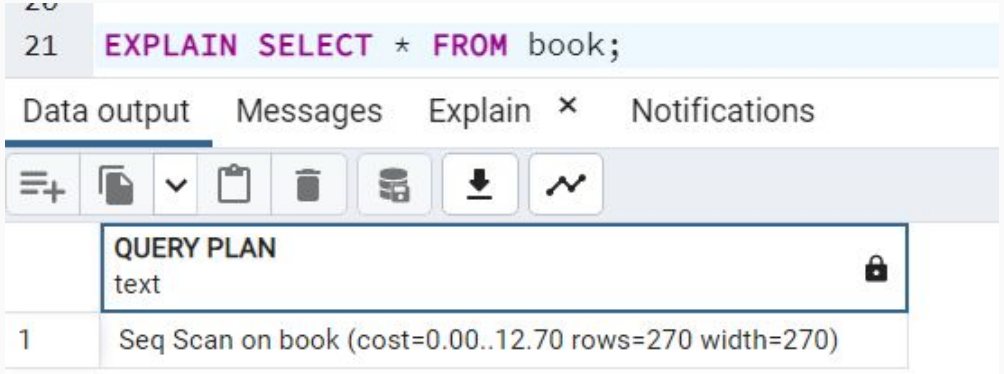
raphical | Analysis | Statistics



```
EXPLAIN [ ( option [, ...] ) ]   statement
EXPLAIN [ ANALYZE ] [ VERBOSE ]  statement

Здесь вариант может быть:
ANALYZE [ boolean ]
VERBOSE [ boolean ]
COSTS [ boolean ]
BUFFERS [ boolean ]
TIMING [ boolean ]
FORMAT { TEXT | XML | JSON | YAML }
```

- ANALYZE выполняет команду и отображает фактическое время выполнения и другую статистику
- VERBOSE отображает дополнительную информацию о плане
- FORMAT определяет выходной формат, который может быть TEXT, XML, JSON или YAML. Нетекстовый вывод содержит ту же информацию, что и формат вывода текста, но его легче проанализировать программой. По умолчанию для этого параметра установлено значение ТЕКСТ



ANALYZE собирает статистику о базе данных

```
ANALYZE [ VERBOSE ] [ table_name [ ( column_name
[, ...] ) ] ]
```

- VERBOSE позволяет отображать сообщения о ходе выполнения.
- table_name Имя указанной таблицы для анализа (может быть дополнено схемой). Если этот параметр не указан, будут проанализированы все обычные таблицы (не внешние) в текущей базе данных.
- column_name Имя назначенного столбца для анализа. По умолчанию - все столбцы

```
EXPLAIN SELECT * FROM bookCopy where price>500;
```

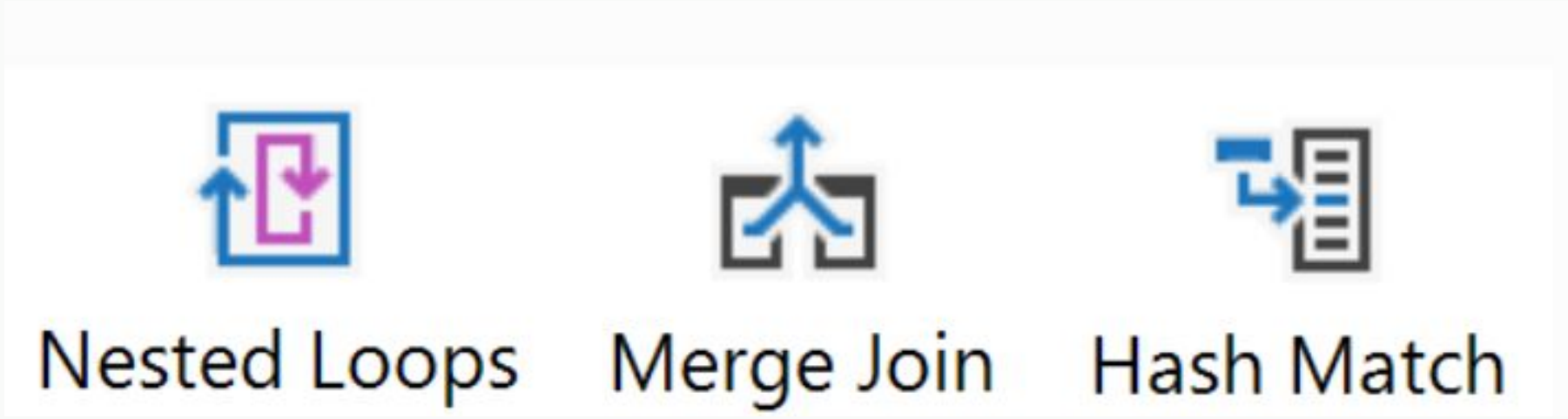
output | Messages | Explain × | Notifications

QUERY PLAN text
Seq Scan on bookcopy (cost=0.00..1.12 rows=10 width=51)
Filter: (price > 500)

```
EXPLAIN ANALYZE SELECT * FROM bookCopy where price>500;
```

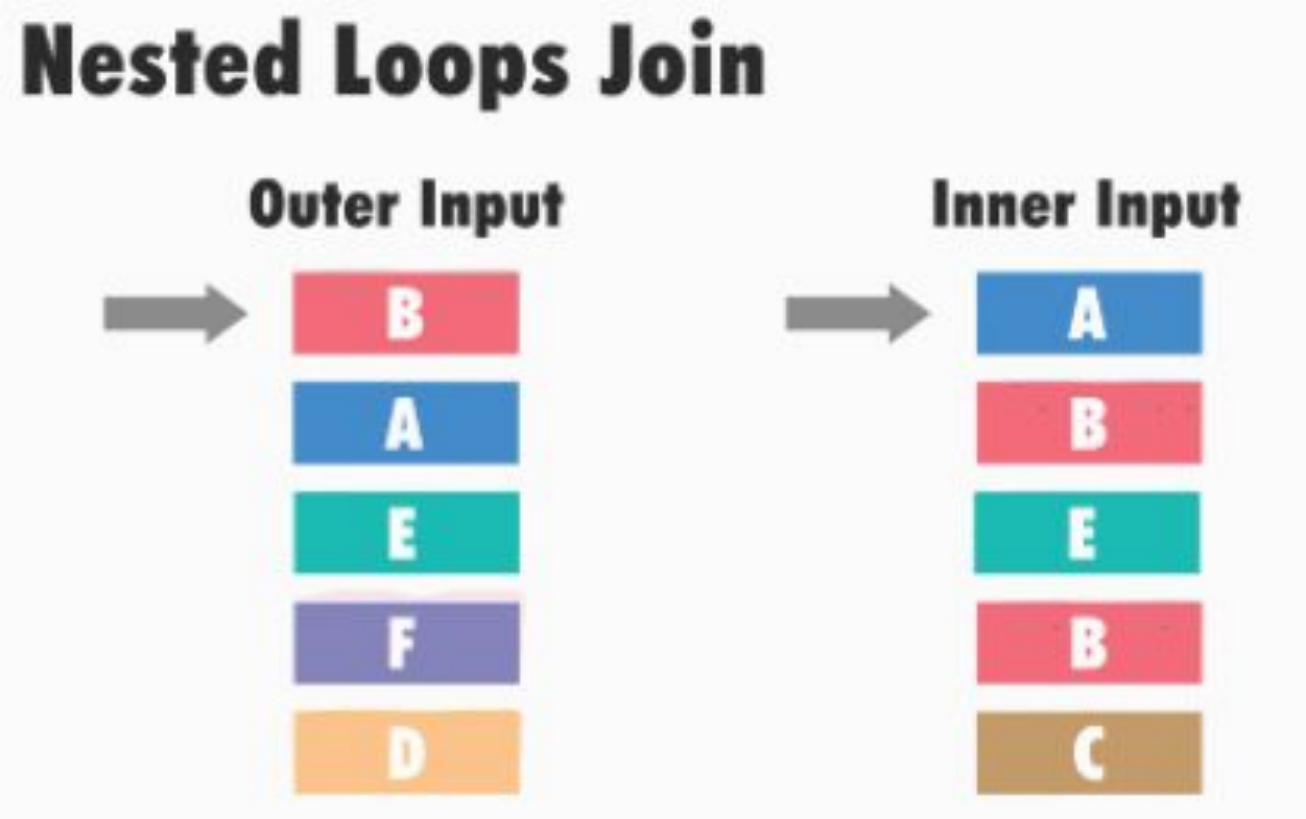
output | Messages | Explain × | Notifications

QUERY PLAN text
Seq Scan on bookcopy (cost=0.00..1.12 rows=10 width=51) (actual time=0.019..0.025 rows=10 loops=1)
Filter: (price > 500)
Planning Time: 0.089 ms
Execution Time: 0.054 ms



Физические операции соединения

Соединение вложенных циклов

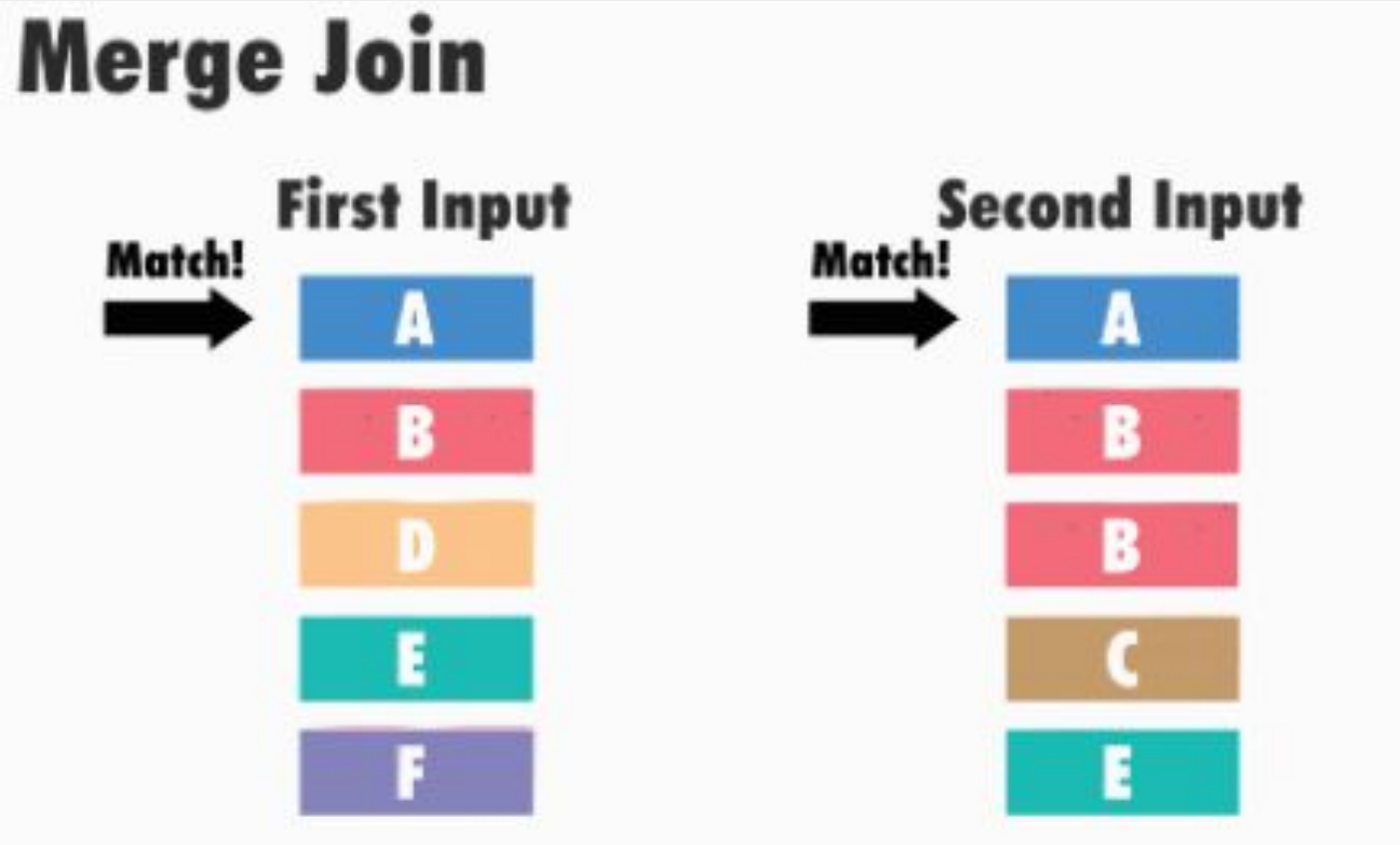


Nested Loops Join работает так: СУБД берет первое значение из первой таблицы (наша "внешняя" таблица выбирается сервером по умолчанию) и сравнивает его с каждым значением во второй "внутренней" таблице в поисках совпадения

```
for each row R1 in the outer table for each row R2 in
the inner table if R1 joins with R2 return (R1, R2)
```

Соединение вложенными циклами. Встречаются очень часто. Выполняют довольно эффективное соединение относительно небольших наборов данных. Соединение вложенными циклами не требует сортировки входных данных. Однако производительность можно улучшить при помощи сортировки источника входных данных; сервер сможет выбрать более эффективный оператор, если оба входа отсортированы. Операция неприменима, если данные слишком велики для хранения в памяти.

МЕТОДЫ СОЕДИНЕНИЙ

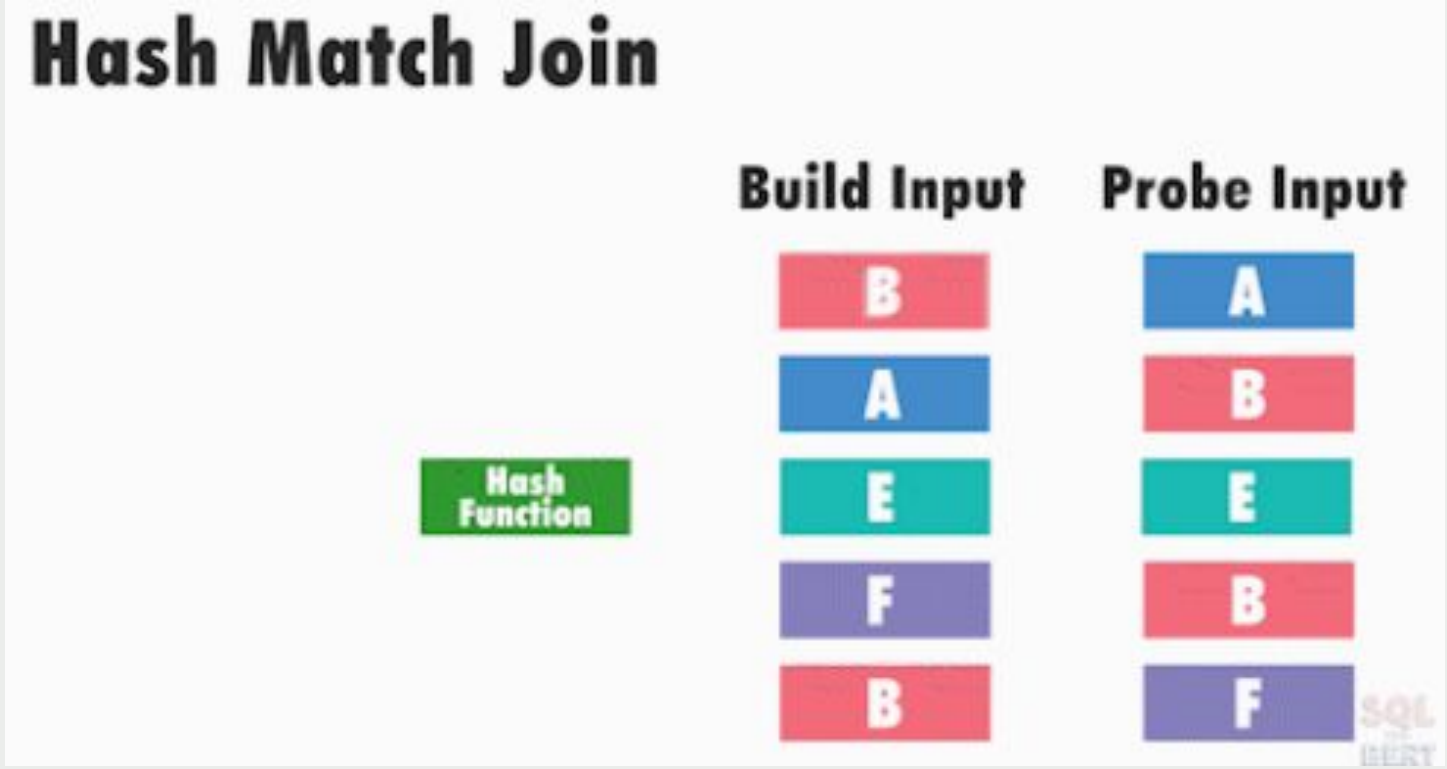


Oracle сравнивает первые строки обоих отсортированных входов.

Затем сравнение продолжается со следующими строками второго входа до тех пор, пока значения соответствуют значению первого входа

Соединение слиянием. Редко встречаются в реальных запросах, как правило, являются наиболее эффективными из операторов логического соединения.
Оптимизатор выбирает использование соединение слиянием, когда входные данные уже **отсортированы** или сервер может выполнить сортировку данных с относительно небольшой стоимостью.
Операция неприменима, если входные данные не отсортированы.

не придется возвращаться и читать неоднократно одни и те же строки



Сервер строит в памяти хэш-таблицу из одного входа (обычно меньшего из двух). Хэши вычисляются на основе ключей соединения входных данных, а затем сохраняются вместе со строкой в хэш-таблице в хэш-блоке, связанном с хэш-ключом

Операция используется всегда, когда невозможно применить другие виды соединения. Она выбираются оптимизатором запросов по одной из двух причин:

1. Соединяемые наборы данных настолько велики, что они могут быть обработаны только с помощью Hash Match Join.
2. Наборы данных не упорядочены по столбцам соединения, и сервер думает, что вычисление хэшей и цикл по ним будет быстрей, чем сортировка данных.

Hash Match Joins (соединения при поиске совпадений в хэше)

Расширенные запросы

Неудобство простого способа выполнения запросов состоит в том, что клиент получает всю выборку сразу, сколько бы строк она не содержала

Для преодоления этого можно:

1. Подготавливать запрос — командой PREPARE и выполняя с помощью EXECUTE
2. Создавать курсор командой DECLARE с последующей выборкой с помощью FETCH. Для клиента это означает заботу об именовании создаваемых объектов, а для сервера — лишнюю работу по разбору дополнительных команд

Подготовка

На этапе *подготовки* запрос разбирается и трансформируется обычным образом, но полученное дерево разбора сохраняется в памяти обслуживающего процесса

```
PREPARE name [ ( data_type [, ...] ) ] AS statement
```

- name: любое имя, присвоенное данному подготовленному оператору, должен быть уникальным в сеансе
- data_type: тип данных параметра подготовленного оператора
- оператор: любой оператор SELECT, INSERT, UPDATE, DELETE или VALUES

```
EXECUTE name [ ( parameter [, ...] ) ]
```

```
PREPARE plane(int) AS
SELECT * FROM book WHERE bshifr = $1;

EXECUTE plane(5);
```

output | Messages | Notifications

bshifr [PK] integer	razdel character varying (50)	kod_avtor integer	nazy character varying (25)	izdat character varying (25)	god_izd integer	price integer
5	романы	2	Роман	Питер	2022	1000

DEALLOCATE используется для освобождения заранее подготовленного

```
DEALLOCATE [ PREPARE ] { name | ALL }
```

Планирование и выполнение

В некоторых случаях планировщик запоминает не только дерево разбора, но и план запроса, чтобы не выполнять планирование повторно

Такой план, построенный без учета значения параметров, называется *общим планом*

Подготовленные операторы с параметрами первые 4 раза всегда оптимизируются с учетом фактических значений; при этом вычисляется средняя стоимость получающихся планов. Начиная с пятого раза, если общий план оказывается в среднем дешевле, чем частные

Если запрос возвращает много строк, и клиенту они нужны все, то огромное значение для скорости передачи данных играет размер выборки, получаемой за один раз. Чем больше выборка, тем меньше коммуникационных издержек на обращение к серверу и получение ответа

Получение результатов

Протокол расширенных запросов позволяет клиенту получать не все результирующие строки сразу, а выбирать данные по несколько строк за раз

Почти тот же эффект дает использование SQL-курсоров

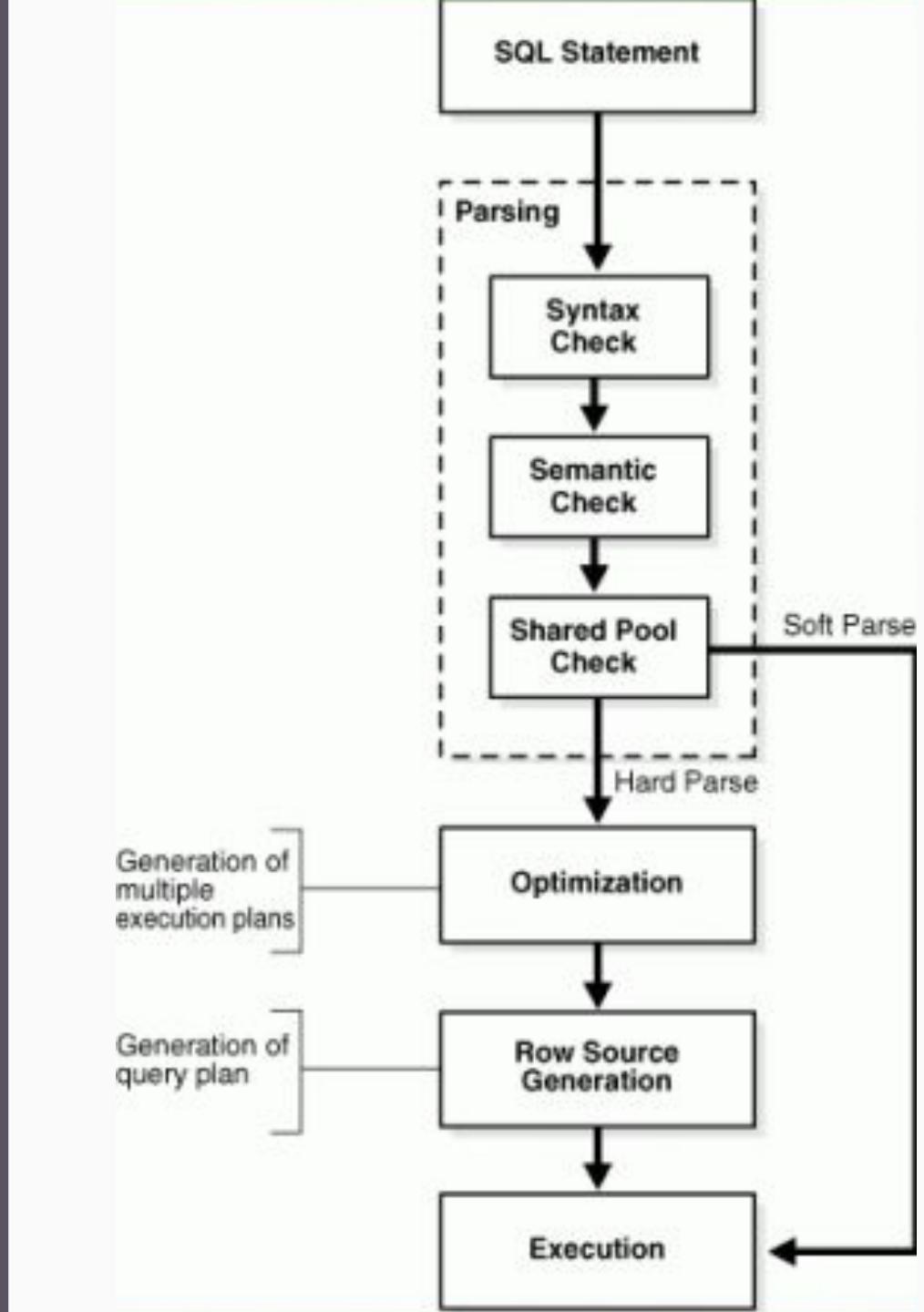
```
BEGIN;
DECLARE cur CURSOR FOR
  SELECT * FROM book ORDER BY nazy;
FETCH 3 FROM cur;

commit;
```

a output | Messages | Explain | Notifications

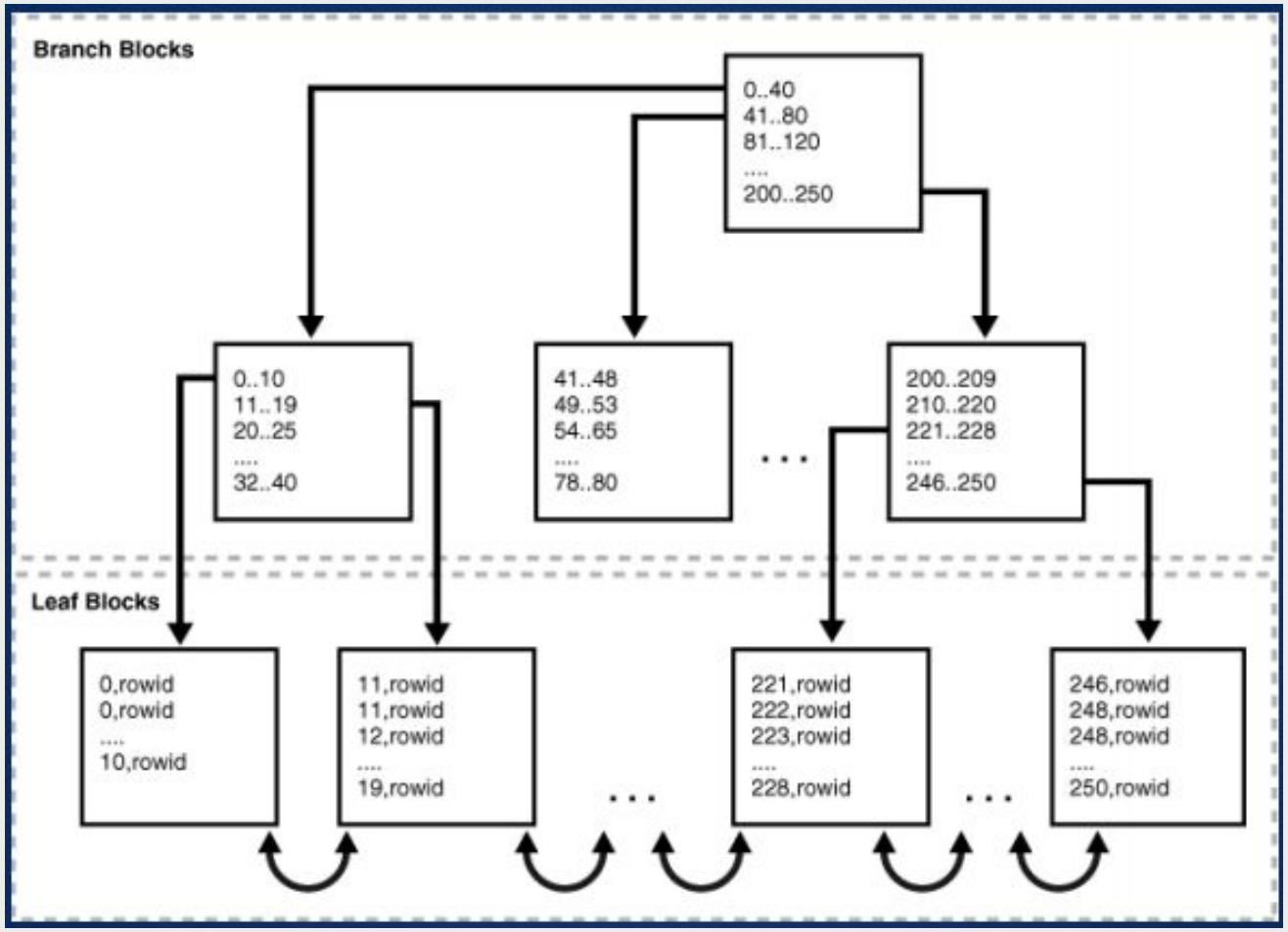
bshifr [PK] integer	razdel character varying (50)	kod_avtor integer	nazy character
4	романы	1	Роман
5	романы	2	Роман

ИНДЕКСЫ И ОПТИМИЗАЦИЯ ЗАПРОСОВ



Доступ к данным

- ✔ По уникальному идентификатору
- ✔ По индексу
- ✔ Полное сканирование таблицы



ИНДЕКСЫ

индекс таблицы должен быть основан на типах запросов, которые будут выполняться над столбцами этой таблицы

Применение индексов

Индексы обеспечивают быстрый доступ к строкам таблиц в базе данных, сохраняя отсортированные значения указанных столбцов и используя эти отсортированные значения для быстрого нахождения ассоциированных строк таблицы

Применение индексов представляет собой компромисс между ускорением получения результатов запросов и замедлением обновлений и вставок данных.

Вообще говоря, если таблицы в основном используются для чтения (выборки) информации, как в хранилищах данных, то лучше иметь много индексов. Если же база данных относится к типу OLTP, с большим количеством вставок, обновлений и удалений, то лучше обойтись меньшим числом индексов.

Виды индексов

- **Уникальные и неуникальные индексы**. Уникальные индексы основаны на уникальном столбце — обычно вроде номера карточки социального страхования сотрудника. Хотя уникальные индексы можно создавать явно, Oracle не рекомендует это делать. Вместо этого следует использовать уникальные ограничения. Когда накладывается ограничение уникальности на столбец таблицы, Oracle автоматически создает уникальные индексы по этим столбцам.
- **Первичные и вторичные индексы**. Первичные индексы — это уникальные индексы в таблице, которые всегда должны иметь какое-то значение и не могут быть равны null. Вторичные индексы — это прочие индексы таблицы, которые могут и не быть уникальными.
- **Составные индексы**. Составные индексы — это индексы, содержащие два или более столбца из одной и той же таблицы. Они также известны как сцепленные индексы (concatenated index). Составные индексы особенно полезны для обеспечения уникальности сочетания столбцов таблицы в тех случаях, когда нет уникального столбца, однозначно идентифицирующего строку.

Индексы и ключи

Без индекса чтение таблицы осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки

Индекс (англ. index) — объект базы данных, создаваемый с целью повышения производительности поиска данных.

Индекс — это физическая структура, хранящаяся в базе данных. Индекс можно создавать, изменять и уничтожать; в основном он служит для ускорения доступа к данным таблицы.

Ключи — полностью логическая концепция. Ключи, с другой стороны, являются чисто логическим концепциями. Они представляют ограничения целостности, создаваемые для реализации бизнес-правил.

Рекомендации по созданию эффективных индексов в базе данных

Индексация имеет смысл, если нужно обеспечить доступ одновременно не более чем к 4–5% данных таблицы

Альтернативой использованию индекса для доступа к данным строки является полное последовательное чтение таблицы от начала до конца, что называется полным сканированием таблицы.

Рекомендации по созданию эффективных индексов в базе данных

- Избегайте создания индексов для сравнительно небольших таблиц. Для таких таблиц больше подходит полное сканирование. В случае маленьких таблиц нет необходимости в хранении данных и таблиц, и индексов
- Создавайте первичные ключи для всех таблиц. При назначении столбца в качестве первичного ключа СУБД автоматически создает индекс по этому столбцу

Рекомендации по созданию эффективных индексов в базе данных

- Индексируйте столбцы, участвующие в многотабличных операциях соединения
- Индексируйте столбцы, которые часто используются в конструкциях *WHERE*
- Индексируйте столбцы, участвующие в операциях *ORDER BY* и *GROUP BY* или других операциях, таких как *UNION* и *DISTINCT*, включающих сортировку. Поскольку индексы уже отсортированы, объем работы по выполнению необходимой сортировки данных для упомянутых операций будет существенно сокращен

Рекомендации по созданию эффективных индексов в базе данных Oracle

- Столбцы, состоящие из длинно-символьных строк, обычно плохие кандидаты на индексацию
- Столбцы, которые часто обновляются, в идеале не должны быть индексированы из-за связанных с этим накладных расходов
- Индексируйте таблицы только с высокой селективностью. То есть индексируйте таблицы, в которых мало строк имеют одинаковые значения
- Сохраняйте количество индексов небольшим
- Составные индексы могут понадобиться там, где одностолбцовые значения сами по себе не уникальны. В составных индексах первым столбцом ключа должен быть столбец с максимальной селективностью

Создание индекса

CREATE INDEX *ИмяИндекса* ON ИмяТаблицы (*ИндексируемыеПоля*)

Способы создание индексов

1. **CREATE INDEX**
2. **ALTER TABLE table_name ADD INDEX [index_name] (index_col_name,...)**

Имя индекса должны быть уникальным среди всех имен индексов БД, а также среди имен ограничений на уровне таблиц.
При создание первичных и внешних ключей система

```sql
create index rep_room on receipt(room_id);

drop index rep_room;

alter table receipt
rename column type_payment to tip_oplat;
```

ДЗ

Написать инструкции SQL для создания индекса в таблице базы данных ИЗ (по выбору), инструкцию для модификации структуры таблицы и добавления индекса

Проверьте выполнение план выполнения запроса с применением индекса

Статистика

Базовая статистика уровня отношения хранится в системном каталоге в таблице **pg_class**

К ней относятся:

- число строк в отношении (reltuples)
- размер отношения в страницах (relpages)
- количество страниц, отмеченных в карте видимости (relallvisible)

```
21  SELECT * FROM pg_class;
```

Data output | Messages | Explain | Notifications

	relname name	relnamespace oid	reltype oid	reloftype oid	relowner oid	relam oid	relfilenode oid	reltablespace oid
1	pg_statist...	11	11319	0	10	0	2619	0
2	avtor	2200	18051	0	16757	0	18049	0
3	avtor_pkey	2200	0	0	16757	403	18054	0
4	pg_toast_...	99	18268	0	18260	0	18267	0
5	pg_toast_...	99	0	0	18260	403	18269	0

Статистика работы PostgreSQL

PostgreSQL собирает статистику с помощью фонового процесса “**stats collector**”(коллектор статистики)

Эта статистика может понадобится для анализа работы сервера PostgreSQL

Статистика ведется с момента первого запуска сервера, а с помощью функции **pg_stat_reset**() её можно сбросить, т.е. обнулить все счетчики.

Это обнулит не все счетчики а только в текущей базе данных

Просмотр статистики для одной таблицы

```
1  SELECT * FROM pg_stat_all_tables where relname='book';
```

Data output | Messages | Notifications

	relid oid	schemaname name	relname name	seq_scan bigint	seq_tup_read bigint	idx_scan bigint	idx_tup_fetch bigint	n_tu bigi
1	18079	public	book	30	35	2	2	

seq_scan – сколько раз выполнялось последовательное чтение всей таблицы;

```
1  select * from book;
2  SELECT * FROM pg_stat_all_tables where relname='book';
3
```

Data output | Messages | Notifications

	relid oid	schemaname name	relname name	seq_scan bigint	seq_tup_read bigint	idx_sc bigint
1	18079	public	book	31	37	

relid – идентификатор базы;

schemaname – имя схемы;

relname – имя таблицы;

seq_scan – сколько раз выполнялось последовательное чтение всей таблицы;

seq_tup_read – количество строк, прочитанных при последовательных чтениях;

idx_scan – количество сканирований по индексу;

idx_tup_fetch – количество строк, отобранных при сканированиях по индексу;

n_tup_ins – количество вставленных строк;

n_tup_upd – количество обновлённых строк (UPDATE);

n_tup_del – количество удалённых строк;

n_tup_hot_upd – количество строк, обновлённых в режиме HOT (без отдельного изменения индекса);

n_live_tup – оценочное количество строк;

n_dead_tup – оценочное количество “мёртвых” строк;

n_mod_since_analyze – оценочное число строк, изменённых в этой таблице, с момента последнего сбора статистики;

n_ins_since_vacuum – примерное число строк, вставленных в эту таблицу с момента последнего сбора статистики;

last_vacuum – когда последний раз работал VACUUM;

last_autovacuum – когда последний раз работал AUTOVACUUM;

last_analyze – когда последний раз VACUUM собирал статистику;

last_autoanalyze – когда последний раз AUTOVACUUM собирал статистику;

vacuum_count – сколько раз VACUUM выполнялся;

autovacuum_count – сколько раз AUTOVACUUM выполнялся;

analyze_count – сколько раз вручную собирали статистику;

autoanalyze_count – сколько раз AUTOVACUUM собирал статистику.

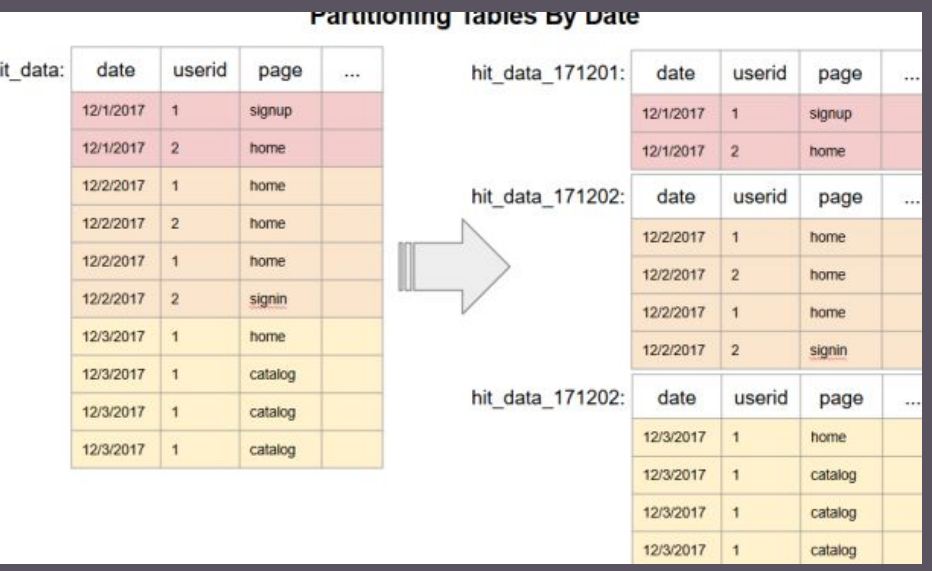
ПРОСМОТР СТАТИСТИКИ ПО БАЗЕ ДАННЫХ

```
5  SELECT * FROM pg_stat_database WHERE datname='db';
6
```

Data output | Messages | Notifications

	datid oid	datname name	numbackends integer	xact_commit bigint	xact_rollback bigint	blks_read bigint	blks_hit bigint	tup_returned bigint
1	16792	db	4	193412	260	2306	7132460	99236202

- **tup_inserted** – сколько строк было вставлено;
- **tup_updated** – сколько строк было изменено;
- **blks_read** – сколько страниц было прочитано с диска;
- **blks_hit** – сколько страниц было прочитано из буферного кэша;
- **numbackends** – сколько сейчас клиентов подключено к базе данных;
- **xact_commit** – сколько транзакций было успешно завершено;
- **xact_rollback** – сколько транзакций было откачено.



ПАРТИЦИОНИРОВАНИЕ

Партиционирование — это разбиение таблиц, содержащих большое количество записей, на логические части по неким выбранным администратором критериям.

Партиционирование таблиц делит весь объем операций по обработке данных на несколько независимых и параллельно выполняющихся потоков, что существенно ускоряет работу СУБД.

Методы секционирования

- ✔ **Фрагментация по диапазону значений**
- ✔ **Фрагментация по списку значений**
- ✔ **Фрагментация с использованием хэш функции**

Задачи, решаемые секционированием

SQL-запросов и DML-операций по модификации строк таблицы достигается за счет того, что поиск и модификация строк в таблице идут не по всей таблице, а только в ее части (в одной или нескольких секциях)

2. Разбиение таблицы на секции позволяет увеличит скорость обработки таблицы за счет использования параллелизма
3. Быстрое удаление значительного числа строк в больших таблицах за счет выполнения операции truncate

Фрагментация – разделение таблицы или индекса на несколько логически связанных частей , фрагментов, секций с неким общим признаком

Фрагментация по диапазону значений
Данные относящиеся таблицы , где значения в заданных колонках относятся к некоторому диапазону распределяются по соответствующим фрагментам(секциям) таблицы.

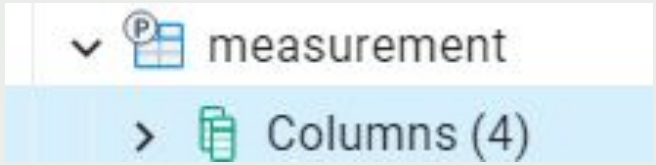
Фрагментация по списку значений
Фрагмент(секция) определяется по элементу списка , такой способ фрагментации идеально подходит , когда в заданной колонке используется ограниченное число значений.

Фрагментация с использованием хэш функции
По данным заданных столбцов таблицы , Oracle вычисляет значение специальной хэш функции на основании которого определяет в какой именно фрагмент таблицы поместить заданную запись

```
select * from
 т а б л и ц а
partition
(ф р а г м е н
 т );
```

С помощью оператора **SELECT** есть возможность выбирать как все данные из фрагментированной таблицы, так и использовать **SELECT** для выбора данных из заданного фрагмента таблицы.

```
CREATE TABLE measurement (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
) PARTITION BY RANGE (logdate);
```

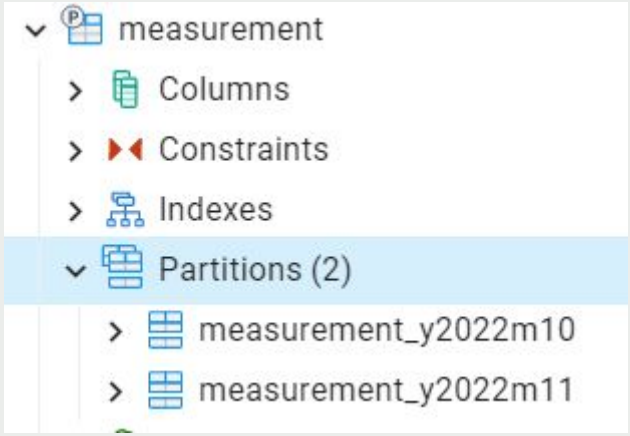


Пример

Предположим, что мы создаём базу данных для большой компании, торгующей мороженым. Компания учитывает максимальную температуру и продажи мороженого каждый день в разрезе регионов

Создадим таблицу measurement как секционированную таблицу с предложением PARTITION BY, указав метод разбиения (в нашем случае RANGE) и список столбцов, которые будут образовывать ключ разбиения

Создание секций



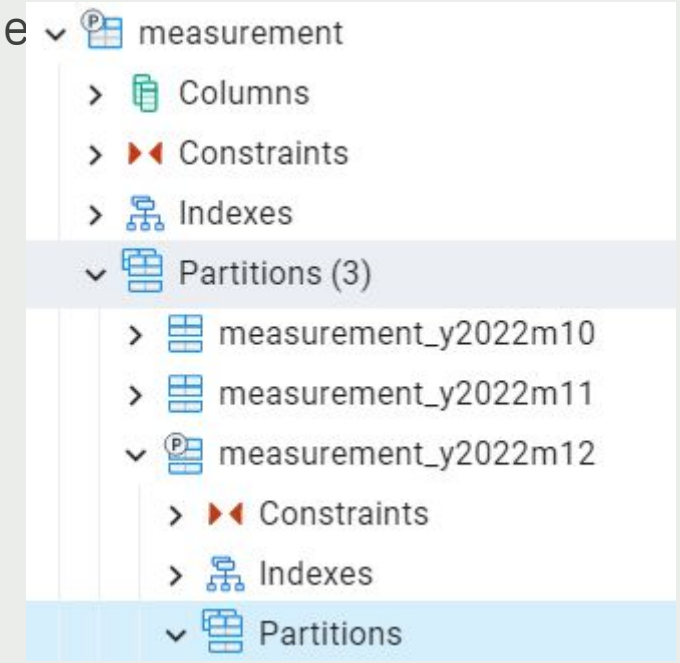

```
CREATE TABLE measurement_y2022m10 PARTITION OF measurement
    FOR VALUES FROM ('2022-10-01') TO ('2022-11-01');

CREATE TABLE measurement_y2022m11 PARTITION OF measurement
    FOR VALUES FROM ('2022-11-01') TO ('2022-12-01');
```

В нашем примере каждая секция должна содержать данные за один месяц, чтобы данные можно было удалять по месяцам согласно требованиям

границы соседних секций могут определяться одинаковыми значениями, так как верхние границы не включаются в диапазон

Если вы хотите реализовать вложенное секционирование, дополнительно укажите предложение PARTITION BY в командах, создающих отдельные се



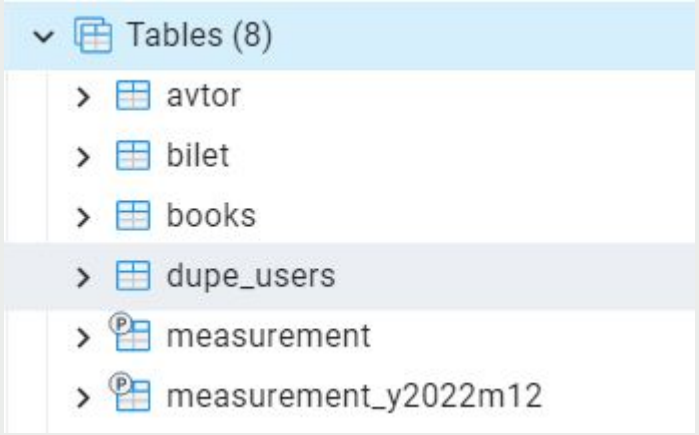

```
CREATE TABLE measurement_y2022m12 PARTITION OF measurement
    FOR VALUES FROM ('2022-12-01') TO ('2023-01-01')
    PARTITION BY RANGE (peaktemp);
```

measurement и попадающие в measurement_y2022m12 будут затем перенаправлены в одну из вложенных секций в зависимости от значения столбца peaktemp

Указанный ключ разбиения может пересекаться с ключом разбиения родителя, хотя определять границы вложенной секции нужно осмотрительно, чтобы множество данных, которое она принимает, входило во множество, допускаемое собственными границами секции; система не пытается контролировать это сама

При добавлении в родительскую таблицу данных, которые не соответствуют ни одной из существующих секций, произойдёт ошибка; подходящую секцию нужно

Обслуживание секций




```
CREATE INDEX ON measurement (logdate);
```

Обычно набор секций, образованный изначально при создании таблиц, не предполагается сохранять неизменным. Чаще наоборот, планируется удалять секции со старыми данными и периодически добавлять секции с новыми

Самый лёгкий способ удалить старые данные — просто удалить секцию

Ещё один часто более предпочтительный вариант — убрать секцию из главной таблицы, но сохранить возможность обращаться к ней как к самостоятельной таблице

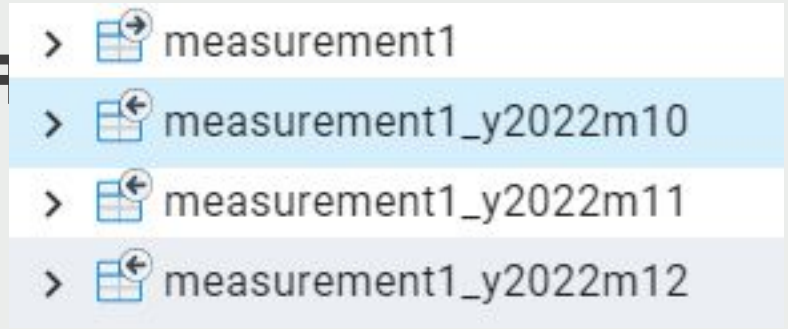
```
ALTER TABLE measurement DETACH PARTITION measurement_y2022m12;
```

Ограничения

первичные ключи) в секционированных таблицах должны включать все столбцы ключа разбиения. Это требование объясняется тем, что отдельные индексы, образующие ограничение, могут непосредственно обеспечивать уникальность только в своих секциях. Поэтому сама структура секционирования должна гарантировать отсутствие дубликатов в разных секциях.

- **Создать ограничение-исключение**, охватывающее всю секционированную таблицу, нельзя; можно только поместить такое ограничение в каждую отдельную секцию с данными. И это также является следствием того, что установить ограничения, действующие между секциями, невозможно.
- Триггеры BEFORE ROW для INSERT не могут менять секцию, в которую в итоге попадёт новая строка
- Смешивание временных и постоянных отношений в одном дереве секционирования не допускается. Таким образом, если секционированная таблица постоянная, такими же должны быть её секции; с

а н и е с и с п о л ь з о в а н и е м н а с л е д о в а н и я




```
CREATE TABLE measurement1 (
    city_id         int not null,
    logdate         date not null,
    peaktemp        int,
    unitsales       int
);

CREATE TABLE measurement1_y2022m10 () INHERITS (measurement1);
CREATE TABLE measurement1_y2022m11 () INHERITS (measurement1);
CREATE TABLE measurement1_y2022m12 () INHERITS (measurement1);
```

Добавим в дочерние таблицы неперекрывающиеся ограничения, определяющие допустимые значения ключей для каждой из них

```
CREATE TABLE measurement1_y2022m10 (
    CHECK ( logdate >= DATE '2022-10-01' AND logdate < DATE '2022-11-01' )
) INHERITS (measurement1);

CREATE TABLE measurement1_y2022m11 (
    CHECK ( logdate >= DATE '2022-11-01' AND logdate < DATE '2022-12-01' )
) INHERITS (measurement1);
```

Перенаправление добавляемых строк в соответствующую дочернюю таблицу можно реализовать определив триггер для главной таблицы или правила

Правила

```
CREATE RULE measurement_insert_y2022m10 AS
ON INSERT TO measurement1 WHERE
    ( logdate >= DATE '2022-10-01' AND logdate < DATE '2022-11-01' )
DO INSTEAD
    INSERT INTO measurement1_y2022m10 VALUES (NEW.*);
```

Для каждой дочерней таблицы создайте индекс по ключевому столбцу (или столбцам), а также любые другие индексы по своему усмотрению

```
CREATE INDEX measurement1_y2022m10_logdate ON measurement1_y2022m10 (logdate);
CREATE INDEX measurement1_y2022m11_logdate ON measurement1_y2022m11 (logdate);
```

Оптимизация запросов к секционным таблицам

Устранение секций — это приём оптимизации запросов, который ускоряет работу с декларативно секционированными таблицами

Когда устранение секций включено, планировщик рассматривает определение каждой секции и может заключить, что какую-либо секцию сканировать не нужно, так как в ней не может быть строк, удовлетворяющих предложению WHERE в запросе

```
SET enable_partition_pruning = on;                    -- по умолчанию
SELECT count(*) FROM measurement WHERE logdate >= DATE '2022-01-01';
```

```
SET enable_partition_pruning = off;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2022-10-01';
```

a output | Messages | Graph Visualiser × | Explain × | Notifications

QUERY PLAN text
Aggregate (cost=62.55..62.56 rows=1 width=8)
-> Append (cost=8.93..59.46 rows=1234 width=0)
-> Bitmap Heap Scan on measurement_y2022m10 (cost=8.93..26.65 rows=617 width=0)
Recheck Cond: (logdate >= '2022-10-01'::date)
-> Bitmap Index Scan on measurement_y2022m10_logdate_idx (cost=0.00..8.78 rows=617 width=0)
Index Cond: (logdate >= '2022-10-01'::date)
-> Bitmap Heap Scan on measurement_y2022m11 (cost=8.93..26.65 rows=617 width=0)
Recheck Cond: (logdate >= '2022-10-01'::date)
-> Bitmap Index Scan on measurement_y2022m11_logdate_idx (cost=0.00..8.78 rows=617 width=0)
Index Cond: (logdate >= '2022-10-01'::date)

Оптимизация запросов к секционным таблицам

Исключение по ограничению — приём оптимизации запросов, подобный устранению секций. Прежде всего он применяется, когда секционирование осуществляется с использованием старого метода наследования, но он может быть полезен и для других целей, включая декларативное секционирование

Исключение по ограничению работает во многом так же, как и устранение секций; отличие состоит в том, что оно использует ограничения CHECK всех таблиц (поэтому оно так и называется), тогда как для устранения секций используются границы секции, которые существуют только в случае декларативного секционирования

Ещё одно различие состоит в том, что исключение по ограничению применяется только во время планирования; во время выполнения секции из плана удаляться не будут

ДЗ

Подготовка к устному зачёту