Улучшения оптимизатора в Firebird 5.0

Дмитрий Еманов
dimitr@firebirdsql.org

Firebird Project
https://www.firebirdsql.org/

РЕД СОФТ
https://www.red-soft.ru/

Больше информации в explained-плане

v5.0 Beta 1

- Тип курсора (select expression / sub-query / named cursor)
- Атрибуты курсора (invariant / scrollable)
- Номера позиций (строка/столбец) в PSQL

Больше информации в explained-плане

v5.0 Beta 1

- Тип курсора (select expression / sub-query / named cursor)
- Атрибуты курсора (invariant / scrollable)
- Номера позиций (строка/столбец) в PSQL

В очереди

- Возможность рекурсивного вывода планов для селективных процедур
- Вывод оценочной кардинальности для каждого узла расширенного плана

Раннее вычисление инвариантных предикатов

Примеры

- Фейковые условия

```
WHERE 1 = 0
```

- Инвариантные условия:

```
WHERE RDB$GET_CONTEXT(«USER_SESSION», «PARAM») = 1
```

- Еще сложнее:

```
WITH RECURSIVE TR AS (
      SELECT T.*, 1 AS LEVEL FROM "Tree" T
        WHERE PARENT_ID IS NULL
      UNION ALL
      SELECT T.*, TR.LEVEL + 1 AS LEVEL FROM "Tree" T, TR
        WHERE T.PARENT_ID = TR.ID AND TR.LEVEL < :MAX_LEVEL
)
SELECT ID, LEVEL FROM TR
```

Раннее вычисление инвариантных предикатов

В плане

```
Select Expression
    -> Filter (preliminary)
        -> Table "T1" Full Scan


Select Expression
    -> Recursion
        -> Filter
            -> Table "Tree" as "TR T" Full Scan
        -> Filter (preliminary)
            -> Filter
                -> Table "Tree" as "TR T" Full Scan
```

Трансформация outer join в inner join

v5.0 Beta 1

- Если WHERE-предикат не умеет обрабатывать NULLы в «правой» таблице, то во внешнем джойне нет смысла
- `A LEFT JOIN B ON A.ID = B.ID WHERE B.FLD1 = 0`
- Планы могут измениться!
- Если нужно сохранить «прибитый гвоздиком» порядок соединения, то `B.FLD1 IS NOT NULL`

TODO

- Параметр конфига — нужен или нет?

Стоимостной выбор между HASH и MERGE JOIN

MERGE JOIN

- Был «заглушен» в Firebird 3.0 в пользу HASH JOIN
- Но в ряде случаев может таки выигрывать
- В Firebird 5.0 разрешен снова,
  сделан выбор на основе стоимости

TODO

- MERGE JOIN можно использовать также в случае предварительно сортированных потоков

Стоимостной выбор между NESTED LOOP и HASH JOIN

Проблема

- PLAN JOIN (T1 …, T2 INDEX (T2_PK))
- Все плохо, если выборка из T1 превышает размер T2
- cardinality(T1) = 100000, cardinality(T2) = 100 — каждая запись T2 читается 1000 раз!
- Плюс как минимум 100000 индексных фетчей!

Стоимостной выбор между NESTED LOOP и HASH JOIN

Проблема

- PLAN JOIN (T1 …, T2 INDEX (T2_PK))
- Все плохо, если выборка из T1 превышает размер T2
- cardinality(T1) = 100000, cardinality(T2) = 100 — каждая запись T2 читается 1000 раз!
- Плюс как минимум 100000 индексных фетчей!

Решение

- Читать T2 однократно, Ватсон!
- В этом нам поможет HASH JOIN
- Стоимость хеширования тоже ненулевая, надо считать что выгоднее

Стоимостной выбор между NESTED LOOP и HASH JOIN

Было

```
PLAN JOIN (T1 NATURAL, T2 INDEX (T2_PK))

Elapsed time = 1.253 sec
```

Стало

```
PLAN HASH (T1 NATURAL, T2 NATURAL)

Elapsed time = 0.286 sec
```

Трансформация подзапросов в semi-joins

Идея

- IN/EXISTS это по сути semi-join
- Доработка алгоритмов NESTED LOOP и HASH / MERGE JOIN для поддержки семантики semi-join (и заодно anti-join)
- Трансформация IN/EXISTS в semi-join
- Далее оптимизатор выбирает алгоритм соединения
- Ситуация аналогична предыдущей — если подзапрос возвращает немного строк, то его выгоднее кешировать, чем перевыполнять каждый раз

Трансформация подзапросов в semi-joins

Идея

- IN/EXISTS это по сути semi-join
- Доработка алгоритмов NESTED LOOP и HASH / MERGE JOIN для поддержки семантики semi-join (и заодно anti-join)
- Трансформация IN/EXISTS в semi-join
- Далее оптимизатор выбирает алгоритм соединения
- Ситуация аналогична предыдущей — если подзапрос возвращает немного строк, то его выгоднее кешировать, чем перевыполнять каждый раз

Что дальше

- Трансформация NOT IN / NOT EXISTS → anti-join

Трансформация подзапросов в inner joins

Идея

- where t1.id in (select t2.id from t2 where …)
  → t1 semi-join (select t2.id from t2 where …) on t1.id = t2.id
  → t1 semi-join t2 on t1.id = t2.id and …
- where t1.id in (select t2.id from t2 where …)
  → t1 join (select distinct t2.id from t2 where …) on t1.id = t2.id

Трансформация подзапросов в inner joins

Идея

- where t1.id in (select t2.id from t2 where …)
  → t1 semi-join (select t2.id from t2 where …) on t1.id = t2.id
  → t1 semi-join t2 on t1.id = t2.id and …
- where t1.id in (select t2.id from t2 where …)
  → t1 join (select distinct t2.id from t2 where …) on t1.id = t2.id

Что это нам дает

- Не только порядок соединения T1→T2, но и обратный
- При обратном порядке уже может оказаться дешевле алгоритм NESTED LOOP
- Но DISTINCT это лишняя сортировка
- Пусть оптимизатор сам оценивает!

Эффективное выполнение IN <constant list>

Сейчас

- Индексный поиск на каждое значение из списка
- Номера записей выставляются в битмапе
- Если ключей много, то оптимизатор отказывается от использования индекса
- Лимит в 1500 элементов, ибо внутри трансформируется в

```
FLD = 1 OR FLD = 2 OR …
```

и работает рекурсивно

Эффективное выполнение IN <constant list>

Что можно сделать

- Не трансформировать список, передавать и обрабатывать «как есть» (прощаемся с лимитом на число элементов)
- Значения отсортировать
- Если индекса нет — бинарный поиск по отсортированному массиву
- Если есть — один индексный поиск по диапазону от min-значения до max-значения
- При этом не обязательно перебирать все ключи на всех листовых страницах, можно двигаться «скачками»
- При небольшом числе значений старый способ может оказаться быстрее, нужна оценка стоимости обоих вариантов

Подсказка OPTIMIZE FOR

Идея

- Указывает предпочтения клиента по выборке записей
- `ALL ROWS` / `FIRST ROWS`
- Неявно используется с Firebird 3.0 (`FIRST <n>`, `EXISTS`)
- Явно и в синтаксисе и в конфиге — в Firebird 5.0
- Влияет на выбор плана, в частности `SORT` vs `ORDER`

TODO

- Стоимостная оценка с учетом числа результирующих строк
- Управление на уровне сессии (`DPB` / `ALTER SESSION`)

Вопросы?