FIREBIRD CONF'26

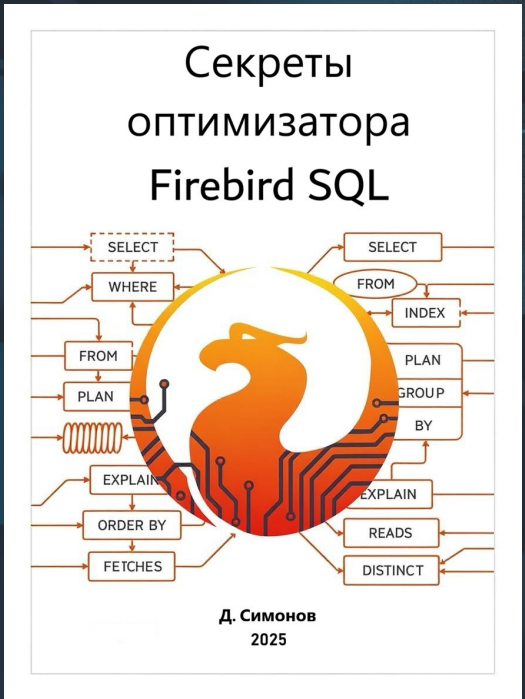
Организатор конференции РЕДСОФТ

Денис Симонов

Трюки с оптимизатором Firebird 5.0

Материалы

- Статья «Методы доступа используемые в Firebird»
- Книга «Секреты оптимизатора Firebird»



FIREBIRD CONF'26

О чём будем говорить?

- Отключение индекса +0 и || ‘’
- Множество индексов vs композитный индекс
- Частичные индексы
- Множество предикатов фильтрации
- OuterJoinConversion = true/false
- SubQueryConversion = true/false
- Оптимизация широких группировок
- Что есть и возможно будет в Firebird 6.0
- Чего не будет в FB 6.0, но хотелось бы

Отключение использования индекса

- Превращение индексированного поля в выражение с помощью +0 или || ‘’

```
SELECT *
FROM T
WHERE T.FIELD+0 = 1;

SELECT COUNT(*)
FROM T1
JOIN T2 ON T2.ID+0 = T1.REF_ID;
```

```
SELECT
  T.FIELD,
  …
FROM T
ORDER BY T.FIELD+0;

SELECT
  T.FIELD+0 AS FIELD,
  …
FROM T
ORDER BY 1;
```

Для чего отключают индекс?

- Изменение порядка соединения (JOIN)
- Принудительный HASH JOIN
- Принудительный SORT вместо ORDER
- Неравномерное распределение значений в индексе

Недостатки отключения индекса +0

- В запросах со множеством JOIN трудно угадать последствия. Вместо точечного воздействия может быть перестроен весь план (и порядок соединений)
- Подсказки вроде +0 лишают данных о селективности индекса (начинают использоваться константы для предикатов)

Когда оптимизатор сам выбирает HASH JOIN, он может использовать селективность индексов из полей условий связи для расчёта кардинальности соединения

Композитный индекс или несколько индексов

- Селективность композитного индекса точнее, чем селективность пересечения битовых масок нескольких индексов
- Многократное сканирование одного композитного индекса дешевле, чем нескольких индексов

Минус:

- Нужно думать о порядке полей в композитном индексе.

FIREBIRD CONF'26

КОМПОЗИТНЫЙ ИНДЕКС ДЕШЕВЛЕ ПРИ МНОГОКРАТНОМ СКАНИРОВАНИИ

```
SELECT COUNT(*)
FROM
  HORSE
WHERE HORSE.CODE_MOTHER > 0
  AND HORSE.CODE_BREED = 65
  AND NOT EXISTS(
    SELECT * FROM COVER
    WHERE COVER.CODE_MOTHER = HORSE.CODE_MOTHER
      AND COVER.CODE_FATHER = 742363
  );

PLAN (COVER INDEX (FK_COVER_MOTHER, FK_COVER_FATHER))
PLAN (HORSE INDEX (FK_HORSE_BREED, FK_HORSE_MOTHER))

 COUNT
======================
                105824

Current memory = 557153984
Delta memory = 352
Max memory = 581203216
Elapsed time = 0.939 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 758223
```

КОМПОЗИТНЫЙ ИНДЕКС ДЕШЕВЛЕ ПРИ МНОГОКРАТНОМ СКАНИРОВАНИИ

```
CREATE INDEX IDX_COVER_FATHER_MOTHER ON COVER (CODE_FATHER, CODE_MOTHER);

PLAN (COVER INDEX (IDX_COVER_FATHER_MOTHER))
PLAN (HORSE INDEX (FK_HORSE_BREED, FK_HORSE_MOTHER))

 COUNT
======================
                105824

Current memory = 555365664
Delta memory = 352
Max memory = 595697568
Elapsed time = 0.256 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 544101
```

Когда частичный индекс «рулит»

FIREBIRD CONF'26

- Частичный индекс – это индекс по части записей, которые удовлетворяют предикату фильтрации, заданному при создании этого индекса
- Частичные индексы занимают меньше места, чем обычные

Минусы:

- Его селективность менее точная
- Для использования оптимизатором необходимо указать в запросе тоже условие фильтрации, что и при создании индекса (за исключением IS NOT NULL)

ЧАСТИЧНЫЙ ИНДЕКС ДЛЯ ПОЛЯ С МНОЖЕСТВОМ NULL ЗНАЧЕНИЙ

```
SELECT
  COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
WHERE HORSE.MICROCHIP IS NULL;

PLAN JOIN (HORSE INDEX (HORSE_IDX_MICROCHIP), COLOR
INDEX (PK_COLOR), FARM INDEX (PK_FARM))

                COUNT
=====================
               521254

Elapsed time = 1.719 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 4237295
```

```
SELECT
  COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
WHERE HORSE.MICROCHIP = '643094100413542';

PLAN JOIN (HORSE INDEX (HORSE_IDX_MICROCHIP), COLOR
INDEX (PK_COLOR), FARM INDEX (PK_FARM))

                COUNT
=====================
                    1

Elapsed time = 0.001 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 11
```

ЧАСТИЧНЫЙ ИНДЕКС ДЛЯ ПОЛЯ С МНОЖЕСТВОМ NULL ЗНАЧЕНИЙ

```
SELECT
  COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
WHERE HORSE.MICROCHIP || '' IS NULL;

PLAN HASH (JOIN (COLOR NATURAL, HORSE INDEX (FK_HORSE_COLOR)), FARM NATURAL)

                COUNT
=====================
               521254

Elapsed time = 0.534 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 714470
```

ЧАСТИЧНЫЙ ИНДЕКС ДЛЯ ПОЛЯ С МНОЖЕСТВОМ NULL ЗНАЧЕНИЙ

```
-- CREATE INDEX HORSE_IDX_MICROCHIP ON HORSE (MICROCHIP);
DROP INDEX HORSE_IDX_MICROCHIP;
CREATE INDEX HORSE_IDX_MICROCHIP ON HORSE (MICROCHIP) WHERE MICROCHIP IS NOT NULL;
```

```
SELECT COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
WHERE HORSE.MICROCHIP IS NULL;

PLAN HASH (JOIN (COLOR NATURAL, HORSE INDEX (FK_HORSE_COLOR)),
FARM NATURAL)

                COUNT
=====================
               521254

Elapsed time = 0.507 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 714582
```

```
SELECT COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
WHERE HORSE.MICROCHIP = '643094100413542';

PLAN JOIN (HORSE INDEX (HORSE_IDX_MICROCHIP), COLOR INDEX (PK_COLOR),
FARM INDEX (PK_FARM))

                COUNT
=====================
                    1

Elapsed time = 0.001 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 11
```

Множество предикатов фильтрации

- Начиная с Firebird 5.0 учитываются не только селективность индексированных предикатов, но и не индексированных. Например для предиката = она равна 0.1
- Предикаты считаются некоррелированными между собой, поэтому конъюнкция (AND) предикатов имеет селективность равной произведению селективностей. На самом деле корреляция между значениями отдельных столбцов обычно есть.
- Применение множества предикатов для фильтрации одной таблицы приводит к недооценки кардинальности

Множество предикатов фильтрации объединённых AND

```
SELECT
  COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
  JOIN BREED ON BREED.CODE_BREED = HORSE.CODE_BREED
WHERE HORSE.BLOOD IS NULL
  AND HORSE.FAMILY IS NULL
  AND HORSE.BLOOD_EN IS NULL
  AND HORSE.REF_GPB_SUN IS NULL
  AND FARM.ZIP IS NOT NULL

Elapsed time = 1.039 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 2586111
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
BREED                           |         |  194819 |
COLOR                           |         |  194819 |
FARM                            |         |  194819 |
HORSE                           |  570122 |         |
--------------------------------+---------+---------+

570000 * 0.1 * 0.1 * 0.1 * 0.1 = 57 записей
```

```
PLAN JOIN (HORSE NATURAL, COLOR INDEX (PK_COLOR), BREED INDEX
(PK_BREED), FARM INDEX (PK_FARM))

Select Expression
    -> Aggregate
        -> Nested Loop Join (inner)
            -> Filter
                -> Table "HORSE" Full Scan
            -> Filter
                -> Table "COLOR" Access By ID
                    -> Bitmap
                        -> Index "PK_COLOR" Unique Scan
            -> Filter
                -> Table "BREED" Access By ID
                    -> Bitmap
                        -> Index "PK_BREED" Unique Scan
            -> Filter
                -> Table "FARM" Access By ID
                    -> Bitmap
                        -> Index "PK_FARM" Unique Scan

                  COUNT
=====================
                  40106
```

Множество предикатов фильтрации объединённых AND

```
SELECT
  COUNT(*)
FROM
  HORSE
  JOIN FARM ON FARM.CODE_FARM = HORSE.CODE_FARM
  JOIN COLOR ON COLOR.CODE_COLOR = HORSE.CODE_COLOR
  JOIN BREED ON BREED.CODE_BREED = HORSE.CODE_BREED
WHERE (HORSE.BLOOD IS NULL
  AND HORSE.FAMILY IS NULL
  AND HORSE.BLOOD_EN IS NULL
  AND HORSE.REF_GPB_SUN IS NULL OR FALSE)
  AND FARM.ZIP IS NOT NULL

Elapsed time = 0.465 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 714782
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
BREED                           |     294|         |
COLOR                           |     242|         |
FARM                            |   42298|         |
HORSE                           |        |  570122|
--------------------------------+---------+---------+

Selectivity(<pred> OR FALSE) = 0.5 -> 570000 * 0.5 = 235000
```

```
PLAN HASH (HASH (JOIN (COLOR NATURAL, HORSE INDEX (FK_HORSE_COLOR)), BREED
NATURAL), FARM NATURAL)

Select Expression
  -> Aggregate
      -> Filter
            -> Hash Join (inner)
                -> Hash Join (inner)
                    -> Nested Loop Join (inner)
                       -> Table "COLOR" Full Scan
                        -> Filter
                           -> Table "HORSE" Access By ID
                              -> Bitmap
                                 -> Index "FK_HORSE_COLOR" Range Scan (full match)
                    -> Record Buffer (record length: 25)
                        -> Table "BREED" Full Scan
                -> Record Buffer (record length: 81)
                    -> Filter
                        -> Table "FARM" Full Scan

                  COUNT
=====================
                  40106
```

OuterJoinConversion

- Появилось в Firebird 5.0
- Преобразует LEFT/RIGHT JOIN в INNER JOIN если по ведомой таблице, есть фильтр не учитывающий NULLs
- Развязывает руки оптимизатору: порядок соединений, разные алгоритмы соединения (Nested Loop vs Hash)
- Портит жизнь тем кто использовал LEFT JOIN в качестве подсказки для указания порядка соединения
- OuterJoinConversion = false выключает эту оптимизацию глобально

OuterJoinConversion=true

```
SELECT
  COUNT(*)
FROM
  TRIAL
  JOIN TRIAL_LINE TL ON TL.CODE_TRIAL = TRIAL.CODE_TRIAL
  LEFT JOIN HORSE H ON H.CODE_HORSE = TL.CODE_HORSE
WHERE H.CODE_DEPARTURE = 1
  AND H.CODE_BREED = 65
  AND TRIAL.CODE_DEPARTURE = 1
  AND TRIAL.BYDATE >= date '2005-01-01';

Elapsed time = 0.383 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 748340
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
HORSE                           |         |   100854|
TRIAL                           |         |    11423|
TRIAL_LINE                      |         |   185012|
--------------------------------+---------+---------+
```

OuterJoinConversion=true

```
PLAN HASH (JOIN (H INDEX (FK_HORSE_BREED, FK_HORSE_DEPARTURE), TL INDEX (FK_TRIAL_LINE_HORSE)), TRIAL INDEX (IDX_TRIAL_BYDATE,
FK_TRIAL_DEPARTURE))

Select Expression
    -> Aggregate
        -> Filter
            -> Hash Join (inner)
                -> Nested Loop Join (inner)
                    -> Filter
                        -> Table "HORSE" as "H" Access By ID
                            -> Bitmap And
                                -> Bitmap
                                    -> Index "FK_HORSE_BREED" Range Scan (full match)
                                -> Bitmap
                                    -> Index "FK_HORSE_DEPARTURE" Range Scan (full match)
                    -> Filter
                        -> Table "TRIAL_LINE" as "TL" Access By ID
                            -> Bitmap
                                -> Index "FK_TRIAL_LINE_HORSE" Range Scan (full match)
                -> Record Buffer (record length: 41)
                    -> Filter
                        -> Table "TRIAL" Access By ID
                            -> Bitmap And
                                -> Bitmap
                                    -> Index "IDX_TRIAL_BYDATE" Range Scan (lower bound: 1/1)
                                -> Bitmap
                                    -> Index "FK_TRIAL_DEPARTURE" Range Scan (full match)

                COUNT
=====================
                71560
```

OuterJoinConversion=true (переписываем запрос)

```
SELECT
  COUNT(*)
FROM
  TRIAL
  JOIN TRIAL_LINE TL ON TL.CODE_TRIAL = TRIAL.CODE_TRIAL
  LEFT JOIN HORSE H ON H.CODE_HORSE = TL.CODE_HORSE
                   AND H.CODE_DEPARTURE = 1 AND H.CODE_BREED = 65
WHERE H.CODE_HORSE IS NOT NULL
  AND TRIAL.CODE_DEPARTURE = 1
  AND TRIAL.BYDATE >= date '2005-01-01';

Elapsed time = 0.249 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 460043
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
HORSE                           |         |    71581|
TRIAL                           |         |    11423|
TRIAL_LINE                      |         |    71581|
--------------------------------+---------+---------+
```

OuterJoinConversion=true (переписанный запрос)

```
PLAN JOIN (JOIN (TRIAL INDEX (IDX_TRIAL_BYDATE, FK_TRIAL_DEPARTURE), TL INDEX (FK_TRIAL_LINE_TRIAL)), H INDEX (PK_HORSE))

Select Expression
    -> Aggregate
        -> Filter
            -> Nested Loop Join (outer)
                -> Nested Loop Join (inner)
                    -> Filter
                        -> Table "TRIAL" Access By ID
                            -> Bitmap And
                                -> Bitmap
                                    -> Index "IDX_TRIAL_BYDATE" Range Scan (lower bound: 1/1)
                                -> Bitmap
                                    -> Index "FK_TRIAL_DEPARTURE" Range Scan (full match)
                    -> Filter
                        -> Table "TRIAL_LINE" as "TL" Access By ID
                            -> Bitmap
                                -> Index "FK_TRIAL_LINE_TRIAL" Range Scan (full match)
                -> Filter
                    -> Table "HORSE" as "H" Access By ID
                        -> Bitmap
                            -> Index "PK_HORSE" Unique Scan

                COUNT
=====================
                71560
```

А если подумать, то и без LEFT JOIN будет хорошо

```
SELECT
  COUNT(*)
FROM
  TRIAL
  JOIN TRIAL_LINE TL ON TL.CODE_TRIAL = TRIAL.CODE_TRIAL
  JOIN HORSE H ON H.CODE_HORSE = TL.CODE_HORSE
WHERE H.CODE_DEPARTURE = 1
  AND H.CODE_BREED+0 = 65
  AND TRIAL.CODE_DEPARTURE = 1
  AND TRIAL.BYDATE >= date '2005-01-01';

PLAN JOIN (TRIAL INDEX (IDX_TRIAL_BYDATE, FK_TRIAL_DEPARTURE), TL INDEX (FK_TRIAL_LINE_TRIAL), H INDEX (PK_HORSE))

Elapsed time = 0.242 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 460043
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
HORSE                           |         |    71581|
TRIAL                           |         |    11423|
TRIAL_LINE                      |         |    71581|
--------------------------------+---------+---------+
```

OuterJoinConversion=true
Когда это плохо работает?

- Множество LEFT JOIN и в INNER JOIN превращается не первый
  - LEFT JOIN зависят друг от друга
  - LEFT JOIN не зависят друг от друга
- LEFT JOIN <table expr> (сложное представление, CTE, производная таблица)

INNER JOIN и LEFT JOIN с табличными выражениями работают по разному

- <table expr> A JOIN T ON T.Field1 = A.Field2
  - Если для Field1 нет индекса, то используется HASH JOIN. Порядок выбирается оптимизатором
  - Если для Field1 есть индекс, то табличное выражение всегда выбирается ведущим потоком и используется алгоритм Nested Loop
- T LEFT JOIN <table expr> A ON A.Field2 = T.Field1
  - Порядок потоков не может быть изменён
  - Всегда используется алгоритм Nested Loop
  - Табличное выражение выполняется многократно, но предикат соединения пробрасывается максимально глубоко

SubQueryConversion

- Появилось в Firebird 5.0.1 (экспериментальная функция)
- По умолчанию отключено (SubQueryConversion = false)
- Преобразует IN/EXISTS/ANY в semi-join
- Преобразование происходит, если все предикаты соединения являются равенством (=) или эквивалентностью (IS NOT DISTINCT FROM). И в подзапросе отсутствуют конструкции ломающие семантическую эквивалентность FIRST/SKIP/FETCH FIRST/OFFSET/ROWS/
- В Firebird 5.0.4 доведено до ума (портировано из 6.0)
- Semi-join может быть выполнен как Hash semi join или Nested Loop semi join (стоимостная оценка)
- До Firebird 5.0.4 мог выполняться только как Hash semi

SubQueryConversion: как работает преобразование в semi-join?

```
SELECT
  COUNT(*) AS CNT
FROM
  T1
WHERE EXISTS(
  SELECT * FROM T2
  WHERE T2.FIELD2 = T1.FIELD1
);

-- или

SELECT
  COUNT(*) AS CNT
FROM
  T1
WHERE T1.FIELD1 IN (
  SELECT T2.FIELD2 FROM T2
);
```

```
-- псевдо-код
SELECT
  SUM(*) AS CNT
FROM
  T1
  SEMI JOIN (
    SELECT T2.FIELD2 FROM T2
  ) T ON T.FIELD2 = T1.FIELD1;

-- Никакого SEMI в SQL не существует!!!
```

SubQueryConversion=false

```
SELECT COUNT(*)
FROM HORSE
WHERE EXISTS(
        SELECT * FROM COVER
        WHERE COVER.CODE_MOTHER = HORSE.CODE_HORSE
          AND EXTRACT(YEAR FROM COVER.RESULTDATE) = 2000
          AND COVER.CODE_MOTHER > 0
      )
  AND HORSE.CODE_SEX = 1;

Elapsed time = 30.657 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 2648094
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |    12430|
HORSE                           |         |   324416|
--------------------------------+---------+---------+
```

SubQueryConversion=false

```
Sub-query
    -> Filter (preliminary)
        -> Filter
            -> Table "COVER" Access By ID
                -> Bitmap And
                    -> Bitmap
                        -> Index "FK_COVER_MOTHER" Range Scan (full match)
                    -> Bitmap
                        -> Index "IDX_COVER_RESULTYEAR" Range Scan (full match)
Select Expression
    -> Aggregate
        -> Filter
            -> Table "HORSE" Access By ID
                -> Bitmap
                    -> Index "FK_HORSE_SEX" Range Scan (full match)

                COUNT
======================
                12430
```

SubQueryConversion=true

```
SELECT COUNT(*)
FROM HORSE
WHERE EXISTS(
        SELECT * FROM COVER
        WHERE COVER.CODE_MOTHER = HORSE.CODE_HORSE
          AND EXTRACT(YEAR FROM COVER.RESULTDATE) = 2000
          AND COVER.CODE_MOTHER > 0
      )
  AND HORSE.CODE_SEX = 1;

Elapsed time = 0.229 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 350498
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |    13080|
HORSE                           |         |   324416|
--------------------------------+---------+---------+
```

SubQueryConversion=true

```
Select Expression
    -> Aggregate
        -> Filter
            -> Hash Join (semi)
                -> Filter
                    -> Table "HORSE" Access By ID
                        -> Bitmap
                            -> Index "FK_HORSE_SEX" Range Scan (full match)
                -> Record Buffer (record length: 41)
                    -> Filter
                        -> Table "COVER" Access By ID
                            -> Bitmap And
                                -> Bitmap
                                    -> Index "IDX_COVER_RESULTYEAR" Range Scan (full match)
                                -> Bitmap
                                    -> Index "FK_COVER_MOTHER" Range Scan (lower bound: 1/1)

                COUNT
======================
                12430
```

Проблемы SubQueryConversion

- Выбор Hash Semi Join вместо Nested Loop Semi Join
- Выбор Nested Loop Join вместо Hash Semi Join
- Альтернатива JOIN (SELECT DISTINCT ...) может быть быстрее

Важно:

- Nested Loop Semi Join почти ничем не отличается от выполнения коррелированного подзапроса

SubQueryConversion=true. Hash вместо Nested Loop

```
SELECT COUNT(*)
FROM COLOR
WHERE EXISTS(
        SELECT * FROM HORSE
        WHERE HORSE.MICROCHIP IS NULL
          AND HORSE.CODE_COLOR = COLOR.CODE_COLOR
      );

Elapsed time = 0.305 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 608692
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COLOR                           |     242|         |
HORSE                           |  570122|         |
--------------------------------+---------+---------+
```

```
Select Expression
    -> Aggregate
        -> Filter
            -> Hash Join (semi)
                -> Table "COLOR" Full Scan
                -> Record Buffer (record length: 129)
                    -> Filter
                        -> Table "HORSE" Full Scan

                     COUNT
======================
                       222
```

SubQueryConversion=true. Hash вместо Nested Loop. Делаем преобразование в semi-join невозможным.

```
SELECT COUNT(*)
FROM COLOR
WHERE EXISTS(
        SELECT * FROM HORSE
        WHERE HORSE.MICROCHIP IS NULL
          AND HORSE.CODE_COLOR = COLOR.CODE_COLOR
        FETCH FIRST ROW ONLY
      );

Elapsed time = 0.020 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 1641
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COLOR                           |     242|         |
HORSE                           |         |     268|
--------------------------------+---------+---------+
```

```
Sub-query
    -> First N Records
        -> Filter
            -> Table "HORSE" Access By ID
                -> Bitmap
                    -> Index "FK_HORSE_COLOR" Range Scan (full match)
Select Expression
    -> Aggregate
        -> Filter
            -> Table "COLOR" Full Scan

                     COUNT
======================
                       222
```

НЕ ДЕЛАЙТЕ ТАК!!!

```
SELECT COUNT(*)
FROM COLOR
WHERE COLOR.CODE_COLOR IN (
        SELECT HORSE.CODE_COLOR FROM HORSE
        WHERE HORSE.MICROCHIP IS NULL
);

-- Не эквивалентно

SELECT COUNT(*)
FROM COLOR
WHERE COLOR.CODE_COLOR IN (
        SELECT HORSE.CODE_COLOR FROM HORSE
        WHERE HORSE.MICROCHIP IS NULL
        FETCH FIRST ROW ONLY
);
```

SubQueryConversion=true. Nested Loop вместо Hash

```
SELECT COUNT(*)
FROM HORSE
WHERE EXISTS(
        SELECT * FROM COVER
        WHERE COVER.CODE_MOTHER = HORSE.CODE_HORSE
          AND EXTRACT(YEAR FROM COVER.RESULTDATE) = 2000
          AND COVER.CODE_MOTHER > 0
      )
  AND HORSE.CODE_DEPARTURE = 1;

Elapsed time = 6.475 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 699238
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |    2026|
HORSE                           |         |  101058|
--------------------------------+---------+---------+
```

```
Select Expression
    -> Aggregate
        -> Nested Loop Join (semi)
            -> Filter
                -> Table "HORSE" Access By ID
                    -> Bitmap
                        -> Index "FK_HORSE_DEPARTURE" Range Scan (full match)
            -> Filter
                -> Table "COVER" Access By ID
                    -> Bitmap And
                        -> Bitmap
                            -> Index "FK_COVER_MOTHER" Range Scan (full match)
                        -> Bitmap
                            -> Index "IDX_COVER_RESULTYEAR" Range Scan (full match)

                     COUNT
=====================
                      2026
```

SubQueryConversion=true.
Nested Loop вместо Hash. Решение.

```
SELECT COUNT(*)
FROM HORSE
WHERE EXISTS(
        SELECT * FROM COVER
        WHERE COVER.CODE_MOTHER+0 = HORSE.CODE_HORSE
          AND EXTRACT(YEAR FROM COVER.RESULTDATE) = 2000
          AND COVER.CODE_MOTHER > 0
      )
  AND HORSE.CODE_DEPARTURE = 1;

Elapsed time = 0.117 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 123374
Per table statistics:
--------------------------------+---------+---------
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |   13080|
HORSE                           |         |  101058|
--------------------------------+---------+---------+
```

```
Select Expression
    -> Aggregate
        -> Filter
            -> Hash Join (semi)
                -> Filter
                    -> Table "HORSE" Access By ID
                        -> Bitmap
                            -> Index "FK_HORSE_DEPARTURE" Range Scan (full
match)
                -> Record Buffer (record length: 41)
                    -> Filter
                        -> Table "COVER" Access By ID
                            -> Bitmap And
                                -> Bitmap
                                    -> Index "IDX_COVER_RESULTYEAR" Range Scan
(full match)
                                -> Bitmap
                                    -> Index "FK_COVER_MOTHER" Range Scan
(lower bound: 1/1)

                  COUNT
=====================
                   2026
```

Альтернатива IN/EXISTS

```
SELECT COUNT(*)
FROM
  HORSE
  JOIN (
    SELECT DISTINCT
      COVER.CODE_MOTHER AS CODE_HORSE
    FROM COVER
    WHERE EXTRACT(YEAR FROM COVER.RESULTDATE) = 2000
      AND COVER.CODE_MOTHER > 0
  ) T ON T.CODE_HORSE = HORSE.CODE_HORSE
WHERE HORSE.CODE_DEPARTURE = 1;

Elapsed time = 0.085 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 70842
Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |    13080|
HORSE                           |         |    12430|
--------------------------------+---------+---------+
```

```
Select Expression
    -> Aggregate
        -> Nested Loop Join (inner)
            -> Unique Sort (record length: 44, key length: 12)
                -> Filter
                    -> Table "COVER" as "T COVER" Access By ID
                        -> Bitmap And
                            -> Bitmap
                                -> Index "IDX_COVER_RESULTYEAR" Range Scan
(full match)
                            -> Bitmap
                                -> Index "FK_COVER_MOTHER" Range Scan (lower
bound: 1/1)
            -> Filter
                -> Table "HORSE" Access By ID
                    -> Bitmap
                        -> Index "PK_HORSE" Unique Scan

                COUNT
=====================
                 2026
```

Широкие GROUP BY

FIREBIRD CONF'26

- GROUP BY может выполняться алгоритмами ORDER и SORT
- Возникают когда много полей включено в предложение GROUP BY и эти поля не входят в один индекс
- Наихудший случай, если встречаются поля с типами VARCHAR(N), где N велико
- Провоцируют широкий внешние сортировки (SORT в Legacy плане)
- В отличие от широких сортировок в ORDER BY, не могут быть оптимизированы с помощью алгоритма Refetch

Вычисление агрегатов с широкими группировками

```
SELECT
  FARM.CODE_FARM,
  FARM.NAME,
  FARM.NAME_EN,
  FARM.ADDRESS,
  COUNT(*) AS CNT
FROM
  COVER
  JOIN FARM ON FARM.CODE_FARM = COVER.CODE_FARM
GROUP BY 1, 2, 3, 4;

Select Expression
    -> Aggregate
        -> Sort (record length: 2686, key length: 1828)
            -> Filter
                -> Hash Join (inner)
                    -> Table "COVER" Full Scan
                    -> Record Buffer (record length: 1441)
                        -> Table "FARM" Full Scan
```

Вычисление агрегатов с широкими группировками

```
Current memory = 554646368
Delta memory = 288
Max memory = 1092118992
Elapsed time = 11.670 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 846691

Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |  767959|         |
FARM                            |   41356|         |
--------------------------------+---------+---------+
```

Вычисление агрегатов с широкими группировками

- Оставляем в **GROUP BY** по которым можно уникально идентифицировать сущность. Для остальных **MIN/MAX**
- В Firebird 6.0 используем новую агрегатную функцию **ANY_VALUE**

```
SELECT
  FARM.CODE_FARM AS CODE_FARM,
  MIN(FARM.NAME) AS NAME,
  MIN(FARM.NAME_EN) AS NAME_EN,
  MIN(FARM.ADDRESS) AS ADDRESS,
  COUNT(*) AS CNT
FROM
  COVER
  JOIN FARM ON FARM.CODE_FARM = COVER.CODE_FARM
GROUP BY 1;
```

```
Select Expression
    -> Aggregate
        -> Filter
            -> Hash Join (inner)
                -> Table "COVER" Access By ID
                    -> Index "FK_COVER_FARM" Full Scan
                -> Record Buffer (record length: 1441)
                    -> Table "FARM" Full Scan
```

Вычисление агрегатов с широкими группировками

```
Current memory = 555886896
Delta memory = 320
Max memory = 1092118992
Elapsed time = 1.456 sec
Buffers = 32768
Reads = 0
Writes = 0
Fetches = 1745358

Per table statistics:
--------------------------------+---------+---------+
 Table name                     | Natural | Index   |
--------------------------------+---------+---------+
COVER                           |         |  767959 |
FARM                            |   41356 |         |
--------------------------------+---------+---------+
```

Новое в оптимизаторе Firebird 6.0

Уже реализовано

- Стоимостная оценка между ORDER index и SORT
- Преобразование IN/EXISTS в semi-join (5.0.1) и стоимостная оценка между Hash Semi Join и Nested Loop Semi Join (портировано в 5.0.4)
- Пропуск NULL при навигации по индексу (ORDER) с границами
- Улучшенная оценка селективности конъюнктов (AND). Алгоритм exponential back-off
- Детерминистические функции с параметрами инвариантами – являются инвариантами

Стоимостная оценка между ORDER и SORT

```
SELECT
  HORSE.CODE_HORSE,
  HORSE.NAME
FROM
  HORSE
WHERE HORSE.BRAND = ''
ORDER BY HORSE.NAME;
```

```
SELECT
  COVER.CODE_FATHER,
  COUNT(*) AS CNT
FROM
  COVER
WHERE COVER.IS_ARTIFICIAL IS TRUE
GROUP BY 1;
```

Алгоритм exponential back-off

- Для таблицы есть фильтр вида P1 AND P2 AND P3 … PN, где P - предикат фильтрации.
- До Firebird 6.0 предикаты считались не коррелированными, хотя обычно это не так. Формула суммарной селективности была такой:

$$S = S(P_1) * S(P_2) * \ \ldots S(P_n\ )$$

- Начиная с Firebird 6.0 предикаты считаются частично коррелированными.
  - Производится сортировка селективностей от меньшей к большей (от лучшей к худшей)

$$S = S_1\ * S_2^{\frac{1}{2}} * \ S_3^{\frac{1}{4}}\ \ldots * \ S_n^{\frac{1}{2^{n-1}}}$$

Детерминистические функции

- До Firebird 6.0 маркировка функции как **DETERMINISTIC** имела смысл только для функций без входных параметров;
- Начиная с Firebird 6.0 имеет смысл создавать детерминистические функции с параметрами, если в них передаются инварианты
- Инвариант – значение которое не меняется в течении выполнения запроса, то есть литерал, или входной параметр.

```
SELECT *
FROM T
WHERE T.NAME = FIRST_UPPER(?);
-- Оптимизируется как
X = FIRST_UPPER(?);
SELECT *
FROM T
WHERE T.NAME = :X;
```

Каких методов доступа не хватает

- Skip index scan
- Only index scan
- Hash/Merge Outer Join
- Hash group by
- Параллельный full table scan
- Материализация CTE при многократном использовании
- Трансформации запроса
  - elimination join (устранение лишних JOIN на основе ограничений)
  - перемещение outer join после inner join

Другие проблемы

- Глубокая стоимостная оценка сложных запросов
- Учёт текущих ограничений из конфига (страничный кеш ограничен, размер temp в памяти ограничен)
- Нет некоторых run-time статистик: например статистики использования индексов (обращений, какое сканирование, какие задействованы сегменты)


FIREBIRD CONF'26
Организатор конференции
РЕДСОФТ
Спасибо за внимание!