

Занятие 5

Оптимизация запросов, функции и триггеры

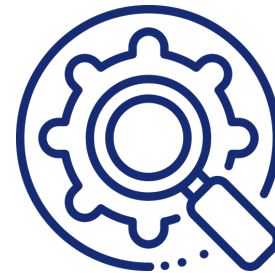
Бояр Владислав

Занятие состоит из:



Теория:

- Физическое хранение данных PostgreSQL;
- Оптимизация SQL-запросов;
- Функции и триггеры.



Практика:

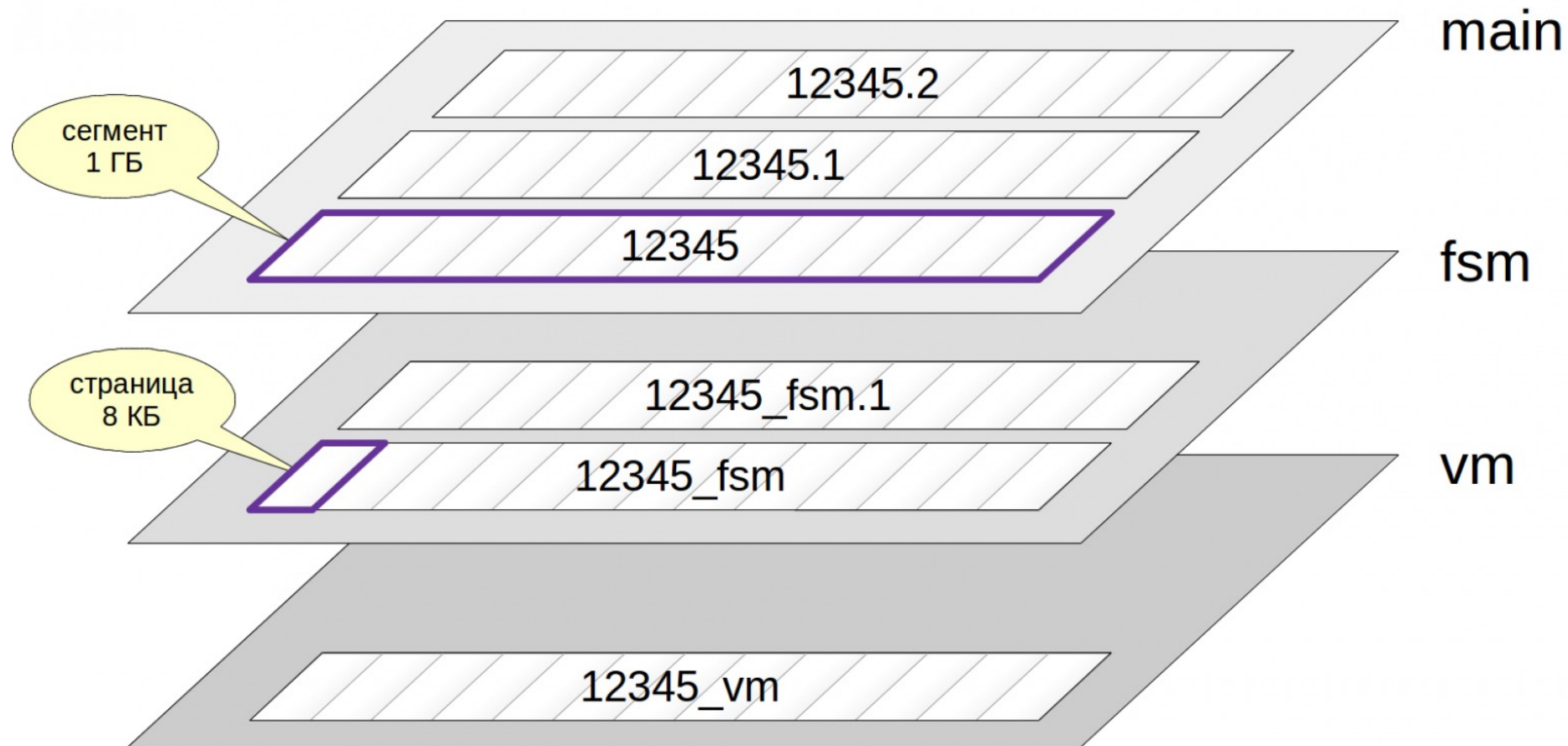
- Примеры оптимизации запросов;
- Реализация функций и триггеров.

Физическое хранение данных в PostgreSQL


Физическое хранение данных в PostgreSQL

- Каждому отношению (таблица/индекс/представление) соответствует несколько слоёв (forks)
- Каждый слой изначально состоит из одного файла (сегмента), затем, когда его размер достигает 1 ГБ, создаётся следующий сегмент.
(Ограничение возникло исторически для поддержки различных файловых систем, некоторые из которых не умеют работать с файлами большого размера. Его можно изменить при сборке инстанса PostgreSQL)
- Файлы (сегменты) разделены на страницы (блоки) по 8 КБ.

Физическое хранение данных в PostgreSQL



Слои PostgreSQL

1. **Основной слой (main)** — содержит данные или индексные строки
 2. **Слой инициализации** — для нежурналируемых таблиц
 3. **Карта свободного пространства (Free Space Map, FSM)** — слой, в котором отслеживается наличие пустого места внутри страниц
 4. **Карта видимости (Visibility Map, VM)** — слой для отслеживания страниц, содержащих только видимые кортежи
- 


Слои PostgreSQL

1. **Основной слой (main)** — содержит данные или индексные строки.

- Создаётся для любых отношений, кроме представлений;
- Имена файлов основного слоя состоят только из числового идентификатора.
- Например, путь к файлу таблицы **bookings.flights**:
`SELECT pg_relation_filepath('bookings.flights');`

Слои PostgreSQL

2. Слой инициализации

- существует только для нежурналируемых таблиц (созданных с указанием UNLOGGED) и их индексов
 - unlogged tables отличаются тем, что данные о транзакциях не логируются в журнале.
 - Работают быстрее обычных таблиц;
 - В случае сбоя невозможно восстановить данные в согласованном состоянии.
- 


Слои PostgreSQL

3. Карта свободного пространства (**Free Space Map, FSM**) — слой, в котором отслеживается наличие пустого места внутри страниц. Это место постоянно изменяется: при добавлении новых версий строк уменьшается, при очистке — увеличивается.

- FSM хранится рядом с данными главного отношения в отдельном слое, имя которого образуется номером файлового узла отношения с суффиксом `_fsm`.
- Файл FSM создаётся не сразу, а только при необходимости. Самый простой способ добиться этого — выполнить очистку таблицы: =>


```
VACUUM bookings.flights;
```

Слои PostgreSQL

4. **Карта видимости (Visibility Map, VM)** — слой для отслеживания страниц, содержащих только видимые кортежи.
- Используется при сканировании данных отношения.
 - Хранится вместе с данными главного отношения в отдельном файле, имя которого образуется номером файлового узла отношения с суффиксом `_vm`. Например, если файловый узел отношения — 12345, VM хранится в файле `12345_vm`, в том же самом каталоге, что и основной файл отношения.
- 

Оптимизация SQL

План выполнения запроса

- Прежде, чем запустить выполнение запроса, СУБД выстраивает пайплан его выполнения
 - Посмотреть план выполнения запросов можно с помощью команды EXPLAIN
 - Визуализация плана выполнения запроса: <https://explain.dalibo.com/>
- 

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. (cost = number_1 .. number_2) – затраты (среднее время доступа до страницы в БД):
3. number_1 - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. number_2 - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= ...) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = ...) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. number_1 - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. number_2 - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= ...) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = ...) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= ...)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = ...)** – ожидаемый средний размер возвращаемых строк в байтах.
7. **Planning time** (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. **Execution time** (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= ...) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = ...) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= ...)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = ...) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= ...)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = ...)** – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= ...)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = ...)** – ожидаемый средний размер возвращаемых строк в байтах.
7. **Planning time** (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. **Execution time** (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= ...)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = ...)** – ожидаемый средний размер возвращаемых строк в байтах.
7. **Planning time (Время планирования)** - время, затраченное на построение плана запроса и его оптимизацию.
8. **Execution time (Время выполнения)** - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Индекс

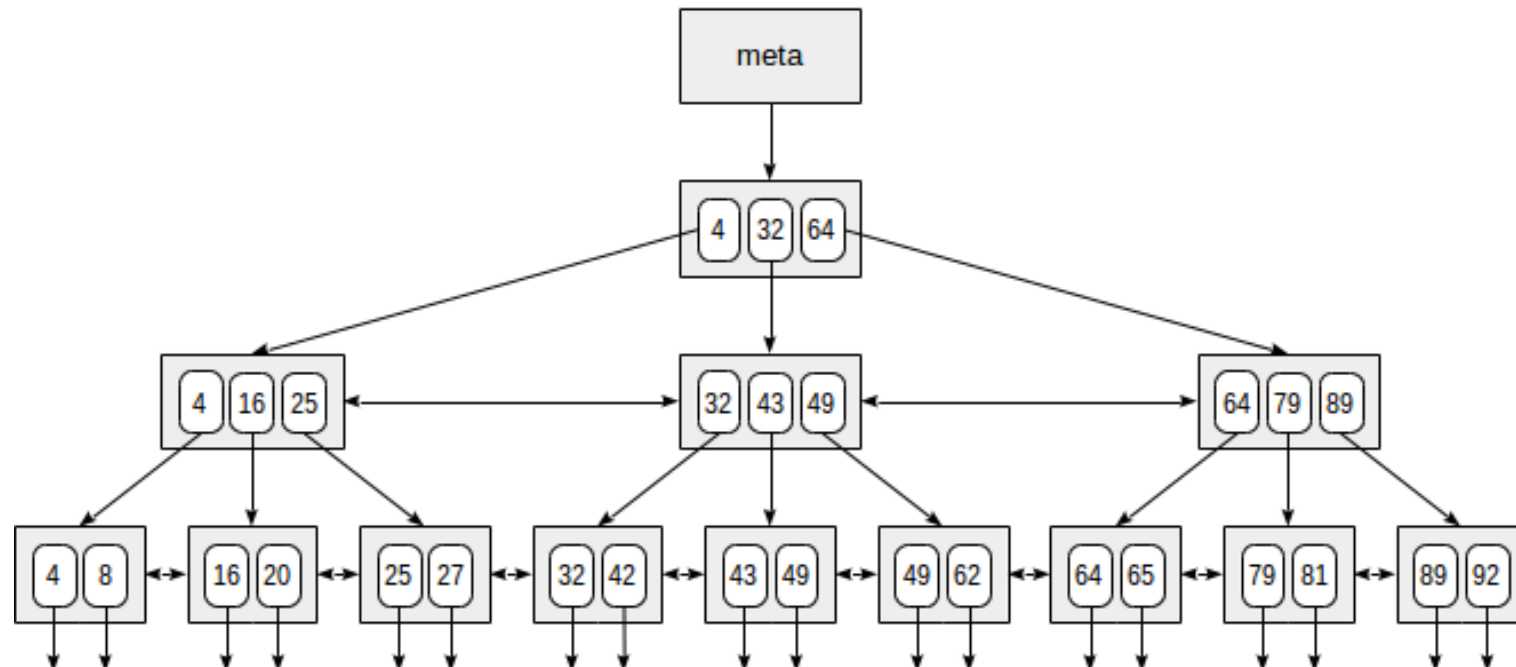
- **Индекс** - объект БД, предназначенный для ускорения доступа к данным
- **Создание индекса** - одно из основных средств оптимизации запросов
- Индекс, ускоряя доступ к данным, взамен требует затрат на своё содержание:
 - индекс занимает место на диске
 - любая операция над проиндексированными данными (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
- При любом способе сканирования (индексном, исключительно индексном, по битовой карте) метод доступа b-tree возвращает упорядоченные данные

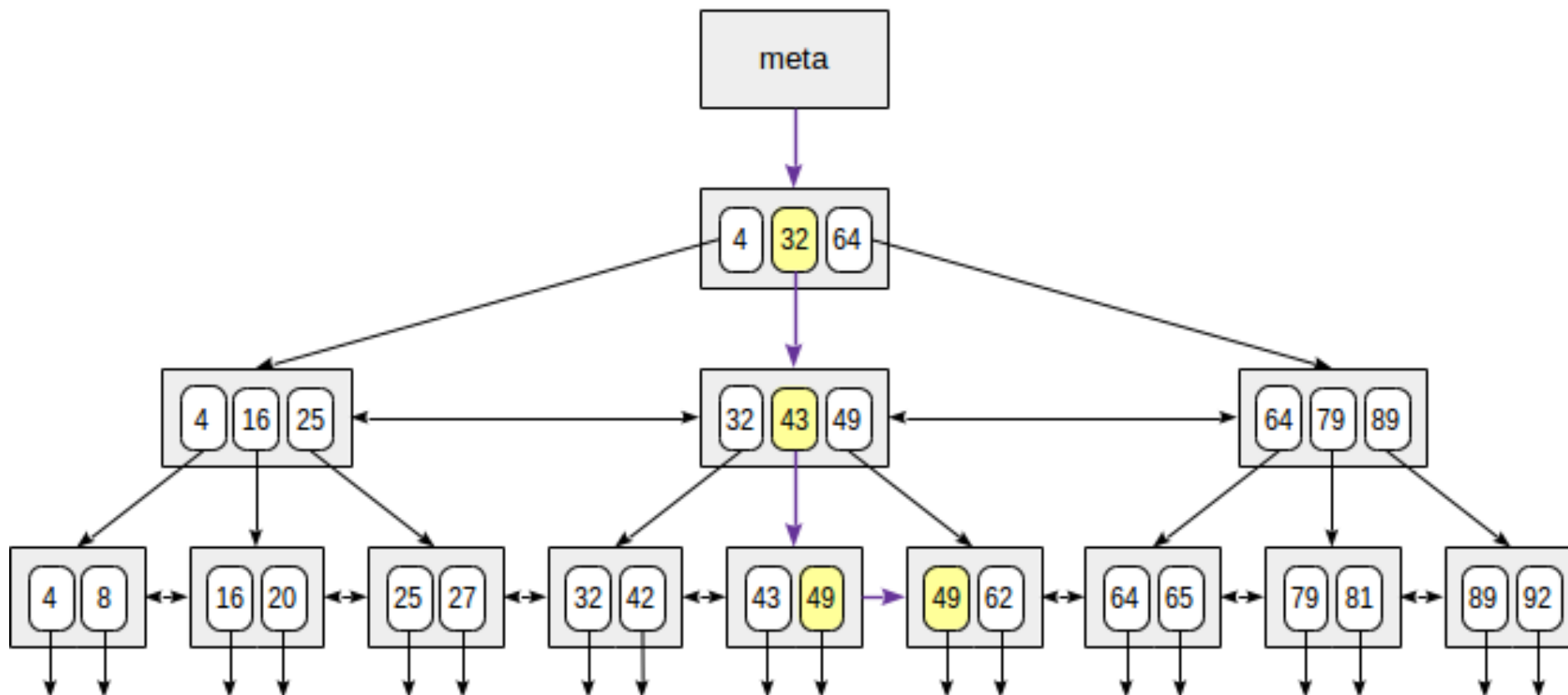
Пример индекса по одному полю с целочисленными ключами

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-ы).



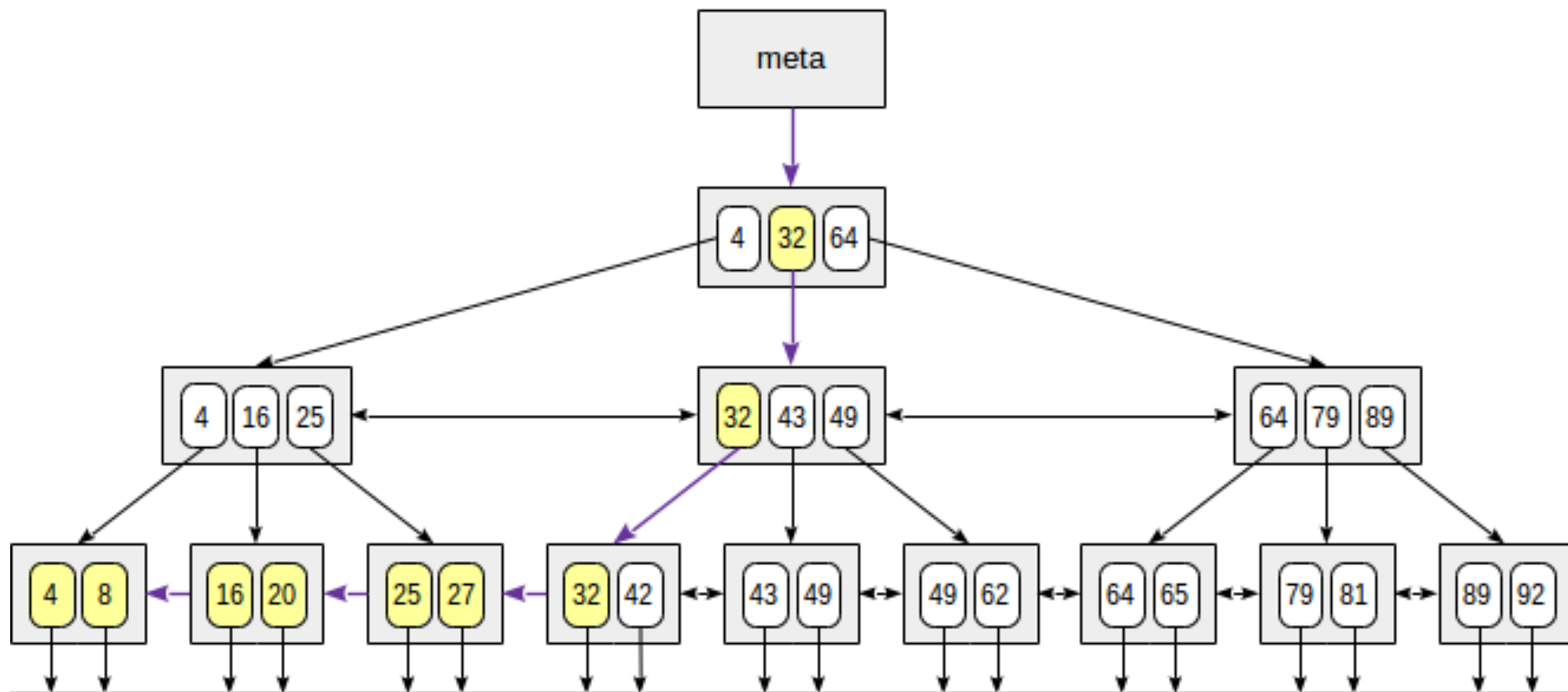
Поиск по равенству

- Индексированный атрибут = 49



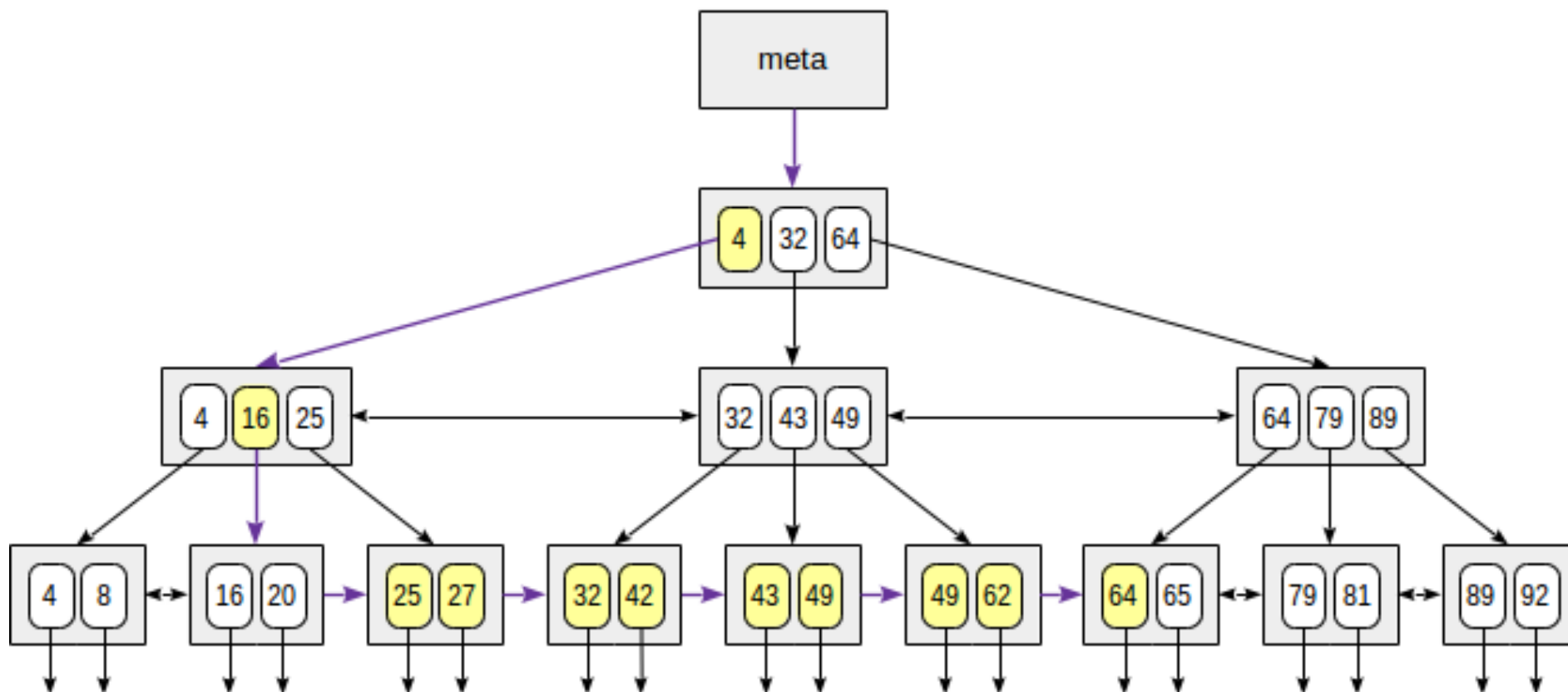
Поиск по неравенству

- Индексированный атрибут ≤ 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

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования.
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость и возвращает полученные данные.
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования.
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость и возвращает полученные данные.
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования.
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость и возвращает полученные данные.
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования.
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость и возвращает полученные данные.
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

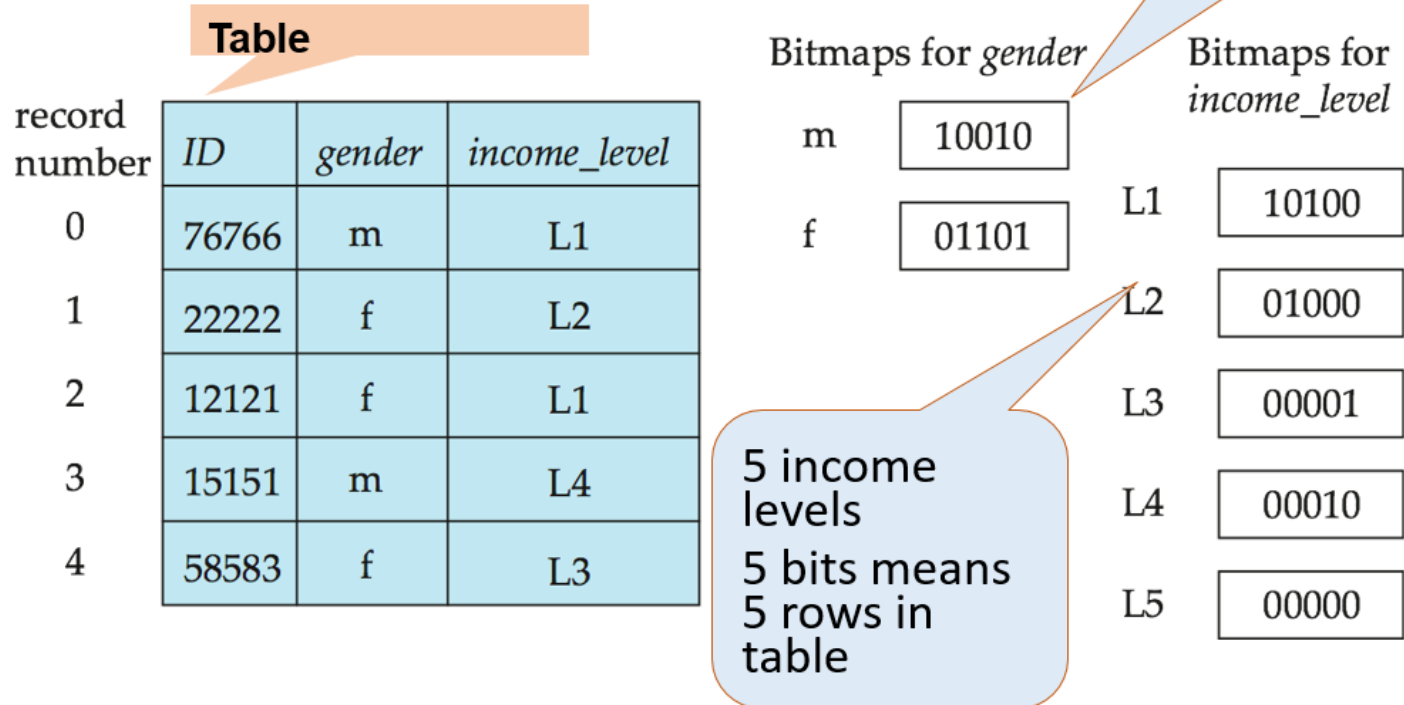
Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования.
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся. Механизм индексирования по очереди обращается к тем страницам таблицы, на которые указывают TID, получает версию строки, проверяет ее видимость и возвращает полученные данные.
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Битовая карта

Bitmap Indexes

Example: index on *gender & income_level*



Другие типы индексов

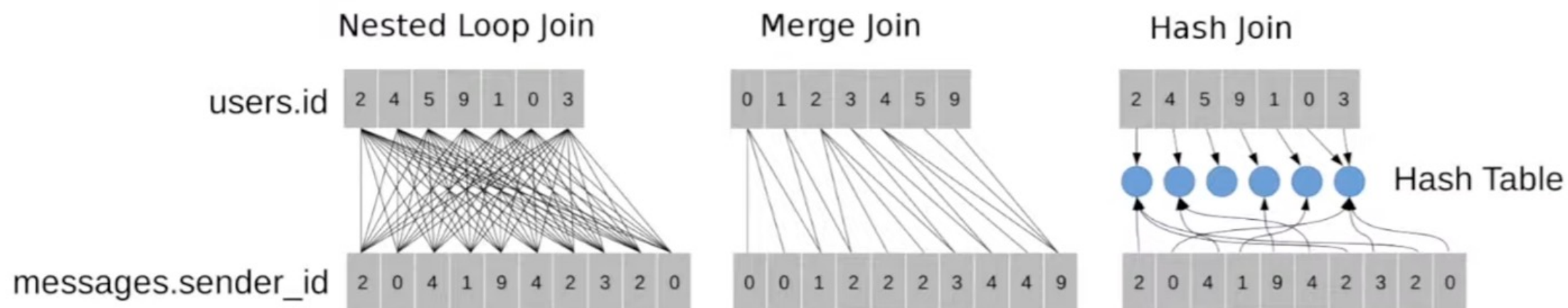
- Хэш-индекс
- GiST (Generalized Search Tree) – геоданные, текст, картинки
- SP-GiST (Space Partitioning Generalized Search Tree)
- GIN (Generalized Inverted Index) – полнотекстовый поиск
- RUN
- BRIN
- Bloom

Частичный индекс

- применяется, когда нет необходимости индексировать все значения атрибута (несбалансированные данные)
- `create index on table (column) where column_name = column_value;`

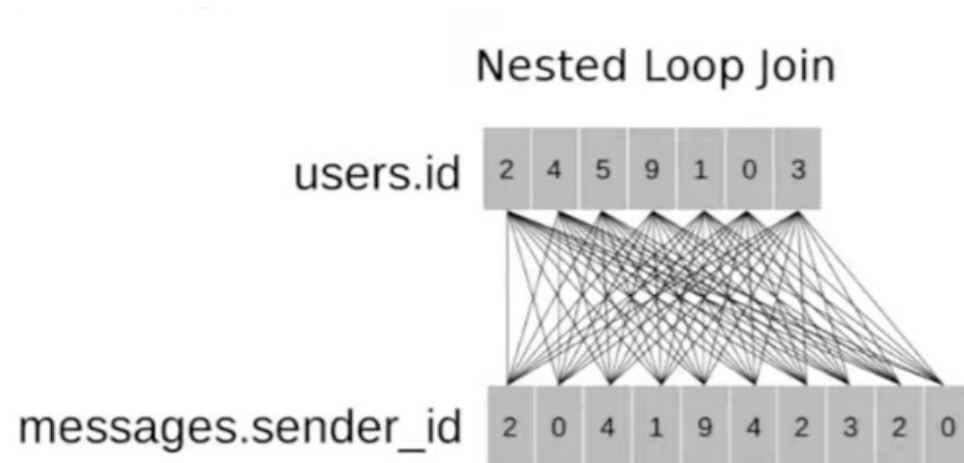
Виды операций объединения

- Nested Loop Join (соединение вложенных циклов);
- Merge Join (соединение слиянием);
- Hash Join (хэш-соединение).



Nested Loop Join

- сравнивает каждую строку первой таблицы с каждой строкой второй таблицы
- выводит строки, которые удовлетворяют условию объединения

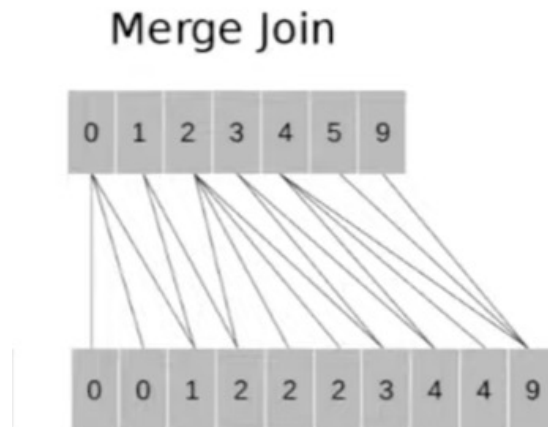


Псевдокод:

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

Merge Join

- используется, если объединяемые наборы данных отсортированы (или могут быть отсортированы с небольшими затратами) с помощью ключа join
- работает на индексах, которые возвращают отсортированные данные

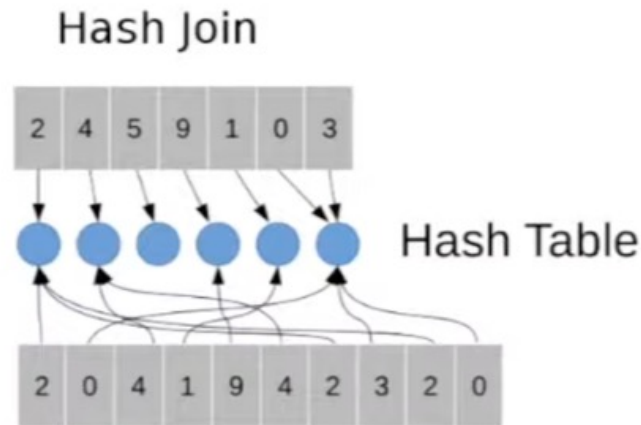


Псевдокод:

```
get first row R1 from input 1
get first row R2 from input 2
while not at the end of either input
  begin
    if R1 joins with R2
      begin
        get next row R2 from input 2
        return (R1, R2)
      end
    else if R1 < R2
      get next row R1 from input 1
    else
      get next row R2 from input 2
  end
end
```

Hash Join

- хэш-соединение строит хэш-таблицу на основе ключа соединения
- планировщик обычно решает выполнить хэш-соединение, когда таблицы имеют более или менее одинаковый размер, и хэш-таблица может разместиться целиком в памяти



Псевдокод:

```
for each row R1 in the build table
  begin
    calculate hash value on R1 join key(s)
    insert R1 into the appropriate hash bucket
  end
for each row R2 in the probe table
  begin
    calculate hash value on R2 join key(s)
    for each row R1 in the corresponding hash
      bucket
        if R1 joins with R2
          return (R1, R2)
        end
  end
```

UNION vs UNION ALL

UNION – объединяет два множества и удаляет дублирующиеся строки (выполняет операцию DISTINCT)


UNION ALL – объединяет два множества без дедубликации

A decorative graphic at the bottom of the slide consisting of two overlapping wavy lines. The bottom-most line is a dark blue, and the line above it is a lighter, medium blue. They create a layered, wave-like effect across the width of the slide.

Для оптимизации следует избегать повторения кода при объединениях

- использование CTE (Common Table Expression) - конструкция WITH;
- сохранение результатов подзапросов во временную таблицу (temporary table – существует только в рамках вашей сессии);

DELETE and TRUNCATE

- При необходимости удаления всех данных из таблицы быстрее будет использовать **TRUNCATE**
 - **DELETE** физически удаляет данные;
 - **TRUNCATE** физически в моменте не удаляет данные, а проставляет «флаг» удаления
- 

Unlogged Tables

- Данные при записи и обновлении не записываются в журнал логирования операций (wal-log)
- Существующую таблицу нельзя переформатировать в unlogged, необходимо указывать данный параметр при создании таблицы:

```
CREATE UNLOGGED TABLE table_name
```

