```
CREATE INDEX idx_order_date ON orders (order_date);
CLUSTER orders USING idx_order_date;
https://www.postgresql.org/docs/current/sql-cluster.html

https://www.postgresql.org/docs/9.1/pageinspect.html
SELECT * FROM bt_page_items('index_test_table_random_num_idx', 1);

https://postgrespro.ru/docs/postgrespro/9.5/runtime-config-resource#guc-
maintenance-work-mem

https://postgrespro.ru/docs/postgresql/11/runtime-config-resource#GUC-MAX-
WORKER-PROCESSES

SET max_parallel_workers_per_gather TO 4;

REINDEX [ VERBOSE ] TABLE имя_таблицы
REINDEX [ VERBOSE ] INDEX имя_индекса

SELECT
        relname AS table_name,
        indexrelname AS index_name,
        pg_relation_size(indexrelid) AS index_size_bytes,
        idx_scan AS index_scans,
        idx_tup_read AS tuples_read,
        idx_tup_fetch AS tuples_fetched
FROM
        pg_stat_user_indexes
JOIN
        pg_index USING (indexrelid)
JOIN
        pg_indexes ON pg_indexes.indexname =
pg_stat_user_indexes.indexrelname

CREATE EXTENSION pgstattuple;

SELECT * FROM pgstattuple('index_test_table_random_num_idx');
```

План

- Алгоритмы соединения
- Виды JOIN
- Виды объединений

Алгоритмы соединения

Два массива данных с id:
3 4 1 5 2 - массив 1 N
2 3 1 4 5 - массив 2 M

O(N*M)

1 2 3 4 5 - массив 1 N
1 2 3 4 5 - массив 2 M

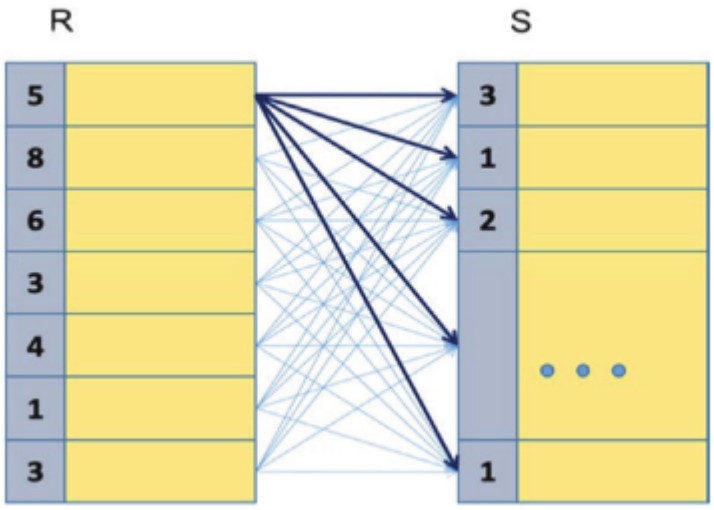
1. Пройтись циклом по обоим массив и объеденить числа O(N*M). 3 Таблицы на 1000 строк, с общим id -> 1000* 1000 * 1000.
2. Hash JOIN для объединения. Сложность алгоритма O(N + M + hashCalc(N)). **work_mem**
3. Merge JOIN для объединения. Сложность для несортированных O(log(N)*N + log(M)*M + N + M), для сортированных O(N + M)

Nested Loop

Nested Loop Join в PostgreSQL - это метод объединения двух таблиц, где для каждой строки одной таблицы производится поиск соответствующих строк в другой таблице. Этот метод особенно полезен, когда одна таблица значительно меньше другой и/или когда есть условия соединения, которые не поддерживаются индексами.

Как работает

1. PostgreSQL выбирает одну таблицу в качестве внешней (обычно меньшего размера) и другую в качестве внутренней.
2. Для каждой строки из внешней таблицы выполняется полный перебор строк внутренней таблицы.
3. Для каждой пары строк (одна из внешней таблицы, одна из внутренней) проверяется условие соединения.
4. Если условие соединения истинно, соответствующие строки объединяются и добавляются в результат.



Алгоритм объединения

Сложность: O(N*M)

```
function nestedLoopJoin(tableA, tableB, isMatchFunction) {
    const result = [];

    for (let rowA of tableA) {
        for (let rowB of tableB) {
            if (isMatchFunction(rowA, rowB)) {
                // Объединяем строки, если они соответствуют условиям
соединения
                result.push({ ...rowA, ...rowB });
            }
        }
    }

    return result;
}

// Пример функции, проверяющей условие соединения
function isMatch(rowA, rowB) {
    return rowA.someKey === rowB.someKey; // условие соединения
}
```

Недостатки Nested Loop Join

1. **Масштабируемость**: Он не масштабируется хорошо для больших таблиц, поскольку требует выполнения внутреннего цикла для каждой строки внешней таблицы, что приводит к большому количеству операций.
2. **Производительность**: Для таблиц без индексов, соответствующих условиям соединения, производительность может быть низкой.
3. **Ресурсоемкость**: Потребляет значительные вычислительные ресурсы, особенно с увеличением размера таблиц.

Оптимизация

1. **Использование индексов**: Создание индексов на столбцах, используемых в условиях соединения, может значительно ускорить процесс, т.к. PostgreSQL сможет быстрее находить соответствующие строки в внутренней таблице.
2. **Минимизация размера таблиц**: Предварительная фильтрация данных с помощью предложений `WHERE` может уменьшить размер обеих таблиц до выполнения соединения.
3. **Правильный выбор внешней и внутренней таблиц**: PostgreSQL обычно делает это автоматически, но иногда подсказки оптимизатора или изменение порядка таблиц в запросе могут повлиять на производительность.
4. **Параллелизм**: В некоторых случаях PostgreSQL может выполнить Nested Loop Join параллельно, распределяя работу между несколькими процессорами/ядрами.

Hash JOIN

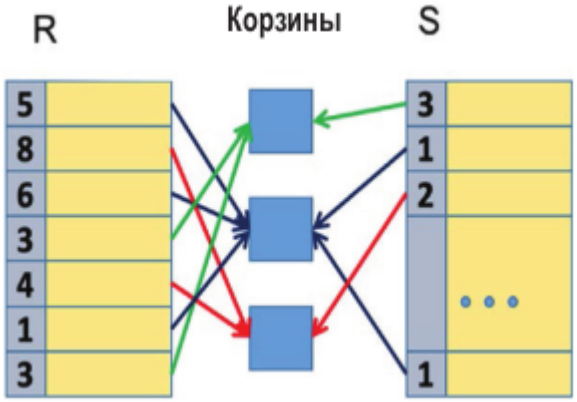
Hash Join в PostgreSQL — это метод объединения двух таблиц, который особенно эффективен для больших наборов данных с условиями равенства. Этот метод разделяется на две основные фазы: создание хеш-таблицы из одной таблицы (обычно меньшей) и затем сканирование другой таблицы для поиска соответствующих записей.

// hashFunction(args) =>
// hashFuncition(1, 2) = hashFunction(1, 2)

Как работает

1. **Фаза построения**: PostgreSQL выбирает одну из таблиц (называемую внутренней таблицей) и создает хеш-таблицу **в памяти** на основе столбца или столбцов, используемых в условии соединения. Каждая запись внутренней таблицы хешируется и помещается в хеш-таблицу.
2. **Фаза проба**: Затем система перебирает строки второй таблицы (внешней таблицы) и для каждой строки вычисляет хеш-значение по тем же ключам, что и для хеш-таблицы. Если в хеш-таблице находится соответствующая хеш-значению запись,

строки объединяются и добавляются в результат.



Алгоритм объединения

Сложность: O(N+M).

```
function hashJoin(tableA, tableB, keyA, keyB) {
    const hashTable = {};
    const result = [];

    // Фаза построения: строим хеш-таблицу из tableA
    for (let rowA of tableA) {
        const hashKey = rowA[keyA];
        if (!hashTable[hashKey]) {
            hashTable[hashKey] = [];
        }
        hashTable[hashKey].push(rowA);
    }

    // Фаза проба: сканируем tableB и ищем соответствия в хеш-таблице
    for (let rowB of tableB) {
        const hashKey = rowB[keyB];
        if (hashTable[hashKey]) {
            for (let rowA of hashTable[hashKey]) {
                result.push({ ...rowA, ...rowB });
            }
        }
    }

    return result;
```

```
}
```

Недостатки Hash Join

1. **Потребление памяти**: Создание хеш-таблицы требует дополнительной памяти, что может стать проблемой для очень больших таблиц.
2. **Ограничения условий соединения**: Hash Join эффективен только для условий равенства.
3. **Переполнение памяти**: Если хеш-таблица не умещается в доступной памяти, PostgreSQL должен использовать временные дисковые файлы, что снижает производительность.

Оптимизация

1. **Подходящие условия соединения**: Использовать Hash Join для условий равенства, где это наиболее эффективно.
2. **Размер рабочей памяти**: Увеличение параметра `work_mem` в конфигурации PostgreSQL позволяет хеш-таблице оставаться в памяти, улучшая производительность.
3. **Использование партиционирования**: Разбиение больших таблиц на более мелкие секции (партиции) может уменьшить размер хеш-таблицы и улучшить производительность.
4. **Планирование запросов**: Понимание и оптимизация плана запроса, возможно, с использованием подсказок оптимизатора, для обеспечения выбора Hash Join в тех случаях, когда он предпочтителен.

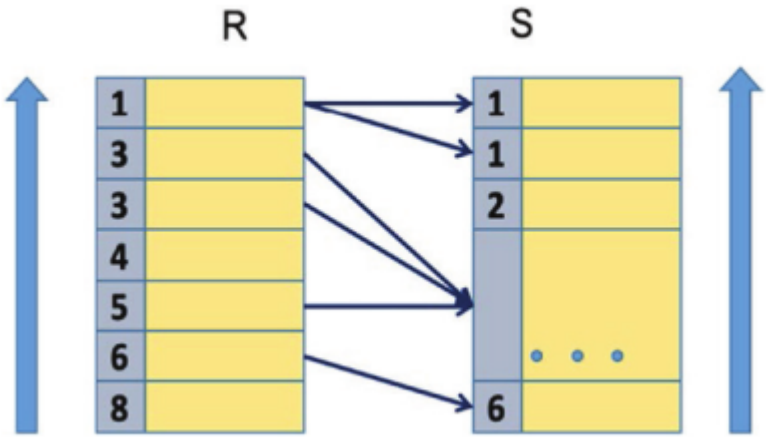
Merge JOIN

Merge Join в PostgreSQL — это метод объединения двух отсортированных наборов данных на основе условия равенства. Этот метод эффективен, когда обе таблицы уже отсортированы по ключу соединения или когда индексы, поддерживающие нужный порядок сортировки, доступны для обеих таблиц.

Как работает

1. **Подготовка**: PostgreSQL убеждается, что обе таблицы отсортированы по ключам соединения. Если таблицы не отсортированы, они сортируются перед выполнением соединения.
2. **Сканирование**: Система одновременно сканирует обе таблицы, начиная с наименьшего элемента.

3. **Соединение**: Когда ключ соединения текущей строки одной таблицы совпадает с ключом текущей строки другой таблицы, строки объединяются. Если ключ одной таблицы меньше ключа другой, указатель перемещается вперед в таблице с меньшим ключом, чтобы найти соответствие.
4. **Продолжение**: Процесс продолжается до тех пор, пока не будут полностью просмотрены обе таблицы.



Алгоритм объединения на JavaScript

Сложность: O(NlogN+MlogM+N+M). O(N + M).

```
function mergeJoin(tableA, tableB, keyA, keyB) {
    const result = [];

    // Сортировка таблиц по ключам соединения, если они уже не отсортированы
    tableA.sort((a, b) => a[keyA] - b[keyA]);
    tableB.sort((a, b) => a[keyB] - b[keyB]);

    let i = 0, j = 0;

    while (i < tableA.length && j < tableB.length) {
        if (tableA[i][keyA] < tableB[j][keyB]) {
            i++;
        } else if (tableA[i][keyA] > tableB[j][keyB]) {
            j++;
        } else {
            // Ключи равны, объединяем строки
            result.push({ ...tableA[i], ...tableB[j] });
            // Перемещаемся вперед в обеих таблицах
```

```
            i++;
            j++;
        }
    }

    return result;
}
```

Недостатки Merge Join

1. **Предварительная сортировка**: Если таблицы изначально не отсортированы по ключу соединения, требуется дополнительное время и ресурсы на их сортировку.
2. **Ограниченность условий**: Эффективен только для условий равенства.
3. **Использование памяти**: Предварительная сортировка и хранение отсортированных таблиц могут потребовать значительного количества памяти.

Оптимизация

1. **Индексы**: Использование индексов, которые поддерживают нужный порядок сортировки, может исключить необходимость в предварительной сортировке.
2. **Планирование запросов**: Понимание плана запроса и его оптимизация может помочь PostgreSQL выбрать Merge Join там, где это наиболее эффективно.
3. **Подбор размера `work_mem`**: Настройка параметра `work_mem` для оптимизации операций сортировки и минимизации использования дискового пространства для временных файлов.

Операции над множествами

В SQL существует несколько основных операций над множествами, которые позволяют комбинировать результаты нескольких запросов. Эти операции включают `UNION`, `INTERSECT`, и `EXCEPT`. Каждая из этих операций применяется к двум или более результатам запросов и возвращает таблицу в качестве результата.

1. `UNION`

Операция `UNION` используется для объединения результатов двух или более запросов в один набор результатов, исключая дубликаты. Все запросы, объединяемые операцией `UNION`, должны иметь одинаковое количество столбцов в результатах, совместимые типы данных и в том же порядке.

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

2. `UNION ALL`

`UNION ALL` похожа на `UNION`, но включает в себя все дубликаты. Это может быть полезно, когда необходимо сохранить все записи из каждого запроса, включая повторяющиеся.

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;`
```

3. `INTERSECT`

`INTERSECT` возвращает только те строки, которые присутствуют в результатах обоих запросов. То есть, это пересечение двух наборов данных.

```
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;
```

4. `EXCEPT`

Операция `EXCEPT` возвращает строки из первого запроса, которых нет в результате второго запроса. Это похоже на операцию вычитания одного набора данных из другого.

```
SELECT column_name(s) FROM table1
EXCEPT
SELECT column_name(s) FROM table2;
```

Правила использования

- Все операции над множествами требуют, чтобы количество и типы столбцов во всех объединяемых запросах были одинаковыми.
- Порядок столбцов должен быть одинаковым во всех запросах.
- Результаты операций над множествами могут быть отсортированы с использованием `ORDER BY`. Однако, `ORDER BY` применяется к всему объединенному результату и

должно быть указано в конце последнего запроса.

Важно помнить об алгоритмах.