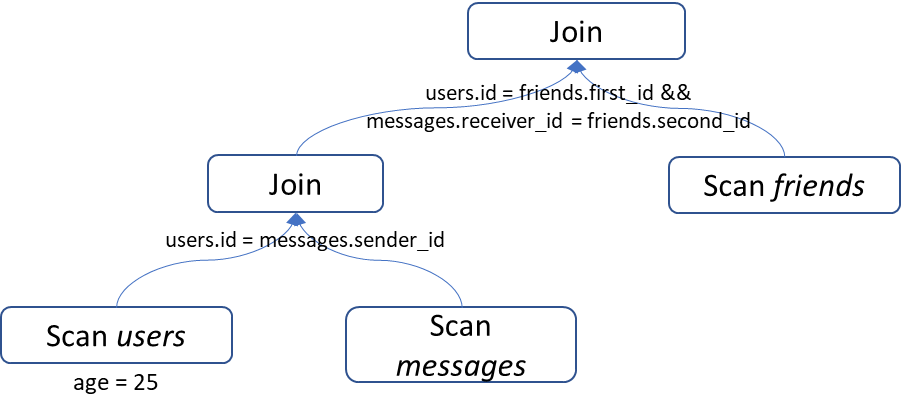
### Простой перебор джоинов

#### Контекст

В базах данных и реляционной алгебре есть понятие джоина двух множеств строчных значений в виде таблиц. Джоин в самом примитивном случае без условия (cross join) представляется в виде декартового произведения его операндов. Либо джоин может быть с условием, которое применяется поверх декартового произведения (inner join). Как привило, это условие - сравнение атрибутов от левого и правого операндов.

Джоин от набора таблиц представляется в виде бинарного дерева, у которого листовые узлы представляют сканы по базовым таблицам, а промежуточные узлы - операторы джоина его поддеревьев (джоины меньшего порядка, либо сканы базовых таблиц). Джоин-поддерево может использовать для другого джоина любой атрибут, определенный в какой-либо базовой таблице внутри этого поддерева.



Выбор оптимальной по исполнению структуры дерева джоина от фиксированного набора таблиц за приемлемое время является одной из важнейших задач оптимизации запросов в современных реляционных базах данных.

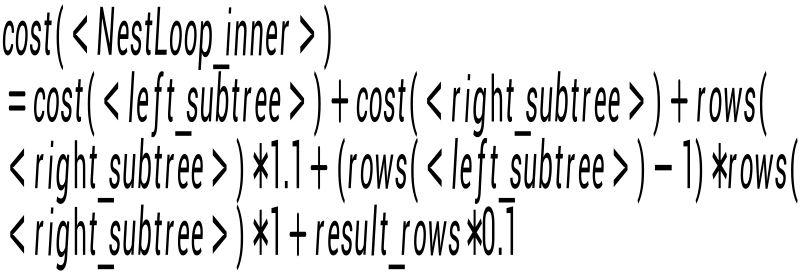
##### Оценка эффективности джоина и скана

Каждому варианту дерева джоина можно задать стоимость исполнения, которая определяется стоимостью самого верхнего оператора джоина. Та в свою очередь рекурентно определяется стоимостями дочерних поддеревьев и количеством обрабатываемых строк на входе.

На самом нижнем уровне дерева для сканов таблиц считаем, что стоимость чтения одной строки равна единице. Сканы могут включать в себя условия фильтрации (предикат) в виде конъюнкции из равенств атрибутов таблицы некоторым константам. В этом случае считаем, что стоимость вычисления предиката (всех конъюнктов в условии) на одной строке также единична, а итоговая стоимость скана удваивается, несмотря на то что результирующее количество строк может уменьшиться (ниже будет представлена формула).

Операторы джоина по равенству атрибутов (inner join) могут быть реализованы двумя способами: вложенным циклом (NestLoop), когда для каждой строки левого поддерева полностью сканируется правое на поиск соответствия атрибутов в условии джоина, либо через построение хэш-таблицы для правого поддерева и поиск строк соответствия левого поддерева в этой хэш-таблице (HashJoin). Причём для NestLoop после первой итерации датасет правого поддерева кэшируется и далее на последующих итерациях ищутся строки соответствия уже из кэша. Считаем, что стоимость заполнения кэша на одну строчку равна десятой доле единицы (0.1). Аналогичная стоимость тратится на формирование результирующих строк джоина, включающее конкатенацию входящих строк и выделение необходимых атрибутов (применение проекции).

Тогда итоговая стоимость для NestLoop алгоритма при наличии условия джоина будет считаться как:

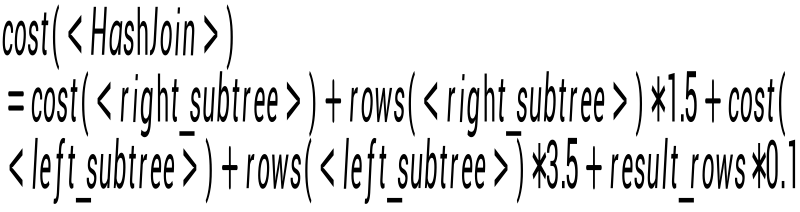
, где

*rows(<subtree>)* - количество строк в поддереве джоина,

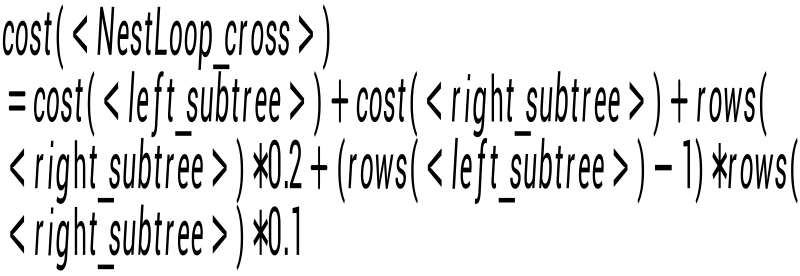
*cost(<subtree>)* - стоимость исполнения поддерева,

*result\_rows* - результирующее кол-во строк джоина.

Для HashJoin мы предполагаем, что стоимость добавления строки в хэш-таблицу равна 1.5 (единица на вычисление хэш-значения от ключа плюс половина на поиск местоположения внутри бакета), а проверка на соответсвие в этой хэш-таблице равна 3.5 (единица на вычисление хэш-значения от ключа плюс 2.5 на проверку соответствия со строками в целевом бакете), т.е.:

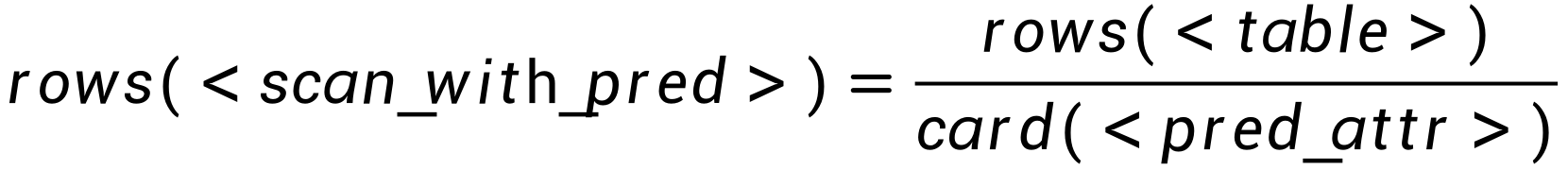


Для джоина без условия (cross join) единственной стратегией реализации является вложенный цикл (NestLoop) и его стоимость будет считаться без учёта проверок предиката как:



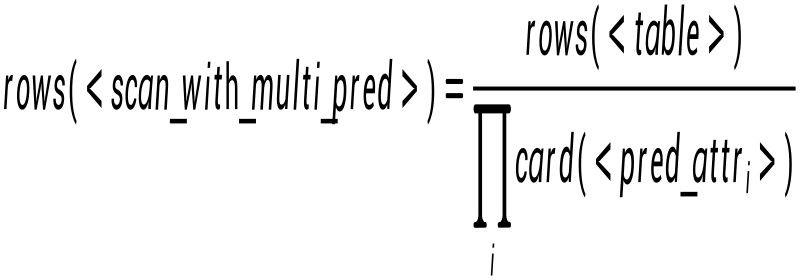
##### **Оценка количества строк для джоина и скана**

Как было сказано, к скану таблицы может быть применено условие фильтрации (предикат), в общем случае состоящее из конъюнкции из равенств некоторого атрибута константе. В случае одного равенства оценочно результрующее количество строк уменьшится кратно количеству уникальных значений (кардинальности) атрибута в предикате:

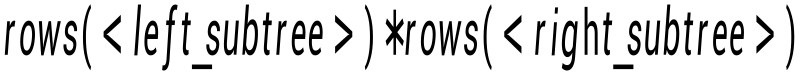
, где

*rows(<table>)* - количество строк в таблице (задаётся на входе),

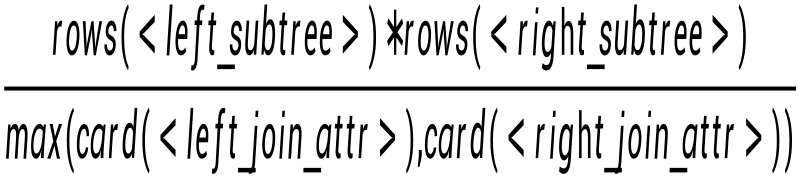
*card(<pred\_attr>)* - кардинальность (количество уникальных значений) атрибута в предикате. В случае когда у таблицы фильтрующих равенств несколько результрующий размер скана сокращается на произведение кардинальностей этих атрибутов:



Для джоина без условия (cross join) итоговое количество строк определяется произведением количеств строк в его поддеревьях:

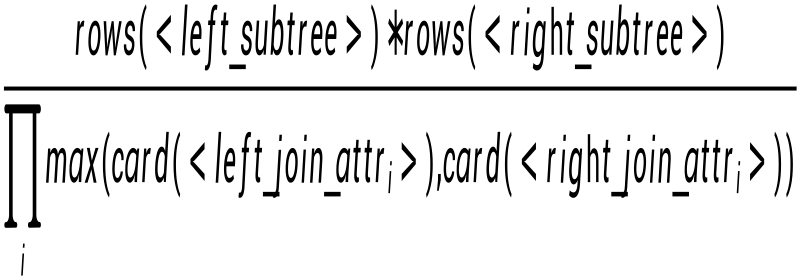


Для inner join`а результирующее количество строк будет считаться как:

, где

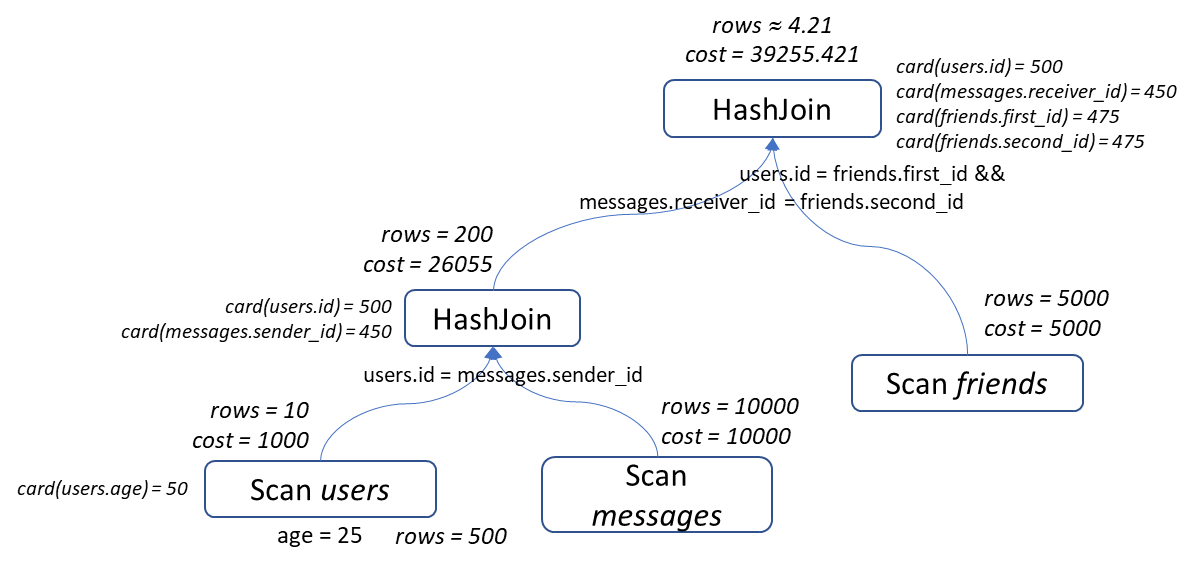
*rows(<subtree>)* - количество строк в поддереве и

*card(<join\_attr>)* - кардинальность атрибута в условии джоина по отношению к базовой таблице, к которой атрибут относится. Если имеем конъюнкцию равенств атрибутов в условии джоина (джоин по нескольких атрибутам), то знаменатель итоговой стоимости будет представлен произведением максимумов по парам атрибутов в конъюнктах:



Следует отметить, что нормально для узлов плана иметь **вещественное** (не целое) количество возвращаемых строк. Так как данная оценка является в некотором роде математическим ожиданием величины.

Размеры базовых таблиц (количества строк) и кардинальности его атрибутов, участвующих в джоинах и предикатах, задаются на входе программы.



#### Описание задачи

Среди всех возможных альтернатив найти дерево джоина с минимальной стоимостью исполнения.

Структура джоина задаётся в виде перечисления базовых таблиц и явных операций джоина для пар номеров таблиц и атрибутов, участвующих в условии равенства джоина. Любые две таблицы могут быть сджоинены по этим атрибутам, если данный джоин определен явно, либо кросс-джоином, если условие не определено. Джоин-поддерево может использовать для джоина верхнего уровня любые атрибуты, определенные в какой-либо из базовых таблиц внутри этого поддерева. Для каждого джоина необходимо подобрать наиболее оптимальный алгоритм реализации (NestLoop vs. HashJoin).

Также для каждой таблицы могут задаваться атрибуты, участвующие в условии фильтрации (предикате) при скане этой таблицы. Применение этого условия сокращает результирующее количество строк от скана по этой таблицы по формуле, объявленной выше. Размеры базовых таблиц, как и кардинальности джоин-атрибутов и атрибутов в предикатах скана, задаются на входе.

Для джоина до 13 таблиц решение по времени должно укладываться в 12 секунд и потреблять не более 256 Mб памяти.

##### 2.1. Описание входных и результирующих данных

Входные данные задаются в следующем формате построчно:

* количество таблиц в джоине
* в одну строку количество строк по номерам таблиц
* количество заваемых атрибутов с кардинальностями
* через строчку атрибуты в виде литеры с кардинальностями в формате:

*<table\_num> <filter\_attr> <card\_value>*

* количество предикатов на сканы
* через строчку атрибуты в предикатах в формате:

*<table\_num> <filter\_attr>*

* количество джоин предикатов
* через строчку явные джоин предикаты между таблицами в формате:

*<table\_num\_1> <table\_num\_2> <join\_table\_1\_attr> <join\_table\_2\_attr>*

***Примечание****:* в условии и далее в результате таблицы нумеруются с **единицы**.

В результате необходимо вывести дерево джоина минимальной стоимости в виде правильного скобочного выражения из номеров таблиц, атрибутов в предикатах на сканы и условий джоинов с итоговой стоимостью всего плана через пробелы.

Формат вывода джоина следующий:

*(<operand1> <operand2> {<join\_clause1>} {<join\_clause2>} ...)*

В роли операнда джоина может быть другой джоин (поддерево), либо скан по таблице в виде номера таблицы с приписанными справа буквами атрибутов в условии предиката:

*<table\_num><filter\_attr1><filter\_attr2>...*

Каждый конъюнкт в условии джоина (join clause) представляются в виде пары атрибутов в равенстве конъюнкта внутри фигурных скобок. Атрибуты там описываются номером таблицы и буквой атрибута через точку:

*<left\_table\_num>.<left\_attr> <right\_table\_num>.<right\_attr>*

В случае cross джоина (джоина без условий) список конъюнктов будет отсутствовать полностью. Например,

* *(1a 2b {1.c 2.d} {1.e 2.f})* - inner-join между таблицами 1 и 2 с условием 1.c=2.d && 1.e=2.f и предикатами на сканы по атрибутам *a* и *b*, применённым к таблицам 1 и 2 соответсвенно
* *(3 4)* - cross-join между таблицами 3 и 4.

##### 2.2.Пример.

На входе:

4

10 12 15 8

5

1 a 3

2 b 5

3 b 10

2 c 1

4 d 5

0

3

1 2 a b

1 3 a b

2 4 c d

Результат:

((1 3 {1.a 3.b}) (4 2 {4.d 2.c}) {1.a 2.b}) 238.98

### Перебор джоинов в Postgres

#### Контекст

Полный перебор всевозможных вариаций дерева джоинов - достаточно дорогая операция в СУБД особенно при наличии значительного числа базовых таблиц. Поэтому все современные движки планирования запросов используют различные эвристические правила (prunning rules), отсекающие те или иные "неудачные" паттерны поддеревьев джоина. К примеру, планировщик известной open-source СУБД PostgreSQL осуществляет перебор cross-join`ов на самых верхних уровнях общего дерева джоина - уже на базе сформированных inner-join поддеревьев. Это оправдано, если предположить, что cross-join`ы относительно inner-join`ов привносят большую стоимость в текущий уровень дерева джоинов. Поэтому, дабы не множить эту стоимость на разных уровнях выше, хотелось бы отложить cross-join`ы после применения по максимуму inner-join`ов в поддеревьях. Данное правило является эвристическим, поскольку итоговое дерево джоинов минимальной стоимости с "отложенными" cross-join`ами не всегда будет являться самым оптимальным (т.е. иметь минимальную стоимость) среди всевозможных альтернатив.

Ещё при формировании нового джоин оператора имеет смысл рассматривать не только прямые равенства атрибутов в условии джоина, объявленные в начальном запросе, но и транзитивные. Например, если таблицы A и B джоинятся по условию *A.a = B.b*, а B и C - по *B.b = C.c*, то мы вправе рассматривать джоин A и С как inner-join по условию *A.a = C.c*. В противном случае, мы бы рассматривали этот джоин как cross-join с увеличенной итоговой стоимостью. К тому же аналогичным образом предикаты по таблицам можно выводить тразитивно, т.е. если таблицы A и B джоинятся по равенству *A.a = B.b* и для таблицы *A* определен предикат по атрибуту *a*, то можно сократить результат скана по таблице *B*, транзитивно применив подобный предикат по атрибуту *b*. В этом случае джоин по таблицам A и B следует рассматривать как cross джоин (без условия *A.a = B.b*), так как вся необходимая фильтрация для джоина уже произошла на уровне сканов базовых таблиц. Также в случае вывода нескольких эквивалентных условий джоина для пары поддеревьев фактическую фильтрацию декартового произведения исходных строк будет проводить только одно из них. Например, при джоине таблицы A на результат джона B и C по условию A.a = B.b и при джоине B и C по условию B.b = C.c применение изначального A.a = B.b вкупе с производным A.a = C.c не даст никакого эффекта относительно применения какого-либо одного из них. При этом оценивая количество строк данного джоина мы действуем оптимистично и стремимся максимально полно сократить финальную стоимость итогового плана.

#### Описание задачи

Среди всевозможных альтернатив c "отложенными" cross-join`ами и применением возможных транзитивных условий равенства для джоина найти дерево с минимальной стоимостью исполнения.

Структура джоина задаётся в виде перечисления базовых таблиц, предикатов сравнения атрибутов с константой, применимых к таблицам, и явных операций джоина для пар номеров таблиц и атрибутов, участвующих в условии равенства джоина. Любые две таблицы могут быть сджоинены по этим атрибутам, если данный джоин определен явно или выводится транзитивно, либо cross-join`ом, если условие никак не может быть определено. Джоин-поддерево может использовать для джоина верхнего уровня любые атрибуты, определенные в какой-либо из базовых таблиц внутри этого поддерева. Cross-join в рассматриваемом дереве всегда должен быть не ниже по уровню inner-join`а. Для каждого inner join`а подобрать наиболее оптимальный алгоритм реализации (NestLoop vs. HashJoin).

Также для каждой таблицы могут задаваться атрибуты, участвующие в условии фильтрации (предикате) при скане этой таблицы. Необходимо рассмотреть его, а также опционально всевозможные транзитивно выводимые предикаты. Размеры базовых таблиц, как и кардинальности джоин-атрибутов задаются на входе константами.

Для джоина до 18 таблиц при количестве связей между таблицами, соразмерном количеству таблиц (отличается на константу-множитель), решение по времени должно укладываться в 19 секунд и потреблять не более 256 Mб памяти.

##### 2.1. Описание входных и результирующих данных

Как в предыдущей задаче (пункт 2.1).

##### 2.2.Пример.

На входе:

4

10 12 15 8

5

1 a 3

2 b 5

3 b 10

2 c 1

4 d 5

0

3

1 2 a b

1 3 a b

2 4 c d

Результат:

(4 (2 (1 3 {1.a 3.b}) {2.b 3.b}) {4.d 2.c}) 228.18