

# SQL

## Занятие 8

# Индексы, партиционирование Функции, триггеры

Бояр Владислав

Индексы

# Индекс

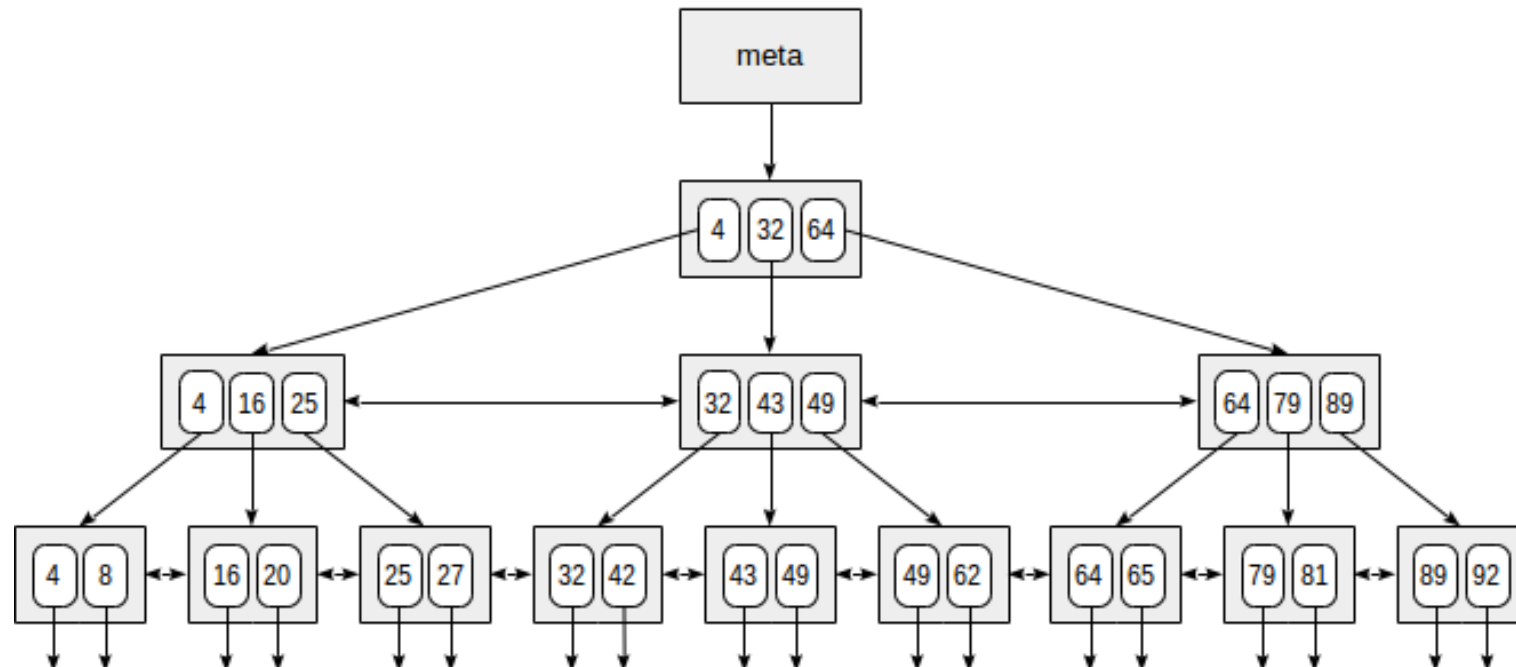
- **Индекс** - объект БД, предназначенный для ускорения доступа к данным;
- **Создание индекса** - одно из основных средств оптимизации запросов;
- Индекс, ускоряя доступ к данным, взамен требует затрат на своё содержание:
  - индекс занимает место не диске;
  - любая операция над проиндексированными данными (вставка, обновление, удаление) приводит к перестраиванию индекса в этой же транзакции.

# Индекс B-tree (бинарное дерево)

- Наиболее часто используемый тип индекса;
- Используется для данных, которые можно отсортировать;
- Позволяет производить поиск данных наподобие оглавления в телефонном справочнике;
- При любом способе сканирования (индексном, исключительно индексном, по битовой карте) метод доступа b-tree возвращает упорядоченные данные.

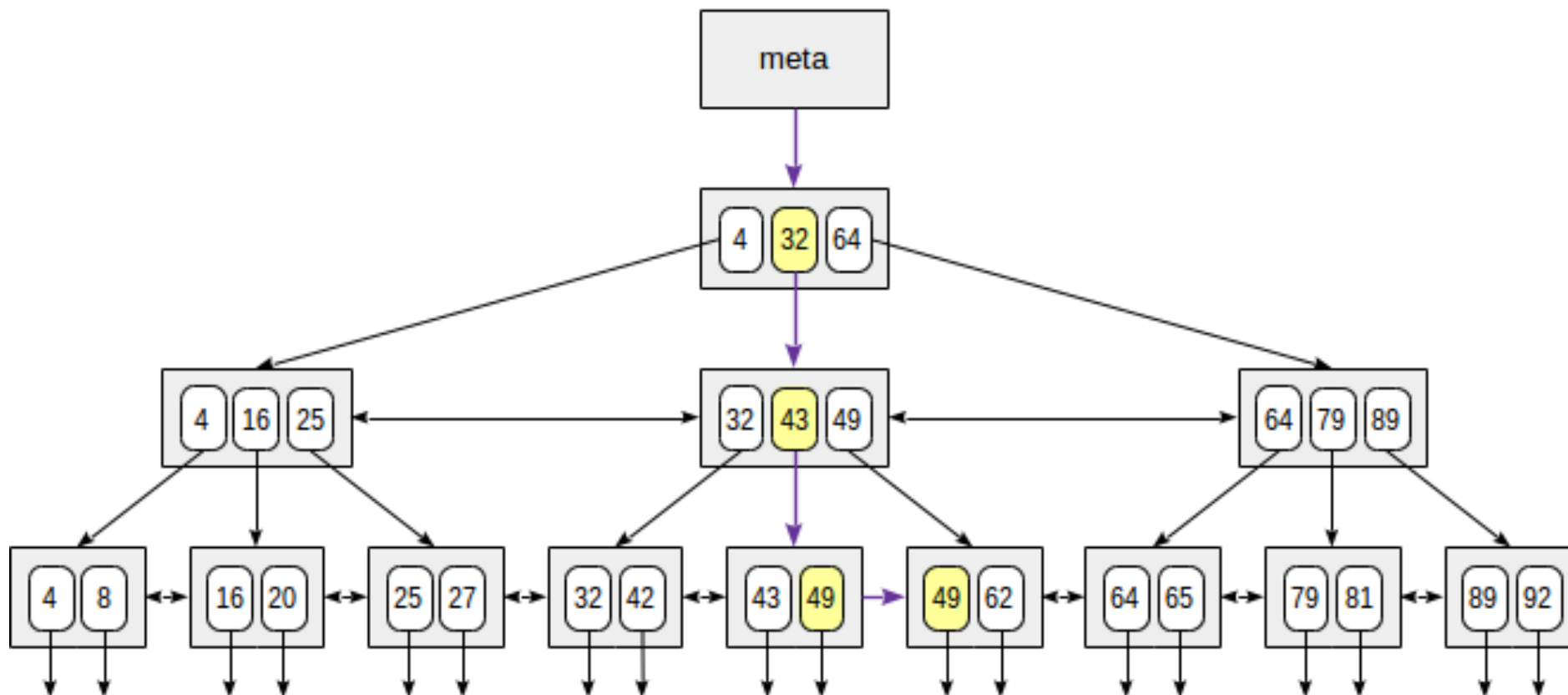
# Пример индекса по одному полю с целочисленными ключами

В самом начале файла находится метастраница, которая ссылается на корень индекса. Ниже корня расположены внутренние узлы; самый нижний ряд — листовые страницы. Стрелочки вниз символизируют ссылки из листовых узлов на строки таблицы (TID - **t**uple **i**dentifier).



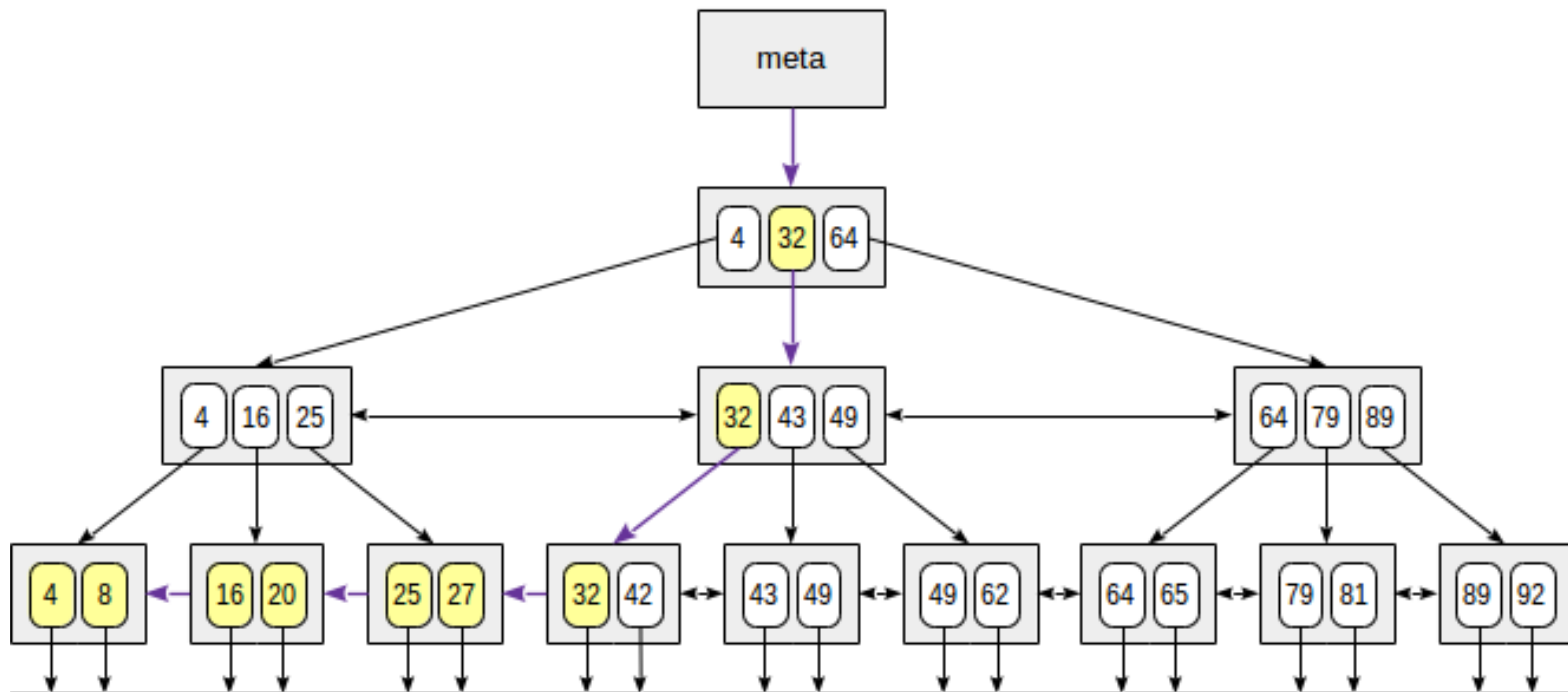
# Поиск по равенству

- Индексированный атрибут = 49



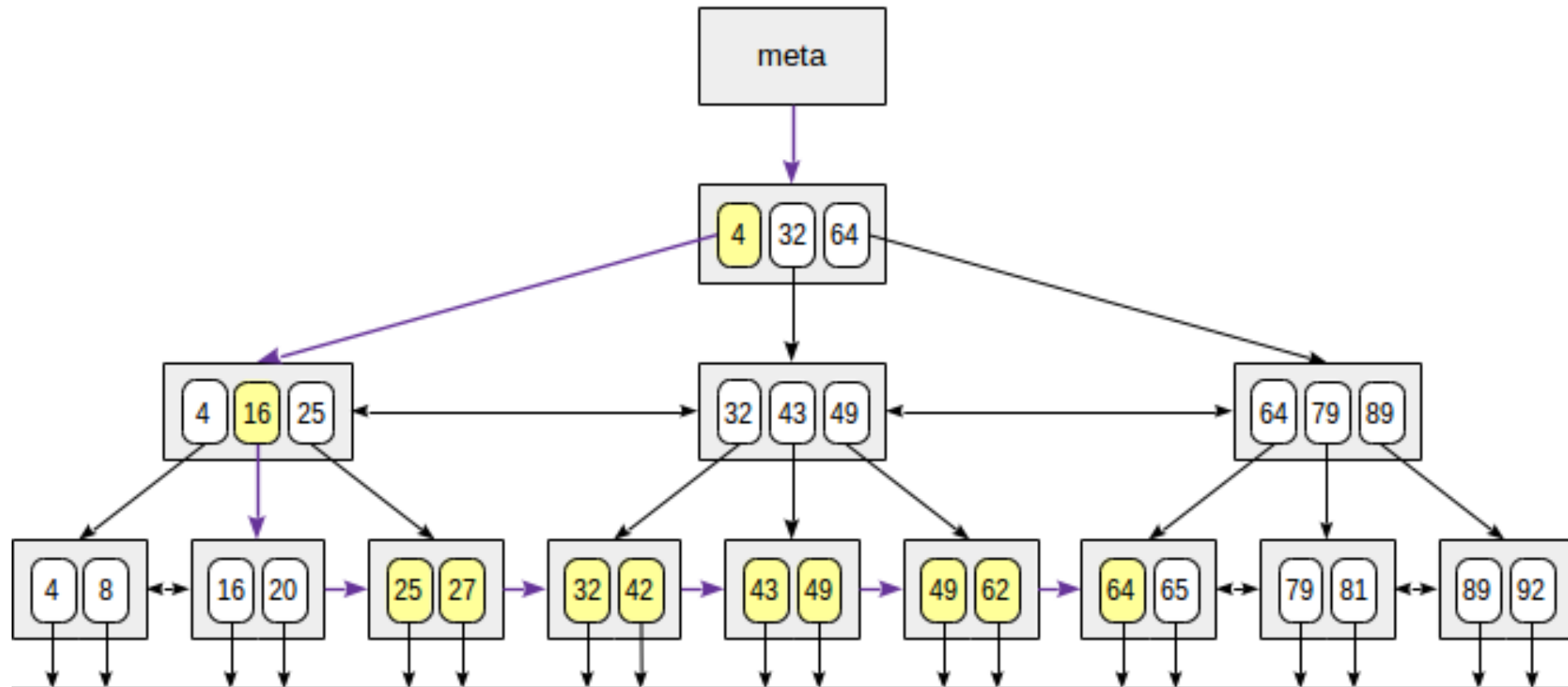
# Поиск по неравенству

- Индексированный атрибут  $\leq 35$



# Поиск по диапазону


- $23 \leq \text{Индексированный атрибут} \leq 64$





# Работа с индексами при масштабных изменениях данных

Если предстоит крупная вставка или обновление таблицы (более 100 тыс. записей), то оптимальнее:

- Удалить все индексы
  - Произвести вставку/обновление записей
  - Создать индексы на новых данных
- 

# Почему нельзя создать индексы на все атрибуты?

- Индексы занимают дисковое пространство (если сделать индексы по всем полям, то они занимают больше места, чем исходная таблица);
- Индексы утяжеляют операции над проиндексированными данными;

# Операции с индексом

Создание индекса:

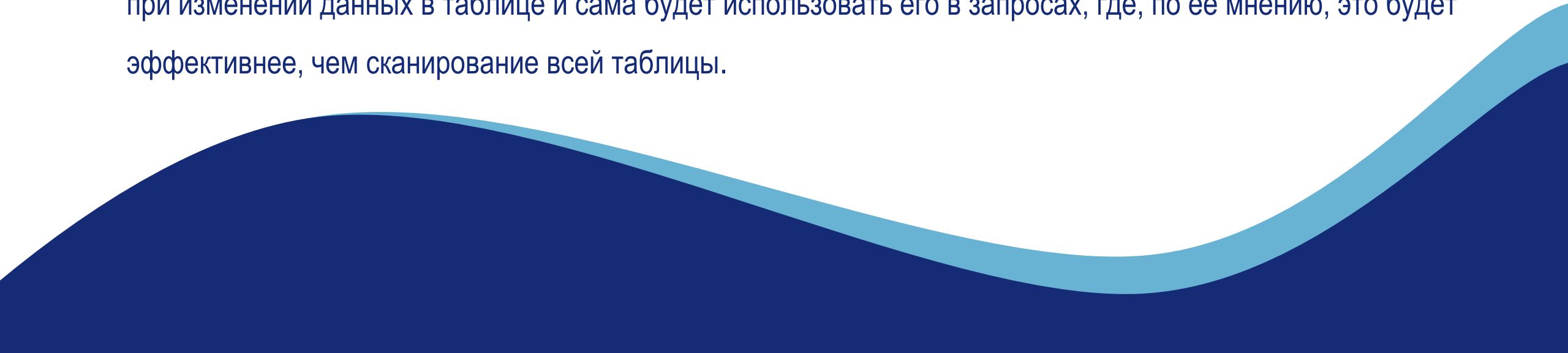
```
CREATE INDEX table_name_column_name_index ON table_name (column_name);
```

Название индекса может быть произвольным.

Удаление индекса:

```
DROP INDEX table_name_column_name_index;
```

Когда индекс создан, никакие дополнительные действия не требуются: система сама будет обновлять его при изменении данных в таблице и сама будет использовать его в запросах, где, по её мнению, это будет эффективнее, чем сканирование всей таблицы.




# Операции с индексом

Индекс можно создавать по нескольким полям:


```
CREATE INDEX table_1_c1c2_idx ON table_1 (c1, c2);
```

Актуально, если эти поля часто используются в операциях объединения или они часто одновременно участвуют в запросах в качестве фильтра:

```
SELECT c_name FROM table_1 WHERE c1 = ... AND c2 = ... ;
```



# Индекс и сортировки

- Помимо поиска строк для выдачи в результате запроса, индексы также могут применяться для сортировки строк в определённом порядке.
  - Это позволяет учесть предложение ORDER BY в запросе, не выполняя сортировку дополнительно.
  - Из всех типов индексов, которые поддерживает PostgreSQL, сортировать данные могут только В-деревья — индексы других типов возвращают строки в неопределённом, зависящем от реализации порядке.
  - Особый случай представляет ORDER BY в сочетании с LIMIT: при явной сортировке системе потребуется обработать все данные, чтобы выбрать первые n строк, но при наличии индекса, первые n строк можно получить сразу, не просматривая остальные вовсе.
- 

# Партиционирование



# Партиционирование

- Партиционирование (секционирование) – разбиение одной большой логической таблицы на несколько меньших физических таблиц (партиций / секций).
- Является способом ускорения доступа к данным.
- Применяется, когда:
  - таблица содержит много данных (десятки млн и более);
  - запросы чаще всего производятся к определённым данным (к примеру, с фильтрацией по дате);

# Партиционирование

stackoverflow.questions_2018		
Creation_date	Title	Tags
2018-03-01	How do I??	Android
2018-03-01	When Should?	Linux
2018-03-02	This is great!	Linux
2018-03-03	Can this?	C++
2018-03-02	Help!!	Android
2018-03-01	What does?	Android
2018-03-02	When does?	Android
2018-03-02	Can you help?	Linux
2018-03-02	What now?	Android
2018-03-03	Just learned!	SQL
2018-03-01	How does!	SQL



20180301	stackoverflow.questions_2018_partitioned		
	Creation_date	Title	Tags
	2018-03-01	How do I??	Android
	2018-03-01	When Should?	Linux
	2018-03-01	What does?	Android
20180302	2018-03-01	How does!	SQL
	Creation_date	Title	Tags
	2018-03-02	This is great!	Linux
	2018-03-02	Help!!	Android
	2018-03-02	When does?	Android
20180303	2018-03-02	Can you help?	Linux
	2018-03-02	What now?	Android
	Creation_date	Title	Tags
20180303	2018-03-03	Can this?	C++
	2018-03-03	Just learned!	SQL



# Партиционирование

- Партиционированная таблица — это виртуальная таблица, в которой нет строк.
- Партиции — это обычные таблицы, связанные с партиционированной.
- Каждая партиция хранит подмножество строк, определяемое значениями ключа партиционирования.
- Строки вставляются в соответствующую партицию на основе ключа партиционирования.
- Если в строке обновляется значение ключа партиции и он больше не соответствует значениям партиции, строка перемещается в другую партицию.

# Плюсы партиционирования

- Если данные запрашиваются из определённой партии (к примеру, за определённый месяц/год), то поиск данных в этой партии быстрее, чем поиск по индексу;
- Нет минусов индексов (дополнительное место на диске, тяжелые операции вставки и обновления);
- Добавить/удалить партию быстрее, чем производить последовательную вставку или удаление;

# Особенности партиционирования

- Не путать с PARTITION BY в оконных функциях 😊
- К партиции можно обращаться как к обычной таблице;
- Партиций не должно быть очень много (несколько тысяч) или очень мало (две);
- Первичные ключи в партиционированных таблицах не поддерживаются;
- На партиционированные таблицы не могут ссылаться внешние ключи;
- Преобразовать обычную таблицу в партиционированную и наоборот нельзя;
- В партиционированную таблицу можно добавить в качестве партиции обычную или партиционированную таблицу с данными;
- Можно удалить партицию из партиционированной таблицы и превратить её в отдельную таблицу.

# Виды партиционирования

- **RANGE (по диапазону)** - таблица партиционируется по «диапазонам», определённым по ключевому столбцу. Диапазоны не должны пересекаться друг с другом. Например, можно секционировать данные по диапазонам дат или по диапазонам идентификаторов.
- **LIST (по списку)** - таблица партиционируется с помощью списка, явно указывающего, какие значения ключа должны относиться к каждой партиции (например, список аэропортов).
- **HASH (по хэшу)** - таблица партиционируется по определённым модулям и остаткам, которые указываются для каждой партиции. Каждая партиция содержит строки, для которых значение ключа разбиения, разделённое на модуль, равняется заданному остатку. Используется, когда нужно равномерно распределить строки по партициям, а таблица не имеет подходящего ключа партиционирования.

# Партиционирование по диапазону

Создание партиционированной таблицы:

```
CREATE TABLE sales (  
  id int,  
  product text,  
  sale_date date  
) PARTITION BY RANGE (sale_date);
```

Создание партиций:

```
CREATE TABLE sales_y2024m01 PARTITION OF sales  
  FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');  
CREATE TABLE sales_y2024m02 PARTITION OF sales  
  FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');  
CREATE TABLE sales_default PARTITION OF sales  
  DEFAULT;
```

# Партиционирование по списку

Создание партиционированной таблицы:

```
CREATE TABLE books (  
  id int,  
  author text,  
  genre date  
) PARTITION BY LIST (genre);
```

Создание партиций:

```
CREATE TABLE books_novel PARTITION OF sales  
  FOR VALUES IN ('novel');  
CREATE TABLE books_detective PARTITION OF sales  
  FOR VALUES IN ('detective');  
CREATE TABLE sales_default PARTITION OF sales  
  DEFAULT;
```

# Партиционирование по хэшу

Создание партиционированной таблицы:

```
CREATE TABLE books (  
    code          char(5),  
    title         varchar(40),  
    delivery_date date,  
    genre         varchar(10)  
) PARTITION BY HASH (code);
```

Создание партиций:

```
CREATE TABLE books_part1 PARTITION OF books FOR VALUES WITH (MODULUS 5,  
REMAINDER 0);  
CREATE TABLE books_part2 PARTITION OF books FOR VALUES WITH (MODULUS 5,  
REMAINDER 1);  
CREATE TABLE books_part3 PARTITION OF books FOR VALUES WITH (MODULUS 5,  
REMAINDER 2);  
CREATE TABLE books_part4 PARTITION OF books FOR VALUES WITH (MODULUS 5,  
REMAINDER 3);  
CREATE TABLE books_part5 PARTITION OF books FOR VALUES WITH (MODULUS 5,  
REMAINDER 4);
```

# Обслуживание партиций

Удаление партиции:

```
DROP TABLE sales_y2024m01;
```

Убрать партицию из главной таблицы, но сохранить возможность обращаться к ней как к самостоятельной таблице:

```
ALTER TABLE sales DETACH PARTITION sales_y2024m01;
```





# Партиционирование по диапазону (GreenPlum)

```
CREATE TABLE book_order
(id INT,
book_id INT,
client_id INT,
book_count SMALLINT,
order_date DATE)
WITH (appendoptimized=true, orientation=row, compresstype=ZLIB,
compresslevel=5)
DISTRIBUTED BY(id)
PARTITION BY RANGE(order_date)
(START(date '2022-01-01') INCLUSIVE
END(date '2023-01-01') EXCLUSIVE
EVERY(INTERVAL '1 month'),
DEFAULT PARTITION other);
```

# Функции

# Функция

- SQL-функции выполняют произвольный список операторов SQL;
- Тело SQL-функции должно представлять собой список SQL-операторов, разделённых точкой с запятой. Точка с запятой после последнего оператора может отсутствовать.
- 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.

```
CREATE FUNCTION one() RETURNS integer AS $$  
SELECT 1 AS result;  
$$ LANGUAGE SQL;
```

# Синтаксис функции (пример)

**CREATE OR REPLACE FUNCTION**

schema\_name.function\_name (p\_table **text**, p\_schema **text**)

**RETURNS void**

**LANGUAGE** plpgsql -- PL/pgSQL – процедурный язык SQL

**AS \$\$**

**declare**

-- Объявление переменных

v\_var\_name\_1 **text**;

v\_var\_name\_2 **int**;

**begin**

-- Тело функции

**end;**

**\$\$**

**EXECUTE ON ANY;**

# Синтаксис функции (пример)

-- способ присвоения значения переменной напрямую


```
v_var_name_1 := 'some_text';
```

-- способ присвоения значения переменной через запрос

```
select column_1  
from schema_name.table_name  
into v_var_name_2;
```

-- Оператор условия

```
if v_var_name_1 not in ('text_1', 'text_2') then ...;  
end if;
```



# Синтаксис функции (пример)

-- Цикл

```
for v_date_range in ...  
loop
```

```
    if var_name_1 not in ('text_1', 'text_2') then  
        v_var_name_2 := 1;  
    end if;
```

-- Выполнение sql-запросов в теле функции с использованием переменных

```
v_sql := 'select count(*) from ' || p_schema || '.' || p_table;  
execute v_sql;
```

```
end loop;
```



Практика




Напишем функцию и триггер, которые будут сохранять в отдельном справочнике список студентов.

1. Создадим таблицу, которая будет хранить список студентов;
2. Напишем функцию, которая будет:
  1. Проверять, существует ли уже такой студент в списке студентов;
  2. Если нет, то вставлять новое значение в список;
3. Напишем триггер, которые будет вызывать функцию из пункта 2, при осуществлении всех операций вставки и обновления данных (Insert / Update).



```
CREATE OR REPLACE FUNCTION public.check_new_students()  
RETURNS trigger  
LANGUAGE plpgsql  
AS $function$  
BEGIN  
-- IF tg_op = 'INSERT' THEN  
-- Check values in merchant_list  
IF NEW."name" IS NOT NULL AND (SELECT sd.id  
                                FROM public.students_dict sd  
                                WHERE sd.student_name = NEW."name") IS NULL  
    THEN INSERT INTO public.students_dict (student_name)  
    VALUES (NEW."name");  
END IF;  
-- END IF;  
RETURN NEW;  
END;  
$function$  
;
```



```
CREATE TRIGGER insert_new_merchants AFTER  
INSERT  
OR  
UPDATE  
ON  
public.student_grades FOR EACH ROW  
WHEN ((pg_trigger_depth() = 0))  
EXECUTE FUNCTION public.check_new_students();
```

```
-- Включить триггер
```

```
ALTER TABLE public.student_grades ENABLE TRIGGER check_new_students;
```

```
-- Выключить триггер
```

```
ALTER TABLE public.student_grades DISABLE TRIGGER check_new_students;
```

