

Выполнение запросов

План выполнения запроса

Как сервер выполняет запрос

- Синтаксический анализ, преобразование запроса в формальное выражение, например в выражение реляционной алгебры (в виде дерева) – логический план
- Преобразование в каноническую форму на основе законов преобразования
- Оптимизация– физический план
 - Логическая оптимизация
 - Выбор алгоритмов и оптимизация по стоимости
 - Выбор низкоуровневых процедур на основе оценки стоимости
 - Генерация различных вариантов плана выполнения запроса и выбор плана с минимальными затратами
- Интерпретация плана и пересылка результата

Как сервер выполняет запрос

Что используется

- Статистические данные (Системный каталог)

Преимущества

- Многие можно не учитывать при написании запроса
- Оптимизация «здесь» и «сейчас» автоматическая (высокоуровневая)
- Анализ большого числа альтернатив

Проблемы

- Оптимальность не гарантируется
- Всегда выполняется (требует явного отключения в запросе)

Сравнение двух способов

```
select name_ag  
  from agent A join operation O  
    on (A.id_ag = O.id_ag)  
 where O.id_goods = 'T1'
```

100 поставщиков

10 000 операций

50 с товаром T1

Where и Join

1

- соединение
10 000 * 100 чтений
10 000 записей
- селекция
10 000 чтений, выборка 50 записей
- проекция
не более 50 записей

2

- селекция
10 000 чтений, выборка 50 записей
- соединение
50 * не более 50 записей
- проекция
не более 50 записей

План выполнения запроса

План выполнения запроса

- План выполнения запроса — последовательность операций, необходимых для получения результата SQL-операции в реляционной СУБД
- Оценка плана для сравнения и выбора оптимального
 - Кардинальность (количество затрагиваемых строк)
 - Стоимость (стоимость алгоритма доступа)

План запроса (синтаксис)

PLAN <выражение>

<выражение> ::= [JOIN | [SORT] [MERGE]] (<элемент> | <выражение>
[, <элемент> | <выражение> ...])

<элемент> ::= {таблицы | псевдоним}

{NATURAL

| INDEX (индекс [, индекс ...])

| ORDER индекс [INDEX (индекс [, индекс ...])])}]

PLAN JOIN (O INDEX (FK_OP_1), A INDEX (PK_AGENT))

План в EXPLAIN форме:

Select Expression

-> Nested Loop Join (inner)

-> Filter

-> Table "OPERATION" as "O" Access By ID

-> Bitmap

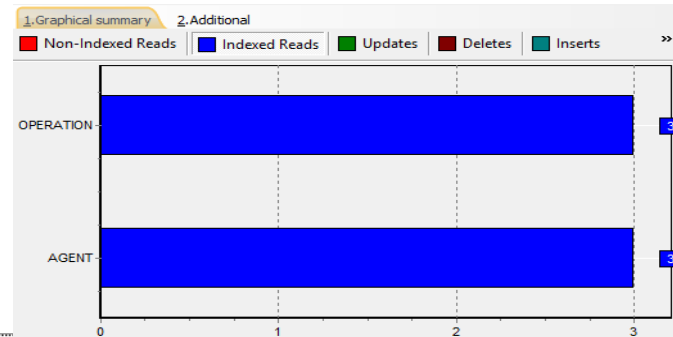
-> Index "FK_OP_1" Range Scan (full match)

-> Filter

-> Table "AGENT" as "A" Access By ID

-> Bitmap

-> Index "PK_AGENT" Unique Scan



PLAN JOIN	OPERATION	AGENT	ID_GOODS	FK
O INDEX (FK_OP_1)	OPERATION		0,166666671633...	FK
FK_OP_1				
A INDEX (PK_AGENT)	AGENT		0,142857149243...	PK
PK_AGENT				

Путь доступа

- Набор операций над данными, выполняемых сервером для получения результата заданной выборки, называется **путем доступа**.
- Путь доступа можно представить в виде дерева с корнем, представляющим собой конечный результат.
- Каждый узел этого дерева называется **источником данных** или **методом доступа** .
- Объектами операций в методах доступа являются **потоки данных**. Каждый метод доступа либо формирует поток данных, либо трансформирует его по определенным правилам.
- Листовые узлы дерева называются **первичными методами доступа**. Их единственная задача – формирование потоков данных.

Источники данных

- Существует три класса источников данных:
 - первичный метод доступа – выполняет чтение из таблицы или хранимой процедуры
 - фильтр – трансформирует один входной поток данных в один выходной поток
 - слияние – преобразует два или более входных потоков данных в один выходной поток
- Источники данных могут быть конвейерными и буферизированными

Первичные методы доступа

- Создание потока данных на основе таблиц и процедур
 - Чтение таблиц
 - Индексный доступ
 - Выборка из процедуры

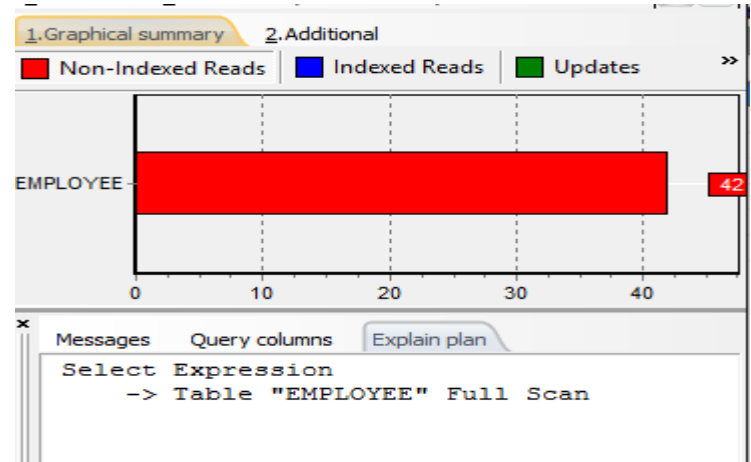
Чтение таблицы

- Полное сканирование (sequential scan)

```
select *  
FROM employee;
```

Plan
PLAN (EMPLOYEE NATURAL)

Кардинальное число = 42
стоимость =42



Чтение таблицы

- Доступ через идентификатор записи

Физический номер записи содержит информацию о странице, на которой расположена данная запись, и о смещении внутри этой страницы

Стоимость данного вида доступа всегда равна единице. Чтения записей отражаются в статистике как индексированные.

```
C:\Program Files\Firebird\Firebird_2_5\bin\isql.exe
SQL> connect localhost:d:\marina\ktr.fdb user sysdba password masterkey;
Commit current transaction (y/n)?y
Committing.
Database: localhost:d:\marina\ktr.fdb, User: sysdba
SQL> select rdb$db_key from operation;

DB_KEY
=====
8200000001000000
8200000002000000
8200000003000000
8200000004000000
8200000005000000
8200000006000000
8200000007000000
8200000008000000
8200000009000000
820000000A000000
820000000B000000
820000000C000000
820000000D000000
820000000E000000
820000000F000000

SQL>
```

Чтение таблицы

- Позиционированный доступ

команды типа UPDATE и DELETE

(WHERE CURRENT OF <cursor name>)

Позиционированный доступ работает только для активного курсора, т.е. для уже прочитанной записи

Индексный доступ

- Индексный доступ (index scan/ index only scan)
 - Основным параметром, влияющим на оптимизацию индексного доступа, является **селективность** индекса
 - В расчетах кардинальности и стоимости предполагается равномерное распределение значений ключа в индексе

Индексный доступ

- Для кластерных индексов возможно использование чтения только индекса
- Если индекс не кластерный, требуется чтение и страниц индекса, и страниц данных
- В Firebird основан на битовых картах и учитывает версии
- В Firebird сканирование индекса - однонаправленное

Индексный доступ

- Стоимость доступа через битовую карту – суммарная стоимость индексного поиска для всех индексов, формирующих битовую карту
- Кардинальное число при пересечении битовых карт не больше минимального, при объединении – не меньше максимального

Индексный доступ

- При выборе индексов для сканирования оптимизатор использует стратегию на основе стоимости (cost based)
- Стоимость сканирования диапазона оценивается на основании селективности индекса, количества записей в таблице, среднего количества ключей на индексной странице и высоты B+ дерева.

Сканирование на равенство для уникального индекса (unique scan)

- `select * from employee
 where emp_no=15`

Plan

`PLAN (EMPLOYEE INDEX (RDB$PRIMARY7))`

Select Expression

-> Filter

-> Table "EMPLOYEE" Access By ID

-> Bitmap

-> Index "RDB\$PRIMARY7" Unique Scan

SQL Editor : 1 : localhost:C:\Program Files\Firebird\Firebird_2_5\examples\em...

localhost:C:\Program Files\Firebird\Firebird_2_5\examples\empbuild\EMPLOYEE.FDB

Edit: 1 Results History Plan Analyzer Performance Analysis Logs

PLAN (EMPLOYEE INDEX (RDB\$PRIMARY7))

Recompute selectivity

	Table	Index fields	Statistics	PK/FK
PLAN				
EMPLOYEE INDEX (RDB\$PRIMARY7)	EMPLOYEE			
RDB\$PRIMARY7	EMPLOYEE	EMP_NO	0,0238095242...	

Messages Query columns

Plan
PLAN (EMPLOYEE INDEX (RDB\$PRIMARY7))

Adapted Plan

1. Graphical summary 2. Additional

Non-Indexed Reads Indexed Reads Updates

EMPLOYEE

0 1

Messages Query columns Explain plan

```
Select Expression
-> Filter
    -> Table "EMPLOYEE" Access By ID
        -> Bitmap
            -> Index "RDB$PRIMARY7"

Unique Scan
```

Сканирование диапазона (range scan)

- select * from employee
where emp_no between 100 and 300

Plan

PLAN (EMPLOYEE INDEX (RDB\$PRIMARY7))

Select Expression

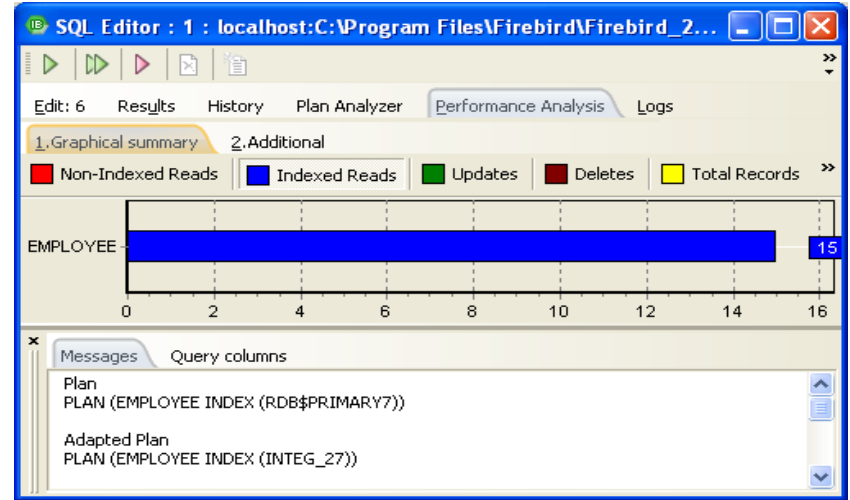
-> Filter

-> Table "EMPLOYEE" Access By ID

-> Bitmap

-> Index "RDB\$PRIMARY7" Range Scan (lower bound: 1/1, upper bound: 1/1)

Кардинальное число =13



Полное сканирование (full scan)

```
select * from employee
  where phone_ext=250
PLAN (EMPLOYEE NATURAL)
```

Select Expression

-> Filter

-> Table "EMPLOYEE" Full Scan

Наложение битовых карт

Тестовая таблица

```
CREATE TABLE PRIME2 (  
    ID  INTEGER NOT NULL,  
    IDX1 INTEGER,  
    IDX2 INTEGER  
);  
ALTER TABLE PRIME2  
    ADD CONSTRAINT PK_PRIME2 PRIMARY KEY (ID);  
  
CREATE INDEX PRIME2_IDX1 ON PRIME2 (IDX1);  
CREATE INDEX PRIME2_IDX2 ON PRIME2 (IDX2);
```

Наложение битовых карт

```
select * from prime2
      where idx1=14 AND idx2=5
```

```
PLAN (PRIME2 INDEX (PRIME2_IDX1, PRIME2_IDX2))
```

```
STATEMENT (SELECT)
```

```
[cardinality=2, cost=9.000]
```

```
=> TABLE (PRIME2) ACCESS BY DB_KEY
```

```
[cardinality=2, cost=9.000]
```

```
=> BITMAP AND
```

```
[cardinality=2, cost=7.000]
```

```
=> INDEX (PRIME2_IDX1) RANGE SCAN
```

```
[cardinality=5, cost=4.000]
```

```
=> INDEX (PRIME2_IDX2) RANGE SCAN
```

```
[cardinality=2, cost=3.000]
```

Навигация по индексу

Используется тогда, когда индекс может помочь в упорядочении результата

```
select * from employee  
order by emp_no
```

```
PLAN (EMPLOYEE ORDER  
RDB$PRIMARY7)
```

Select Expression

- > Table "EMPLOYEE" Access By ID
- > Index "RDB\$PRIMARY7" Full Scan

```
select * from employee  
order by emp_no desc
```

```
PLAN SORT (EMPLOYEE NATURAL)
```

Select Expression

- > Sort (record length: 140, key length: 8)
- > Table "EMPLOYEE" Full Scan

Процедурный доступ

- Используется при выборке из хранимых процедур, использующих предложение SUSPEND для возврата результата
 - Индексация результатов невозможна
 - Аналог полного сканирования таблицы

```
SELECT * FROM DEPT_BUDGET('100')
```

```
PLAN (DEPT_BUDGET NATURAL)
```

Фильтры

- Преобразователи данных
- Имеют только один вход
- Изменяют кардинальное число
- Для фильтров не оценивается кардинальность и стоимость

Проверка предикатов

- WHERE, HAVING, IN, ALL, ANY
- В плане выполнения проверка предикатов не отображается

Сортировка

- ORDER BY, GROUP BY, DISTINCT
- построение B+ дерева индексов
- подготовка данных для однопроходного слияния

Алгоритм сортировки представляет собой многоуровневую быструю сортировку

```
select * from employee  
order by salary
```

```
PLAN SORT ((EMPLOYEE NATURAL))
```

Select Expression

- > Sort (record length: 132, key length: 12)

- > Table "EMPLOYEE" Full Scan

Методы слияния

- всегда оперирует с несколькими входными потоками
- обычным результатом их работы является либо расширение выборки по полям, либо увеличение ее кардинальности
- соединение (join)
- объединение (union)

Соединение (join)

- У любого вида соединений есть два входных потока - левый и правый.
- Для внутреннего и полного внешнего соединений эти потоки семантически равноценны.
- В случае одностороннего внешнего соединения один из потоков является ведущим (обязательным), а второй - ведомым (необязательным).

Соединение (join)

- У каждого соединения помимо входных потоков существует еще один атрибут - условие связи. Именно это условие и определяет результат, то есть как именно будут поставлены в соответствие данные входных потоков.
- При отсутствии данного условия получаем вырожденный случай - декартово произведение (cross join) входных потоков.

Правила выбора потока

- Для односторонних внешних соединений внешний (ведущий) поток всегда должен быть прочитан перед внутренним (ведомым), иначе невозможно будет выполнить требуемую стандартом подстановку NULL-значений в случае отсутствия соответствий внутреннего потока внешнему.

Правила выбора потока

- Для внутренних и полных внешних соединений входные потоки независимы и могут читаться в произвольном порядке, следовательно алгоритм выполнения таких соединений определяется исключительно оптимизатором и никак не зависит от текста запроса.

Методы слияния

- Рекурсивный перебор или соединение посредством вложенных циклов (nested loops join)
- Однопроходное слияние (merge join)
- Хеширование (hash join)

Рекурсивный перебор

```
select *
```

```
from department d left join employee e on d.mngr_no=e.emp_no
```

```
PLAN JOIN (D NATURAL, E INDEX (RDB$PRIMARY7))
```

Select Expression

- > **Nested Loop Join (outer)**

- > Table "DEPARTMENT" as "D" Full Scan

- > Filter

- > Table "EMPLOYEE" as "E" Access By ID

- > **Bitmap**

- > Index "RDB\$PRIMARY7" Unique Scan

Рекурсивный перебор

```
select *  
from department d  
  join employee e  on  
    d.mngr_no=e.emp_no  
  join country c  on  
    c.country=e.job_country
```

```
PLAN JOIN (D NATURAL, E INDEX (RDB$PRIMARY7), C  
INDEX (RDB$PRIMARY1))
```

Select Expression

-> **Nested Loop Join (inner)**

-> Table "DEPARTMENT" as "D" Full Scan

-> Filter

-> Table "EMPLOYEE" as "E" Access By ID

-> **Bitmap**

-> Index "RDB\$PRIMARY7" Unique Scan

-> Filter

-> Table "COUNTRY" as "C" Access By ID

-> **Bitmap**

-> Index "RDB\$PRIMARY1" Unique Scan

Однопроходное слияние

- Оптимизатор выбирает данный алгоритм соединения только в случае невозможности или неоптимальности использования рекурсивного алгоритма, то есть в первую очередь при отсутствии индексов по условию связи или их неприменимости, а также при отсутствии зависимости между входными потоками.

```
select *  
from department d  
      join employee e on d.mngr_no+0=e.emp_no+0
```

```
PLAN MERGE (SORT (E NATURAL), SORT (D NATURAL))
```

Select Expression

-> Filter

-> Hash Join (inner)

-> Table "EMPLOYEE" as "E" Full Scan

-> Record Buffer (record length: 105)

-> Table "DEPARTMENT" as "D" Full Scan

Объединение (union)

- Существует два режима выполнения этой операции: ALL и DISTINCT.
- В первом случае реализация тривиальна: данный метод просто читает первый входной поток и выдает его на выход, по получении из него EOF начинает читать второй входной поток и так далее.

Объединение (union)

- В случае же DISTINCT требуется устранить полные дубликаты записей, присутствующие в результате объединения. Для этого на выходе метода объединения размещается фильтр сортировки, работающий в "усекающем" режиме по всем полям.

Объединение (union)

- Стоимость выполнения объединения равна суммарной стоимости всех входных потоков, кардинальность также получается суммированием.

```
select job_country
from employee
union
select country
from customer
```

```
PLAN (EMPLOYEE NATURAL)
PLAN (CUSTOMER NATURAL)
```

Select Expression

- > Unique Sort (record length: 62, key length: 24)
- > Union
 - > Table "EMPLOYEE" Full Scan
 - > Table "CUSTOMER" Full Scan

```
select job_country
from employee
union all
select country
from customer
```

```
PLAN (EMPLOYEE NATURAL)
PLAN (CUSTOMER NATURAL)
```

Select Expression

- > Union
 - > Table "EMPLOYEE" Full Scan
 - > Table "CUSTOMER" Full Scan

Включение плана в запрос

```
SELECT [DISTINCT | ALL]
      [FIRST <record_number> }|SKIP <record_number> ]
<select_list>
FROM <reference_expression_list>
[ WHERE <search condition> ]
[ GROUP BY <group_value_list> ]
[ HAVING <group_condition> ]
[ PLAN <plan_item_list> ]
```

PLAN ({ <stream_retrieval> |
 <sorted_streams> |
 <joined_streams> })

<stream_retrieval> ::= { <natural_scan> | <indexed_retrieval> |
 <navigational_scan> }

<natural_scan> ::= <stream_alias> **NATURAL**

<indexed_retrieval> ::= <stream_alias> **INDEX** (<index_name> [,
 <index_name> ...])

<navigational_scan> ::= <stream_alias> **ORDER** <index_name> [**INDEX** (
 <index_name> [, <index_name> ...])]

<sorted_streams> ::= **SORT** (<stream_retrieval>)

<joined_streams> ::= **JOIN** (<stream_retrieval>, <stream_retrieval> [,
 <stream_retrieval> ...]) | [**SORT**] **MERGE** (<sorted_streams>,
 <sorted_streams>)