

ЗАЩИЩЕННАЯ СИСТЕМА УПРАВЛЕНИЯ
БАЗАМИ ДАННЫХ «ЈАТОВА»

Руководство по настройке. Часть 31.
Корректировка запросов.
Компонент «pg_hint_plan»

643.72410666.00067-07 98 01-31

Листов 33

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

АННОТАЦИЯ

В документе приведены сведения, необходимые для установки и эксплуатации компонента «pg_hint_plan» (далее по тексту – «компонент»), предназначенного для выполнения корректировки запросов.

Настоящее руководство предназначено для администраторов СУБД.

Для СУБД «Jatoba» версии ядра 5 используется версия компонента — 1.5.

Для СУБД «Jatoba» версии ядра 6 используется версия компонента — 1.6.

Для СУБД «Jatoba» версии ядра 18 используется версия компонента — 1.8.

Степени важности примечаний, применяемые в документе:



Важная информация – указания, требующие особого внимания



Дополнительная информация – указания, позволяющие упростить работу с изделием

СОДЕРЖАНИЕ

1. Назначение компонента.....	4
1.1. Условия применения.....	4
1.2. Ограничения.....	4
2. Установка и настройка.....	5
3. Функциональные возможности компонента.....	6
3.1. Использование.....	6
3.2. Таблица указаний.....	6
3.3. Типы указаний.....	6
3.4. Указания для методов сканирования.....	7
3.5. Указания для методов соединения	7
3.6. Указание для порядка соединения.....	7
3.7. Указания для управления поведением соединения.....	7
3.8. Указание для корректировки числа строк	8
3.9. Указания для параллельных планов	8
3.10. Временное переопределение параметров GUC	8
3.11. Параметры GUC для настройки pg_hint_plan.....	8
3.12. Подробное описание указаний.....	9
3.12.1. Синтаксис и расположение	9
3.12.2. Использование с PL/pgSQL.....	9
3.12.3. Регистр букв в именах объектов	9
3.12.4. Экранирование спецсимволов в именах объектов	9
3.12.5. Различение нескольких вхождений таблицы.....	10
3.12.6. Нижележащие таблицы представлений или правил.....	10
3.12.7. Таблицы с наследованием	10
3.12.8. Указания с составными операторами	10
3.12.9. Выражения VALUES.....	10
3.12.10. Подзапросы	10
3.12.11. Использование указания IndexOnlyScan	11
3.12.12. Поведение указания NoIndexScan	11
3.12.13. Указание Parallel и UNION.....	11
3.12.14. Указание DisableIndex	11
3.12.15. Установка параметров pg_hint_plan в указаниях Set	12
3.12.16. Поддерживаемые указания.....	12
3.12.17. Примеры корректировки планов выполнения запросов.....	15
4. Ошибки	31
Перечень сокращений.....	32

1. НАЗНАЧЕНИЕ КОМПОНЕНТА

Компонент «pg_hint_plan» предназначен для корректировки планов выполнения, применяя так называемые «указания», записываемые в виде простых описаний в SQL-комментариях особого вида.

1.1. Условия применения

Компонент «pg_hint_plan» может использоваться с СУБД «Jatoba» версий 5.x и выше, под управлением операционной системы GNU/Linux.



В текущей реализации компонента не поддерживается управление через компонент пользовательского веб-интерфейса для администраторов «Jatoba data safe».

Ограничений по совместимости с другими компонентами нет.

1.2. Ограничения

Существуют следующие функциональные ограничения планировщика:

- если в предложении FROM больше чем from_collapse_limit элементов, компонент не может повлиять на порядок соединения;
- если принудительно выбранный план нельзя выполнить, компонент выберет любой исполнимый;
- нельзя передать запрос в ECPG;
- одинаковые запросы с разными указаниями будут консолидироваться как один и тот же запрос.

2. УСТАНОВКА И НАСТРОЙКА

Установка компонента должна производиться от имени пользователя, обладающего административными привилегиями в системе. Данный компонент штатным образом может быть установлен только с СУБД «Jatoba» (см. документ «Защищенная система управления базами данных «Jatoba». Руководство по установке).

Для активации компонента в файле postgresql.conf прописать следующую строку:

```
shared_preload_libraries = 'pg_hint_plan'
```

Для автоматической загрузки определенных сеансов необходимо использовать:

```
ALTER USER SET/ALTER DATABASE SET
```

Компонент не требует выполнения CREATE EXTENSION, но для использования таблицы указаний необходимо создать расширение и включить параметр:

```
CREATE EXTENSION pg_hint_plan;  
SET pg_hint_plan.enable_hint_table TO on;
```

3. ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ КОМПОНЕНТА

3.1. Использование

Компонент считывает указания в комментариях особого вида, заданных оператором SQL. Эта особая запись начинается с последовательности символов /*+ и заканчивается последовательностью */. Фразы указаний состоят из имени указания и последующих параметров, которые заключаются в скобки и разделяются пробелами. Такие указания могут размещаться в нескольких строках для улучшения читаемости.

3.2. Таблица указаний

Для удобства использования указания вносятся в специальную таблицу `hint_plan.hints`.

Таблица 3.1 – Содержание таблицы `hint_plan.hints`

Столбец	Описание
id	Уникальный номер строки с указанием. Этот столбец заполняется автоматически генератором последовательности.
norm_query_string	Шаблон для выбора запросов, к которым будет относиться указание. Константы, фигурирующие в целевом запросе, должны заменяться знаками ?. Пробельные символы в шаблоне являются значимыми.
application_name	Значение переменной, выбирающее сеансы, в которых будет действовать указание. С пустой строкой будут выбираться сеансы с любым значением <i>application_name</i> .
hint	Фраза указания. Это поле должно содержать указания без обрамляющей разметки комментариев.

Таблица указаний принадлежит пользователю, создавшему расширение, и для нее назначаются права доступа, установленные по умолчанию в момент `CREATE EXTENSION`. Указания, заданные в таблице, имеют больший приоритет, чем указания в комментариях.

3.3. Типы указаний

Фразы указаний подразделяются на шесть типов по видам объектов, на которые они могут воздействовать, и видам воздействия: методы сканирования, методы соединения, порядок соединения, корректировка количества строк, параллельные запросы и параметры GUC.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

3.4. Указания для методов сканирования

Указания для методов сканирования принудительно устанавливают метод сканирования для заданной таблицы. В качестве имени целевой таблицы компонент может распознать и ее псевдоним, если он определен.

Такие указания применимы к обычным таблицам, таблицам с наследованием, нежурналируемым таблицам, временным таблицам и системным каталогам. Они не применяются к внешним (сторонним) таблицам, табличным функциям, результатам VALUES, CTE, представлениям и вложенным запросам.

3.5. Указания для методов соединения

Указания для методов соединения принудительно выбирают определенный метод для соединения заданных таблиц.

Эти указания могут работать с обычными таблицами, таблицами с наследованием, нежурналируемыми и временными таблицами, внешними (сторонними) таблицами, системными каталогами, табличными функциями, результатами команд VALUES и CTE. Однако на представления и подзапросы они не воздействуют.

Указания для методов соединения рекомендуется использовать вместе с указанием для порядка соединения Leading, так как оно гарантирует применение порядка, указанного в запросе.

3.6. Указание для порядка соединения

Указание Leading устанавливает порядок соединения двух и более таблиц. Выбрать порядок можно двумя способами:

- установить определенный порядок соединения, но не ограничивать направление на каждом уровне;
- ограничить направление соединения.

Указание Leading не может функционировать с модулем GEQO, если число указанных в запросе таблиц превышает `geqo_threshold`.

3.7. Указания для управления поведением соединения

Указание «Memoize» позволяет соединению запоминать внутренний результат, а указание «NoMemoize» запрещает это.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

3.8. Указание для корректировки числа строк

Указание Rows корректирует неверную оценку количества строк при соединениях, возможно вызванную ограничениями планировщика.

3.9. Указания для параллельных планов

Указание Parallel устанавливает конфигурацию параллельного выполнения при сканировании. Третий параметр определяет режим изменений конфигурации:

- «soft» pg_hint_plan меняет только max_parallel_workers_per_gather;
- «hard» меняются и другие параметры планировщика, чтобы принудительно установить количество параллельных исполнителей.

Это указание может воздействовать на обычные таблицы, родительские таблицы в иерархии наследования, нежурналируемые таблицы и системные каталоги. На внешние таблицы, табличные функции, предложения VALUES, CTE, представления и вложенные запросы оно не действует. Обращаться в данном указании к внутренней таблице в представлении можно по имени или псевдониму этой таблицы.

3.10. Временное переопределение параметров GUC

Указание Set меняет параметры GUC только на время планирования. Желаемое влияние на планирование могут оказывать параметры GUC, если только какое-либо другое указание не конфликтует с заданными параметрами метода планирования. Если для одного параметра GUC задано несколько указаний, в силу вступает последнее.

3.11. Параметры GUC для настройки pg_hint_plan

На поведение pg_hint_plan влияют следующие описанные ниже параметры GUC.

Таблица 3.2 – Параметры GUC

Имя параметра	Описание	Значение по умолчанию
pg_hint_plan.enable_hint	Значение True включает pg_hint_plan	on (вкл.)
pg_hint_plan.enable_hint_table	Значение True включает использование указаний из таблицы	on (вкл.)
pg_hint_plan.parse_messages	Задаёт уровень, с которым будут попадать в журнал ошибки разбора указаний. Допустимые значения: error, warning, notice, info, log, debug	INFO

Имя параметра	Описание	Значение по умолчанию
pg_hint_plan.debug_print	Управляет выводом и детализацией отладочной информации. Допустимые значения: off, on, detailed и verbose	off (выкл.)
pg_hint_plan.message_level	Задаёт уровень, с которым будут попадать в журнал отладочные сообщения. Допустимые значения: error, warning, notice, info, log, debug	LOG

3.12. Подробное описание указаний

3.12.1. Синтаксис и расположение

Обработчик pg_hint_plan считывает указания только из первого блочного комментария и немедленно прекращает разбор, обнаруживая недопустимый символ. Допустимыми символами являются буквы, цифры, пробелы, подчеркивания, запятые и скобки.

3.12.2. Использование с PL/pgSQL

Компонент может работать с запросами в скриптах PL/pgSQL с некоторыми ограничениями.

- Указания воздействуют только на следующие типы запросов:
 - Запросы, возвращающие одну строку (SELECT, INSERT, UPDATE и DELETE);
 - Запросы, возвращающие множество строк (RETURN QUERY);
 - Динамические операторы SQL (EXECUTE);
 - Запрос, открывающий курсор (OPEN);
 - Цикл по результату запроса (FOR);
- Комментарий с указанием должен добавляться после первого слова запроса, так как комментарии, идущие перед ним, в составе запроса не передаются.

3.12.3. Регистр букв в именах объектов

В отличие от стандартных функциональных возможностей СУБД, pg_hint_plan, разбирая имена объектов в указаниях, сравнивает их с внутренними именами объектов с учетом регистра.

3.12.4. Экранирование спецсимволов в именах объектов

Если имя объекта включает в себя скобки, кавычки или пробелы, оно должно заключаться в кавычки. При этом действуют те же правила экранирования, что и в СУБД.

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

3.12.5. Различение нескольких вхождений таблицы

Компонент может выбирать целевые объекты по псевдонимам, если они заданы. Это позволяет обратиться к одному определенному вхождению таблицы, используемой в запросе неоднократно.

3.12.6. Нижележащие таблицы представлений или правил

Указания не применяются непосредственно к представлениям, но могут воздействовать на выполняемые запросы, если имена объектов в указаниях совпадают с именами объектов в развернутом запросе представления. К таблицам внутри представления можно обращаться снаружи по назначенным им псевдонимам.

Необходимо использовать уникальные псевдонимы во избежание неправильной обработки.

3.12.7. Таблицы с наследованием

Указания могут быть нацелены только на родителя в иерархии наследования, однако при этом они воздействуют на всю иерархию. Указания, нацеленные на потомков в этой иерархии, не будут действовать.

3.12.8. Указания с составными операторами

Для одного описания составного оператора может задаваться только один комментарий, и записанные в нем указания будут распространяться на все отдельные операторы внутри этого составного.

3.12.9. Выражения VALUES

Все выражения VALUES в предложении FROM имеют внутреннее обозначение *VALUES*, так что к ним можно обращаться, только если в запросе фигурирует только одно выражение VALUES. Два или более выражений VALUES в запросе невозможно отличить, посмотрев на результат EXPLAIN, что приводит к неоднозначным результатам.

3.12.10. Подзапросы

В указаниях можно обращаться к подзапросам по имени ANY_subquery.

Для этих синтаксисов планировщик внутри дает имя подзапросу, планируя соединения таблиц с этим подзапросом, так что в указаниях соединений можно обращаться к нему по этому неявному имени.

3.12.11. Использование указания `IndexOnlyScan`

Сканирование индекса может вопреки ожиданиям выполняться с другим индексом, когда индекс, заданный в указании `IndexOnlyScan`, оказывается неподходящим для сканирования только по индексу.

3.12.12. Поведение указания `NoIndexScan`

Указание `NoIndexScan` подразумевает `NoIndexOnlyScan`.

3.12.13. Указание `Parallel` и `UNION`

Предложение `UNION` может выполняться в параллельном режиме, только когда все нижележащие подзапросы безопасны для распараллеливания. С другой стороны, если параллельное выполнение принудительно выбирается для любого из подзапросов, все предложение `UNION` будет обрабатываться параллельно, если это возможно. При этом в случае выбора в указании `Parallel` нулевого количества исполнителей выполнение в параллельном режиме будет запрещено.

3.12.14. Указание `DisableIndex`

`DisableIndex` – указание (hint) в расширении `pg_hint_plan`, которая исключает указанные индексы из рассмотрения при планировании запросов.

Указание `DisableIndex` обрабатывается на ранней стадии планирования с помощью функции `get_relation_info_hook`. Указанные индексы удаляются ещё до того, как они станут видны планировщику.

Особенности:

- Подсказка всегда обрабатывается первым, независимо от позиции в списке подсказок;
- Отключённый индекс не используется, даже если явно запрошен подсказкой `IndexScan`.

Указание `DisableIndex` можно указать в комментариях особого вида, записанных в теле целевого оператора SQL.

Например:

```
=# /*+DisableIndex(t t_c1) IndexScan(t t_c1) */  
EXPLAIN SELECT * FROM t WHERE c1 = 1;
```

```
LOG:  indexes disabled for DisableIndex(t): t_c1
LOG:  available indexes for IndexScan(t):
LOG:  pg_hint_plan:
used hint:
DisableIndex(t t_c1)
not used hint:
IndexScan(t t_c1)
duplication hint:
error hint:
```

QUERY PLAN

```
-----
Index Scan using t_pkey on t  (cost=0.15..8.17 rows=1
width=12)
    Index Cond: (c1 = 1)
(2 rows)
```

3.12.15. Установка параметров pg_hint_plan в указаниях Set

Параметры pg_hint_plan меняют поведение самого обработчика указаний, поэтому некоторые параметры работают не так, как можно ожидать:

— Указания, изменяющие enable_hint и enable_hint_table, игнорируются несмотря на то, что в отладочном выводе они отмечаются как «использованные указания»;

— Изменение debug_print и message_level начинает действовать с середины процедуры обработки целевого запроса.

3.12.16. Поддерживаемые указания

Ниже перечислены все поддерживаемые указания.

Таблица 3.3 – Список указаний

Группа	Формат	Описание
Метод сканирования	SeqScan(таблица)	Принудительно выбирает последовательное сканирование таблицы
	TidScan(таблица)	Принудительно выбирает сканирование таблицы по TID

Группа	Формат	Описание
	IndexScan(таблица [индекс...])	Принудительно выбирает сканирование таблицы по индексу (при добавлении индексов сканирование ограничивается ими)
	IndexOnlyScan(таблица [индекс...])	Принудительно выбирает сканирование таблицы только по индексу (при добавлении индексов сканирование ограничивается ими). Если сканирование только по индексу невозможно, может использоваться обычное сканирование по индексу
	BitmapScan(таблица[индекс...])	Принудительно выбирает сканирование таблицы по битовой карте (при добавлении индексов сканирование ограничивается ими)
	IndexScanRegexp(таблица [регулярное выражение POSIX...])	Принудительно выбирает сканирование таблицы по индексу. Сканирование ограничивается индексами с именами, соответствующими указанному регулярному выражению POSIX
	IndexOnlyScanRegexp(таблица [регулярное выражение POSIX...])	Принудительно выбирает сканирование таблицы только по индексу. Сканирование ограничивается индексами с именами, соответствующими указанному регулярному выражению POSIX
	BitmapScanRegexp(таблица[регулярное выражение POSIX...])	Принудительно выбирает сканирование таблицы по битовой карте. Сканирование ограничивается индексами с именами, соответствующими указанному регулярному выражению POSIX
	NoSeqScan(таблица)	Отключает выбор последовательного сканирование таблицы
	NoTidScan(таблица)	Отключает выбор сканирования таблицы по TID
	NoIndexScan(таблица)	Отключает выбор сканирования по индексу и сканирования только по индексу для заданной таблицы
	NoIndexOnlyScan(таблица)	Принудительно отключает выбор сканирования только по индексу для заданной таблицы
	NoBitmapScan(таблица)	Отключает выбор сканирования по битовой карте для таблицы
Метод соединения	NestLoop(таблица таблица[таблица...])	Принудительно выбирает вложенный цикл для соединений с заданными таблицами
	HashJoin(таблица таблица[таблица...])	Принудительно выбирает соединение по хешу для соединений с заданными таблицами

Группа	Формат	Описание
	MergeJoin(таблица таблица[таблица...])	Принудительно выбирает соединение слиянием для соединений с заданными таблицами
	NoNestLoop(таблица таблица[таблица...])	Отключает выбор вложенного цикла для соединений с заданными таблицами
	NoHashJoin(таблица таблица[таблица...])	Отключает выбор соединения по хешу для соединений с заданными таблицами
	NoMergeJoin(таблица таблица[таблица...])	Отключает выбор соединения слиянием для соединений с заданными таблицами
Порядок соединения	Leading(таблица таблица[таблица...])	Принудительно выбирает заданный порядок соединения
	Leading(<соединяемая пара>)	Принудительно выбирает заданный порядок и направления соединения. Соединяемая пара в данном случае — это пара таблица и/или других соединяемых пар, заключенная в скобки, что позволяет образовывать вложенные структуры
Управление поведением соединения	Memoize(таблица таблица[таблица...])	Позволяет самому верхнему соединению среди соединений, включающих указанные таблицы, запоминать внутренний результат. Обратите внимание, запоминание при этом не будет задействовано принудительно
	NoMemoize(таблица таблица[таблица...])	Запрещает самому верхнему соединению среди соединений, включающих указанные таблицы, запоминать внутренний результат
Корректировка числа строк	Rows(таблица таблица[таблица...] корректировка)	Корректирует число строк, получаемых в результате соединения указанных таблиц. Для корректировки можно задать абсолютное значение (#<n>) или использовать сложение (+<n>), вычитание (-<n>) и умножение (*<n>). Здесь <n> — это строка, которую сможет воспринять функция strtod()
Настройка параллельных запросов	Parallel(таблица<число исполнителей> [soft hard])	Принудительно включает или отключает параллельную обработку заданной таблицы. Параметр <число исполнителей> в этом указании определяет желаемое количество параллельных исполнителей (значение 0 отключает параллельное выполнение). Если третий параметр равен soft (по умолчанию), меняется только значение параметра сервера max_parallel_workers_per_gather, а в остальном планировщику остается свобода выбора. Со значением hard заданное количество исполнителей устанавливается принудительно

Группа	Формат	Описание
GUC	Set(параметр-GUC значение)	Устанавливает значение для параметра GUC на время планирования запроса

3.12.17. Примеры корректировки планов выполнения запросов

Потребность в корректировке планов выполнения запросов чаще всего возникает при снижении производительности, когда СУБД применяет неэффективные пути выполнения, требующие ручной оптимизации.

Для анализа конкретного запроса применяются команды EXPLAIN или EXPLAIN ANALYZE. Кроме того, для сбора статистики на рабочей базе данных можно использовать расширения, такие как pg_store_plans и pg_stat_statements, которые позволяют изучать историю планов запросов и время их выполнения на работающей системе.

Типичные проблемы связаны с неоптимальным использованием индексов оптимизатором или необходимостью ускоренного получения первых строк результата. При работе с индексами разработчик экспериментальным путём определяет оптимальную последовательность соединения таблиц и выбирает наиболее эффективные индексы. Для быстрого возврата начальных строк запроса рекомендуется избегать методов соединения Hash join и Merge join в пользу Nested Loop.

Эффективность внесённых изменений проверяется путем замера времени исполнения. Также обязательно выполняется проверка на соответствие результатов выполнения оптимизированного запроса исходной версии, выдаваемые данные должны совпадать с точностью до перестановки строк.

3.12.17.1 Принудительное использование индекса

В данном подразделе рассматриваются методы принудительного управления выбором индексов планировщиком СУБД с использованием указаний (hints).

Пример 1.

Создадим небольшую таблицу с индексом:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
CREATE TABLE small_t (id int, val text );  
INSERT INTO small_t  
SELECT i, 'v' || i  
FROM generate_series(1, 100) i;  
  
CREATE INDEX ON small_t(id);
```

При таком объеме данных индекс еще не используется:

```
EXPLAIN SELECT * FROM small_t WHERE id = 42;  
QUERY PLAN
```

```
-----  
Seq Scan on small_t (cost=0.00..2.25 rows=1 width=7)  
  Filter: (id = 42)  
(2 rows)
```

Однако с помощью указания `/*+ IndexScan() */` можно явно определить использование индексного сканирования, что меняет план выполнения (Seq Scan изменился на Index Scan):

```
EXPLAIN /*+ IndexOnlyScan(small_t) */  
SELECT * FROM small_t WHERE id = 42;  
QUERY PLAN
```

```
-----  
Index Scan using small_t_id_idx on small_t (cost=0.14..8.16  
rows=1 width=7)  
  Index Cond: (id = 42)  
(2 rows)
```

Пример 2.

Во втором примере для таблиц создаются как первичные ключи, так и дополнительные индексы на те же столбцы.

Создадим две таблицы с первичными ключами и дополнительными индексами:


```
CREATE TABLE a(a int PRIMARY KEY);  
CREATE INDEX a_key2 ON a(a);  
CREATE TABLE b(b int PRIMARY KEY);  
CREATE INDEX b_key2 ON b(b);
```

По умолчанию планировщик запросов может выбрать нежелательный индекс (например, **a_key2**):

```
EXPLAIN /*+ IndexScan(a)*/ SELECT * FROM a JOIN b ON a.a=b.b;  
QUERY PLAN  
-----  
Nested Loop  (cost=0.28..12.32 rows=1 width=8)  
  -> Index Scan using a_key2 on a (cost=0.12..4.14 rows=1  
width=4)  
    -> Index Only Scan using b_key2 on b (cost=0.15..8.17  
rows=1 width=4)  
        Index Cond: (b = a.a)
```

Указание `/*+ IndexScan(a a_pkey) */` позволяет не только определить тип сканирования, но и явно задать конкретный индекс (**a_pkey**), который должен быть использован:

```
EXPLAIN /*+ IndexScan(a a_pkey) IndexScan(b b_pkey)*/ SELECT *  
FROM a JOIN b ON a.a=b.b;  
QUERY PLAN  
-----  
Nested Loop  (cost=0.28..12.32 rows=1 width=8)  
  -> Index Scan using a_pkey on a (cost=0.12..4.14 rows=1  
width=4)  
    -> Index Scan using b_pkey on b (cost=0.15..8.17 rows=1  
width=4)  
        Index Cond: (b = a.a)
```

Если таблица подвержена частым изменениям (INSERT, UPDATE, DELETE), то имеет смысл использовать тип сканирования `IndexOnlyScan()`, так как этот тип сканирования учитывает Visibility Map для блоков данных.

```
EXPLAIN /*+ IndexOnlyScan(a a_pkey) IndexOnlyScan(b b_pkey) */  
SELECT * FROM a JOIN b ON a.a=b.b;
```

QUERY PLAN

```
-----  
Nested Loop  (cost=0.31..566.25 rows=2550 width=8)  
  -> Index Only Scan using a_pkey on a (cost=0.15..82.41  
rows=2550 width=4)  
    -> Index Only Scan using b_pkey on b (cost=0.15..0.19  
rows=1 width=4)  
      Index Cond: (b = a.a)
```

В зависимости от количества получаемых записей и наличия нескольких индексов иногда более эффективным оказывается сканирование BitmapScan(), который может комбинировать данные из нескольких индексов.

```
EXPLAIN /*+ BitmapScan(b b_pkey) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Nested Loop  (cost=0.16..10682.38 rows=2550 width=8)  
  -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    Disabled: true  
  -> Bitmap Heap Scan on b (cost=0.16..4.18 rows=1 width=4)  
    Recheck Cond: (a.a = b)  
      -> Bitmap Index Scan on b_pkey (cost=0.00..0.16  
rows=1 width=0)  
        Index Cond: (b = a.a)
```

Пример 3.

Тех же результатов, что и в примере 2 можно добиться с помощью указаний IndexScanRegexp() IndexOnlyScanRegexp() BitmapScanRegexp(). Они удобны, если на таблицах присутствует множество индексов и можно выбирать имена индексов с помощью регулярных выражений, если ввести некую договоренность об именовании индексов.

Например, можно создать индексы с разными видами сортировок, и в зависимости от сортировки в запросе (order by asc, order by desc) использовать индекс с соответствующей сортировкой (asc, desc) указав название по регулярному выражению. Или можно создать множество индексов содержащих условия. Или индексы разного типа на одно и тоже поле (hash, gin, gist, rum).

В качестве примера будет рассмотрена следующая конфигурация таблицы и индексов:

```
CREATE TABLE t(id int PRIMARY KEY, category int);
CREATE INDEX t_pkey_desc ON t(id desc);
CREATE INDEX t_pkey_hash ON t USING hash (id);
CREATE INDEX t_cat_1 ON t(id) WHERE category=1;
CREATE INDEX t_cat_2 ON t(id) WHERE category=2;
CREATE INDEX t_cat_3 ON t(id) WHERE category=3;
CREATE INDEX t_cat_4 ON t(id) WHERE category=4;
```

По умолчанию используется hash индекс:

```
EXPLAIN  SELECT * FROM t WHERE id=1 AND category=2;

          QUERY PLAN

-----
Index Scan using t_pkey_hash on t (cost=0.00..8.02 rows=1
width=8)

   Index Cond: (id = 1)
   Filter: (category = 2)
```

Но можно сделать выбор среди индексов, которые содержат символы cat, что соответствует договоренности отмечать некоторые индексы таким сочетанием символов:

```
EXPLAIN /*+ IndexScanRegexp(t .*cat.*) */ SELECT * FROM t
WHERE id=1 AND category=2;

                        QUERY PLAN
-----
Index Scan using t_cat_2 on t  (cost=0.12..8.14 rows=1 width=8)
  Index Cond: (id = 1)
```

Или модифицировать сканирование на BitmapScan, и тогда среди того же набора индексов будет определен наиболее подходящий с точки зрения планировщика, но уже с другим типом сканирования.

```
EXPLAIN /*+ BitmapScanRegexp(t .*cat.*) */ SELECT * FROM t
WHERE id=1 AND category=2;

                        QUERY PLAN
-----
Bitmap Heap Scan on t  (cost=4.13..8.15 rows=1 width=8)
  Recheck Cond: ((id = 1) AND (category = 2))
  -> Bitmap Index Scan on t_cat_2 (cost=0.00..4.13 rows=1
width=0)
    Index Cond: (id = 1)
```

Аналогична ситуация с указанием IndexOnlyScanRegexp().

Данные примеры демонстрируют, как можно контролировать план запроса, предлагая планировщику СУБД использовать строго определённые индексы вместо выбранных по умолчанию. Такая необходимость может возникнуть, если какой-нибудь индекс помещен на более медленное хранилище и необходимо подключить индекс с более быстрого хранилища или, когда данные типа GUID плохо индексируются b-tree индексом, но планировщик об этом не догадался.

3.12.17.2 TID сканирование

TID (tuple id) содержит в себе номер блока и номер записи в этом блоке. TID позволяет физически адресовать запись на диске.

В каждой таблице содержится псевдоколонка CTID, в которой содержится уникальный физический адрес каждой записи. Доступ к записи по её TID является наиболее быстрым, так как указывает на конкретное место на диске.

Так как TID может измениться, то его имеет смысл использовать, если таблица испытывает мало изменений или вовремя обновлены значения TID-ов, если они хранятся отдельно.

Если имеется дополнительный индекс, а условие по CTID неочевидно для планировщика, то он может начать использовать индекс, а не желаемый TID Scan.

В качестве примера создадим таблицу с индексом:

```
CREATE TABLE t (id int, v text);
INSERT INTO t SELECT i, 'val' FROM generate_series(1, 100000)
i;
CREATE INDEX t_id_idx ON t (id);
```

Планировщик использует индекс вместо желаемого TID:

```
EXPLAIN SELECT * FROM t WHERE ctid >= '(10, 0)::tid and id=1;
      QUERY PLAN
-----
Index Scan using t_id_idx on t (cost=0.29..8.31 rows=1
width=8)
    Index Cond: (id = 1)
    Filter: (ctid >= '(10,0)::tid)
```

Но можно принудительно включить TID Scan используя указание TidScan():

```
EXPLAIN /*+ TidScan(t)*/ SELECT * FROM t WHERE ctid >= '(10,
0)::tid and id=1;
      QUERY PLAN
-----
Tid Range Scan on t (cost=0.00..1657.77 rows=1 width=8)
    TID Cond: (ctid >= '(10,0)::tid)
    Filter: (id = 1)
```

Того же эффекта можно достичь с помощью указания DisableIndex() или других указаний, которые отключают индекс. Планировщик не будет использовать индекс и будет

выбирать между оставшимися альтернативами Sec Scan или TID Scan. В этом случае планировщик будет использовать TID Scan.

3.12.17.3 Отключение индекса в плане выполнения

В данном подразделе демонстрируется возможность отключения конкретных индексов планировщиком запросов СУБД.

В качестве примера создадим две таблицы с первичными ключами:

```
CREATE TABLE a(a int primary key);  
CREATE TABLE b(b int primary key);
```

Изначально запрос с условием выборки по первичному ключу использует индексное сканирование (Index Only Scan) для таблицы b.

```
EXPLAIN SELECT * FROM a JOIN b ON a.a=b.b WHERE a.a=1;  
  
QUERY PLAN  
  
-----  
Nested Loop  (cost=0.15..8.18 rows=1 width=8)  
  -> Seq Scan on a  (cost=0.00..0.00 rows=1 width=4)  
      Filter: (a = 1)  
  -> Index Only Scan using b_pkey on b  (cost=0.15..8.17  
rows=1 width=4)  
      Index Cond: (b = 1)
```

Однако с помощью указания `/*+ DisableIndex(b b_pkey)*/` можно явно запретить планировщику применять указанный индекс:

```
EXPLAIN /*+ DisableIndex(b b_pkey)*/ SELECT * FROM a JOIN b ON  
a.a=b.b WHERE a.a=1;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.00..42.01 rows=13 width=8)  
-> Seq Scan on a (cost=0.00..0.00 rows=1 width=4)  
    Filter: (a = 1)  
-> Seq Scan on b (cost=0.00..41.88 rows=13 width=4)  
    Filter: (b = 1)
```

В результате этого в плане выполнения происходит замена индексного сканирования на последовательное (Seq Scan).



Отключение индекса может быть не только адресным для одного запроса, но и глобальным для всей сессии. Команда SET enable_indexscan = off; отключает все индексные сканирования.

3.12.17.4 Манипулирование связками таблиц

При помощи указаний можно изменить порядок связки таблиц не меняя при этом порядок JOIN операторов в запросе, например с помощью указания Leading().

В качестве примера создадим две таблицы с первичными ключами:

```
CREATE TABLE a(a int primary key);  
CREATE TABLE b(b int primary key);
```

По умолчанию в плане выполнения запроса таблицы сканируются в том же порядке, что и появляются в самом запросе:

```
EXPLAIN SELECT * FROM a JOIN b ON a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=2550 width=8)  
  Hash Cond: (a.a = b.b)  
-> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
-> Hash (cost=35.50..35.50 rows=2550 width=4)  
    -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

Их порядок связки можно поменять, не переписывая запрос, если добавить указание Leading():

```
EXPLAIN /*+ Leading( (b a) ) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=2550 width=8)  
  Hash Cond: (b.b = a.a)  
    -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
          -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)
```

В зависимости от условий возможно поменять тип связки: Hash join, Merge join и Nested loop.

Связку Hash join необходимо использовать в случае, если в запросе фигурируют операторы равенства (=) и hash таблица помещается в памяти, а также если необходимо прочитать все результирующие данные до конца.

```
EXPLAIN /*+ HashJoin( b a ) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=2550 width=8)  
  Hash Cond: (a.a = b.b)  
    -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
          -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

Связка Merge join не использует hash таблицу и применима для больших данных, если обе стороны можно удачно отсортировать.


```
EXPLAIN /*+ MergeJoin( a b ) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Merge Join (cost=0.31..203.06 rows=2550 width=8)  
  Merge Cond: (a.a = b.b)  
    -> Index Only Scan using a_pkey on a (cost=0.15..82.41  
rows=2550 width=4)  
    -> Index Only Scan using b_pkey on b (cost=0.15..82.41  
rows=2550 width=4)
```

Связку Nested loop можно использовать, если одна из сторон связки имеет мало значений или для приоритетного получения первых записей.

```
EXPLAIN /*+ NestLoop( a b ) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.15..519.34 rows=2550 width=8)  
  -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
  -> Index Only Scan using b_pkey on b (cost=0.15..0.19  
rows=1 width=4)  
      Index Cond: (b = a.a)
```

В комбинации можно использовать все четыре указания из данного подраздела, чтобы изменять и порядок, и тип связей более оптимальным образом изменяя только первые строки запроса.

Также можно косвенно влиять на связи путем изменения количества возвращаемых строк и тем самым влиять на выбор типов связей с помощью директивы Rows.

В качестве примера рассмотрим изначальный план, который показывал значение количества строк 2550:

```
EXPLAIN SELECT * FROM a JOIN b ON a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=2550 width=8)  
  Hash Cond: (a.a = b.b)  
    -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
        -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

В данном случае можно либо умножить rows=2550 на некоторое число:

```
EXPLAIN /*+ Rows(a b *20) */ SELECT * FROM a JOIN b ON a.a=b.b;
```

QUERY PLAN

```
-----  
--  
Hash Join (cost=67.38..109.58 rows=51000 width=8)  
  Hash Cond: (a.a = b.b)  
    -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
        -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

Либо прибавить некоторое число:

```
EXPLAIN /*+ Rows(a b +100000) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=102550 width=8)  
  Hash Cond: (a.a = b.b)  
    -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
        -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

Либо задать точное желаемое значение:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
EXPLAIN /*+ Rows(a b #100) */ SELECT * FROM a JOIN b ON  
a.a=b.b;
```

QUERY PLAN

```
-----  
Hash Join (cost=67.38..109.58 rows=100 width=8)  
  Hash Cond: (a.a = b.b)  
    -> Seq Scan on a (cost=0.00..35.50 rows=2550 width=4)  
    -> Hash (cost=35.50..35.50 rows=2550 width=4)  
          -> Seq Scan on b (cost=0.00..35.50 rows=2550 width=4)
```

3.12.17.5 Контроль количества параллельных процессов

Для примера создадим таблицу и вставим в нее 10 млн записей.

```
CREATE TABLE a(a int primary key);  
INSERT INTO a SELECT * FROM generate_series(1,10*1000*1000);
```

Попытаемся выбрать 1 млн записей из этой таблицы. При этом с помощью указания `Parallel()` можно зарезервировать два процесса для параллельного выполнения запроса.

```
EXPLAIN EXPLAIN (ANALYZE,VERBOSE)
/*+ Parallel(a 2 hard) */ SELECT * FROM a where a between 1 and 1000*1000;
QUERY PLAN
-----
Gather (cost=1000.00..6641.60 rows=56416 width=4) (actual time=157.662..309.571
rows=1000000.00 loops=1)
  Output: a
  Workers Planned: 2
  Workers Launched: 2
  Buffers: shared hit=38 read=7162 dirtied=4425 written=5469
  -> Parallel Bitmap Heap Scan on public.a (cost=0.00..0.00 rows=23507 width=4)
(actual time=155.439..281.726 rows=333333.33 loops=3)
    Output: a
    Recheck Cond: ((a.a >= 1) AND (a.a <= 1000000))
    Heap Blocks: exact=1063
    Buffers: shared hit=38 read=7162 dirtied=4425 written=5469
    Worker 0:  actual time=154.436..298.378 rows=386008.00 loops=1
      Heap Blocks: exact=1708
      Buffers: shared hit=20 read=1708 dirtied=1708 written=1707
    Worker 1:  actual time=154.404..300.543 rows=373804.00 loops=1
      Heap Blocks: exact=1654
      Buffers: shared hit=17 read=1657 dirtied=1654 written=1565
    -> Bitmap Index Scan on a_pkey (cost=0.00..1116.59 rows=56416 width=0)
(actual time=156.830..156.830 rows=1000000.00 loops=1)
      Index Cond: ((a.a >= 1) AND (a.a <= 1000000))
      Index Searches: 1
      Buffers: shared hit=1 read=2734 written=1141
Planning:
  Buffers: shared hit=1 read=1
Planning Time: 0.254 ms
Execution Time: 342.830 ms
```

3.12.17.6 Получение первых строк в приоритетном порядке

Данный подраздел демонстрирует оптимизацию запроса для сценария, когда важно быстро получить первые строки результата, даже если полное выполнение может занять больше времени по сравнению с планом по умолчанию.

Создадим таблицу с первичным ключом:

```
CREATE TABLE a(a int primary key);
```

Свяжем таблицу с последовательностью чисел:

```
EXPLAIN ANALYZE SELECT * FROM generate_series(1,1000) gs JOIN a
ON a.a=gs;
```

QUERY PLAN

```
-----
Hash Join (cost=0.02..12.70 rows=5 width=8) (actual
time=0.013..0.014 rows=0.00 loops=1)
  Hash Cond: (gs.gs = a.a)
    -> Function Scan on generate_series gs (cost=0.00..10.00
rows=1000 width=4) (never executed)
    -> Hash (cost=0.00..0.00 rows=1 width=4) (actual
time=0.010..0.011 rows=0.00 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 8kB
    -> Seq Scan on a (cost=0.00..0.00 rows=1 width=4)
(actual time=0.010..0.010 rows=0.00 loops=1)
```

Исходный план с использованием хэш-соединения (Hash Join) эффективен для полного перебора, но требует построения хэш-таблицы перед началом выдачи данных, что задерживает появление первых строк.

Применение указания `/*+ NestLoop(gs a) */` меняет стратегию, заставляя планировщик СУБД использовать вложенный цикл (Nested Loop). Применение подсказки `NestLoop()` позволяет получать первые строки быстрее:

```
EXPLAIN ANALYZE /*+ NestLoop(gs a) */ SELECT * FROM
generate_series(1,1000) gs JOIN a ON a.a=gs
```

QUERY PLAN

```
-----
Nested Loop (cost=0.00..22.50 rows=5 width=8) (actual
time=0.478..0.479 rows=0.00 loops=1)
  Join Filter: (a.a = gs.gs)
    -> Function Scan on generate_series gs (cost=0.00..10.00
rows=1000 width=4) (actual time=0.128..0.179 rows=1000.00
loops=1)
    -> Seq Scan on a (cost=0.00..0.00 rows=1 width=4) (actual
time=0.000..0.000 rows=0.00 loops=1000)
```

3.12.17.7 Использование указаний No*

Указания `NoNestLoop`, `NoHashJoin`, `NoMergeJoin`, `NoSeqScan`, `NoTidScan`, `NoIndexScan`, `NoIndexOnlyScan`, `NoBitmapScan`, `NoMemoize` исключают одну из альтернатив, оставляя планировщику выбор среди оставшихся.

В качестве примера создадим таблицу с первичным ключом:

№ изменения: _____	Подпись отв. лица: _____	Дата внесения изм: _____
--------------------	--------------------------	--------------------------

```
CREATE TABLE a(a int primary key);
```

Например, в таком запросе по умолчанию используется TID:

```
EXPLAIN SELECT * FROM a WHERE ctid='(1,1)' and a=1;
```

QUERY PLAN

Tid Scan on a (cost=0.00..4.02 rows=1 width=4)

TID Cond: (ctid = '(1,1)::tid)

Filter: (a = 1)

А после применения указания NoTidScan() используется одна из остающихся альтернатив Index Scan по условию a=1:

```
EXPLAIN /*+ NoTidScan( a ) */ SELECT * FROM a WHERE  
ctid='(1,1)' and a=1;
```

QUERY PLAN

Index Scan using a_pkey on a (cost=0.43..8.46 rows=1 width=4)

Index Cond: (a = 1)

Filter: (ctid = '(1,1)::tid)

4. ОШИБКИ

В случае ошибки компонент останавливает разбор, и в большинстве случаев применяет указания, уже разобранные к этому моменту.

Таблица 4.1 – Типичные ошибки

Синтаксические ошибки	Ошибки в записи или неправильные имена указаний Ошибки выводятся в журнал сообщений сервера с уровнем, заданным в параметре <code>pg_hint_plan.message_level</code> , если параметр <code>pg_hint_plan.debug_print</code> имеет значение, отличное от <code>off</code> .
Неправильные обращения к объектам	Указания с неправильными обращениями к объектам просто игнорируются. Ошибки такого типа отмечаются в журнале как «неиспользованные указания» при тех же условиях, что и синтаксические ошибки.
Избыточные или конфликтующие указания	Когда указания избыточны или одно указание конфликтует с другим, действовать будет последнее указание. Ошибки такого типа отмечаются как «дублирующиеся указания» в журнале сообщений сервера при тех же условиях, что и синтаксические ошибки.
Вложенные комментарии	Комментарий с указаниями не может содержать в себе другой блочный комментарий. В случае возникновения такой ошибки, компонент прекращает разбор.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

SQL	–	Structured Query Language
БД	–	База данных
ОС	–	Операционная система
СУБД	–	Система управления базами данных

Лист регистрации изменений

№ изменения: _____

Подпись отв. лица: _____

Дата внесения изм: _____