OPTIMIZATION OF SQL QUERIES IN FIREBIRD

Dmitry Yemanov, Firebird

Alexey Kovyazin, IBSurgeon

Firebird Conference 2019

Berlin, 17-19 October

IBSurgeon

PART 1: EXPLAINED PLANS

Explained plans in Firebird 3

- Run ISQL

- To see plans in the old format

set plan on;

- To see new plans:

set explain;

Full Scan (ex NATURAL)

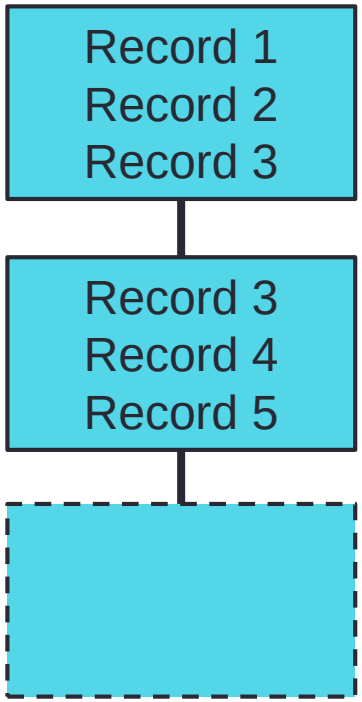
select * from employee

PLAN (EMPLOYEE NATURAL)

Select Expression

-> Table "T1" Full Scan

- The fastest way to read records



Old and new plans

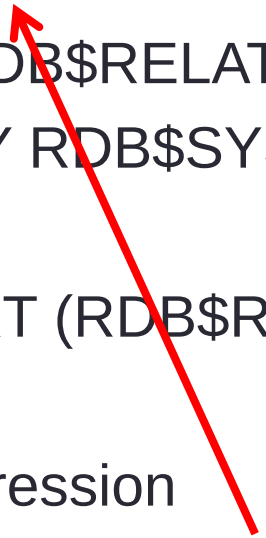
```
SELECT * FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME > :a
ORDER BY RDB$SYSTEM_FLAG
```

PLAN SORT (RDB$RELATIONS INDEX (RDB$INDEX_0))

```
Select Expression
    -> Sort (record length: 484, key length: 8)
        -> Filter
            -> Table "RDB$RELATIONS" Access By ID
                -> Bitmap
                    -> Index "RDB$INDEX_0" Range Scan (lower bound: 1/1)
```

Old and new plans

```
SELECT * FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME > :a
ORDER BY RDB$SYSTEM_FLAG

PLAN SORT (RDB$RELATIONS INDEX (RDB$INDEX_0))

Select Expression
    -> Sort (record length: 484, key length: 8)
        -> Filter
            -> Table "RDB$RELATIONS" Access By ID
                -> Bitmap
                    -> Index "RDB$INDEX_0" Range Scan (lower bound: 1/1)
```



Old and new plans

- SELECT * FROM RDB$RELATIONS
- WHERE RDB$RELATION_NAME > :a
- **ORDER BY RDB$SYSTEM_FLAG**

- PLAN SORT (RDB$RELATIONS INDEX (RDB$INDEX_0))

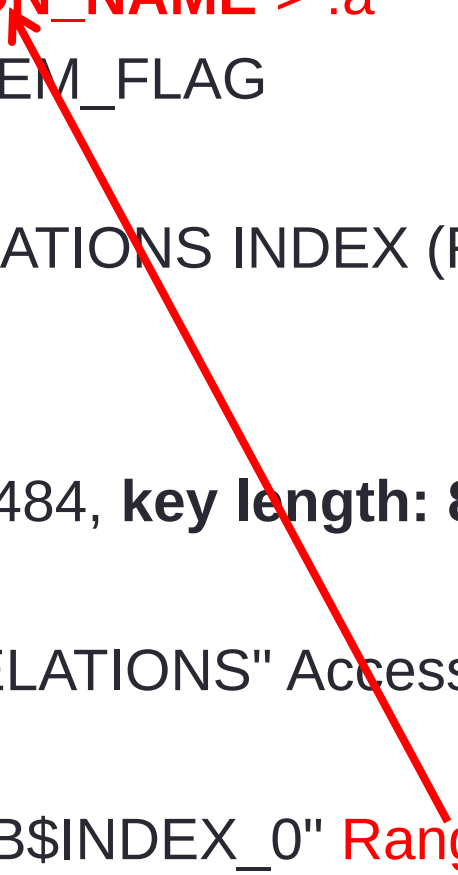
Select Expression
    -> **Sort** (record length: 484, **key length: 8**)
        -> Filter
            -> Table "RDB$RELATIONS" Access By ID
                -> Bitmap
                    -> Index "RDB$INDEX_0" Range Scan (lower bound: 1/1)

Old and new plans

- SELECT * FROM RDB$RELATIONS
- WHERE **RDB$RELATION_NAME** > :a
- ORDER BY RDB$SYSTEM_FLAG

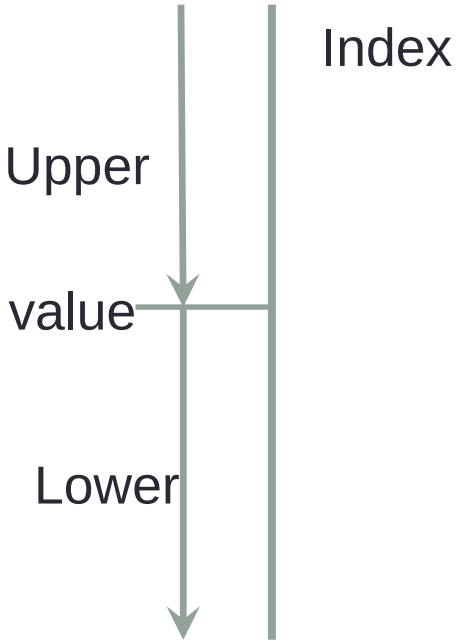
- PLAN SORT (RDB$RELATIONS INDEX (RDB$INDEX_0))

Select Expression
    -> **Sort** (record length: 484, **key length: 8**)
        -> Filter
            -> Table "RDB$RELATIONS" Access By ID
                -> Bitmap
                    -> Index "RDB$INDEX_0" Range Scan (lower bound: 1/1)



Index Range Scan options

- Lower bound - >, >=
- Upper bound - <, <=
- Full scan - <>
- Unique scan - =
  - For unique indices
- Full match - =



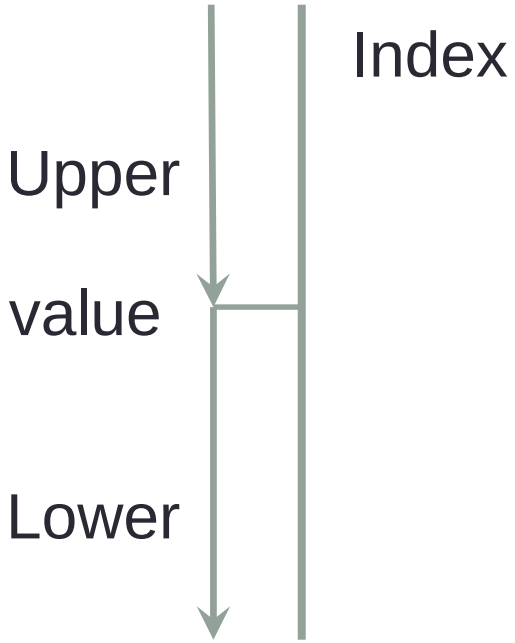
create index ixname on T1(name1)

Natural order

- Darth
- Boris
- Emmanuel
- Carl
- Alexandre
- Carlos

In index

- Alexandre
- Boris
- Carl
- Carlos
- Darth
- Emmanuel



Example of Lower Bound

```
SQL> select * from t1 where name1>'Carlos';

Select Expression
    -> Filter
        -> Table "T1" Access By ID
            -> Bitmap
                -> Index "IXNAME" Range Scan (lower bound: 1/1)

NAME1
==============================
Emmanuel
Dath
```

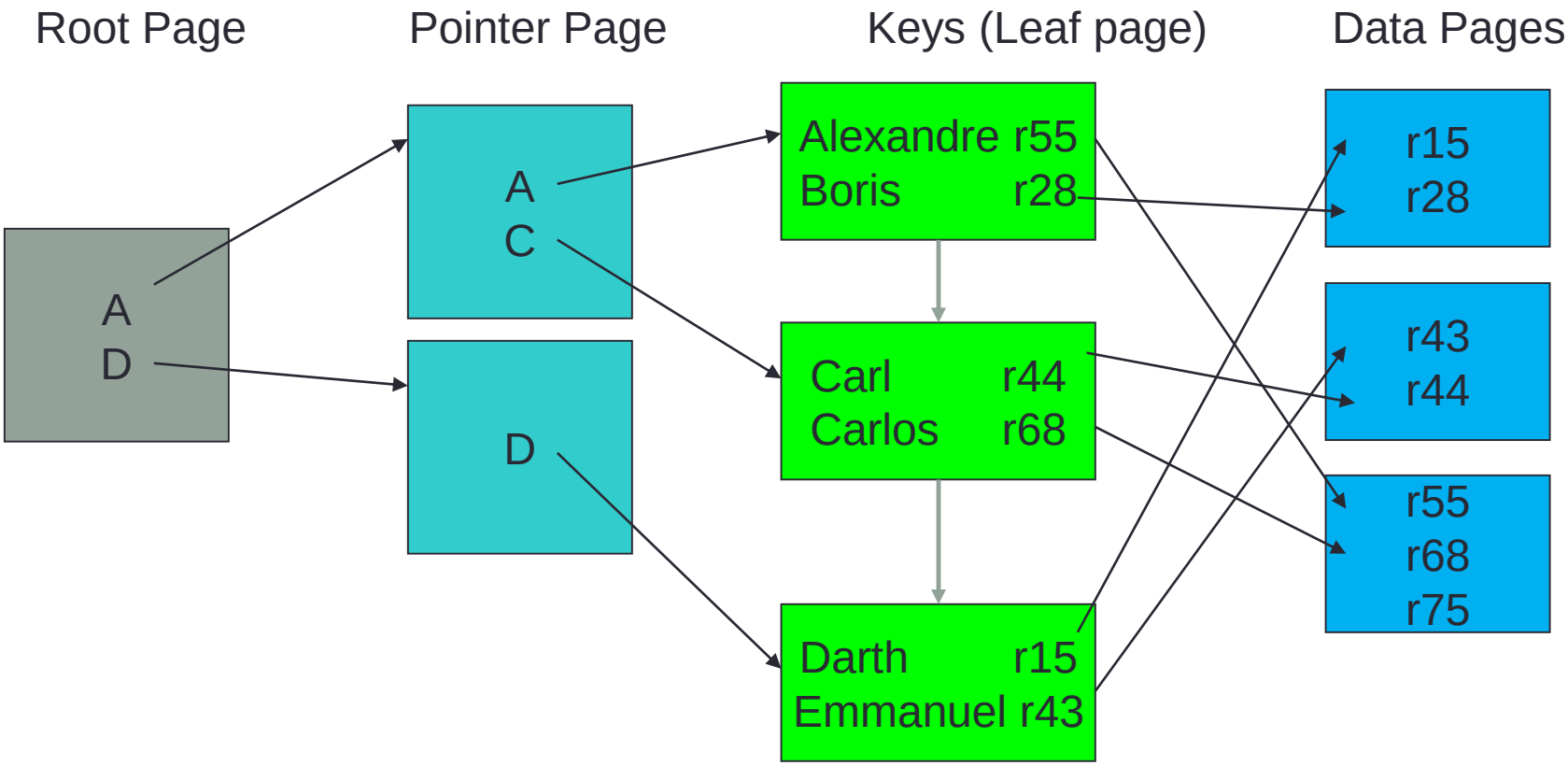
Example of Upper Bound

```
SQL> select * from t1 where name1<'Carlos';

Select Expression
    -> Filter
        -> Table "T1" Access By ID
            -> Bitmap
                -> Index "IXNAME" Range Scan (upper bound: 1/1)

NAME1
==============================
Carl
Alexandre
Boris
```

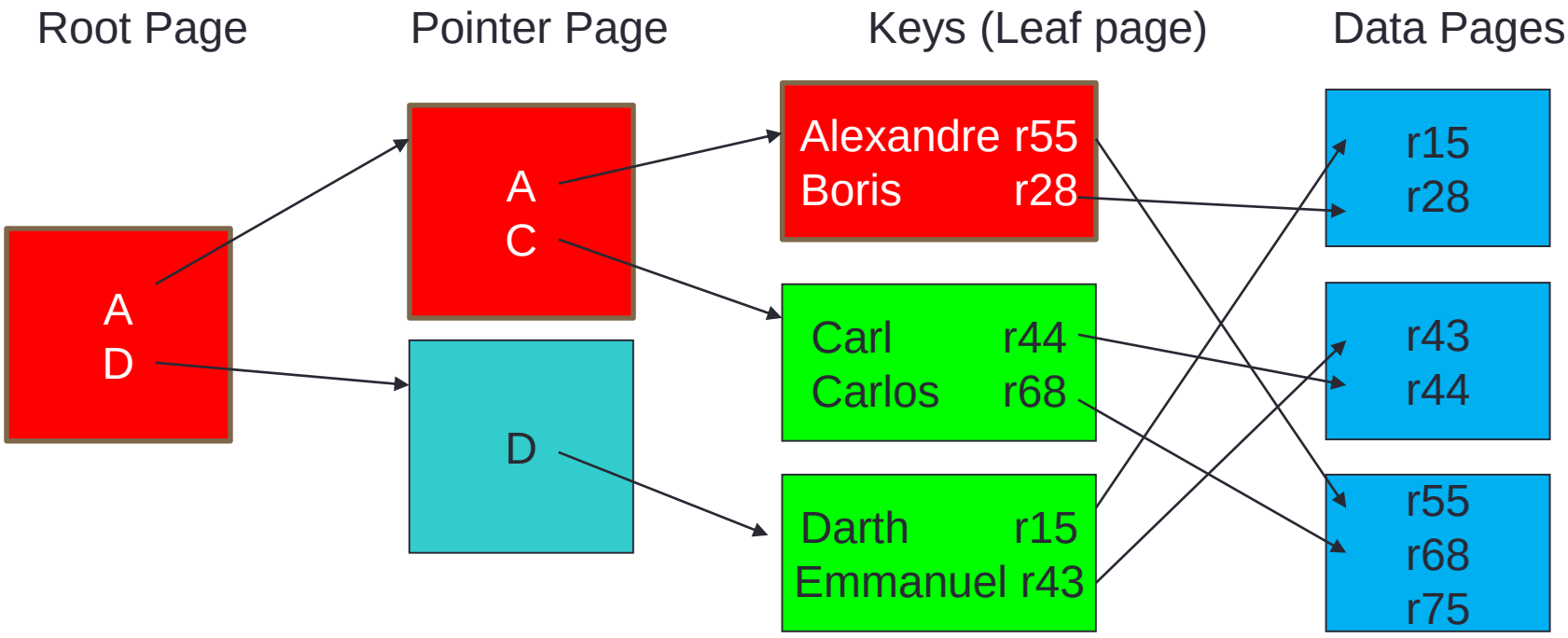
Example of Full match

```
SQL> select * from t1 where name1='Carlos';

Select Expression
    -> Filter
        -> Table "T1" Access By ID
            -> Bitmap
                -> Index "IXNAME" Range Scan (full match)

NAME1
==============================
Carlos
```

How data are stored in the index

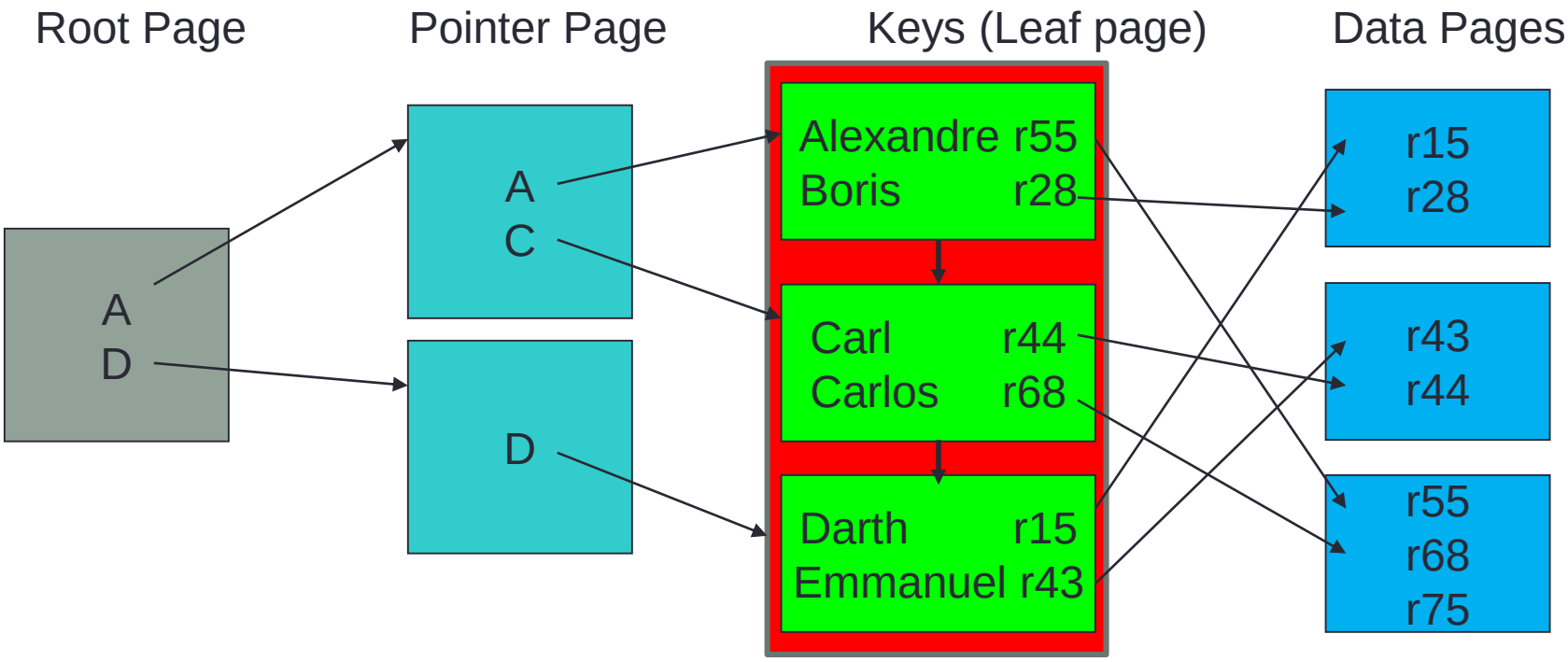


Step 1: Find the first key corresponding to the condition

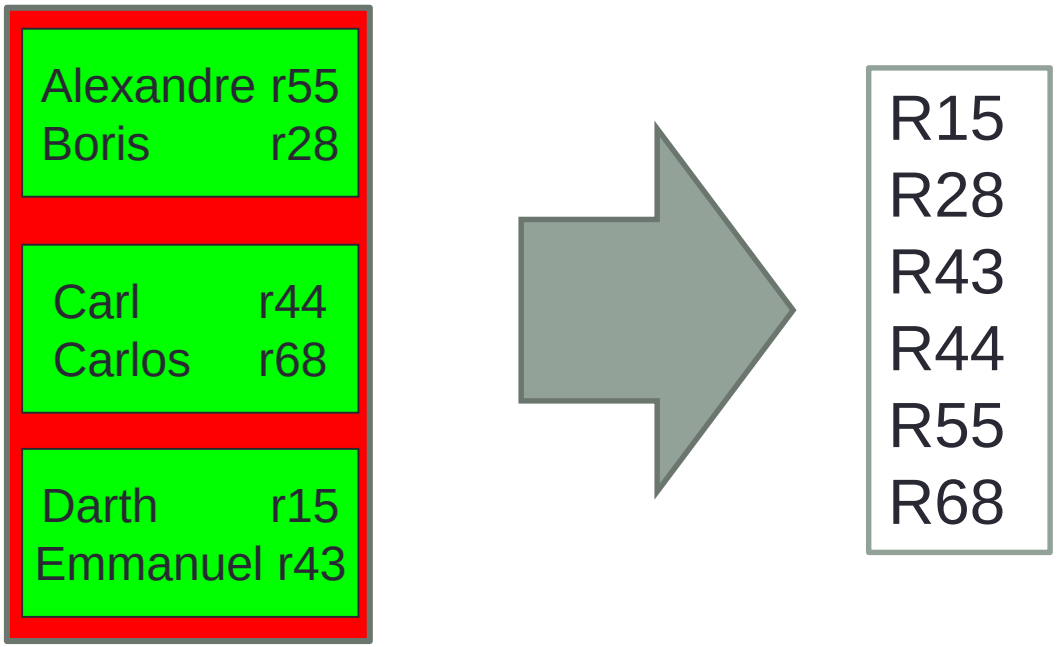


In this case, Firebird does at least 3 reads of index pages and 1 read of data page to read the first record.

Step 2: Get all record numbers for keys according the condition into the array



Step 3: sort records numbers



Composite indices

CREATE INDEX BY_AB ON MYTABLE (A, B)

SELECT * FROM MYTABLE
WHERE A = 1 AND B > 5

PLAN (MYTABLE INDEX (BY_AB))

A	B
1	1
1	2
1	3
2	1
2	2
2	3
3	1

- The second column depends on the first column.
- **where A > 1 and B > 5** - it will not use the second condition
- **where A = 1 and B = 5** it will use both conditions

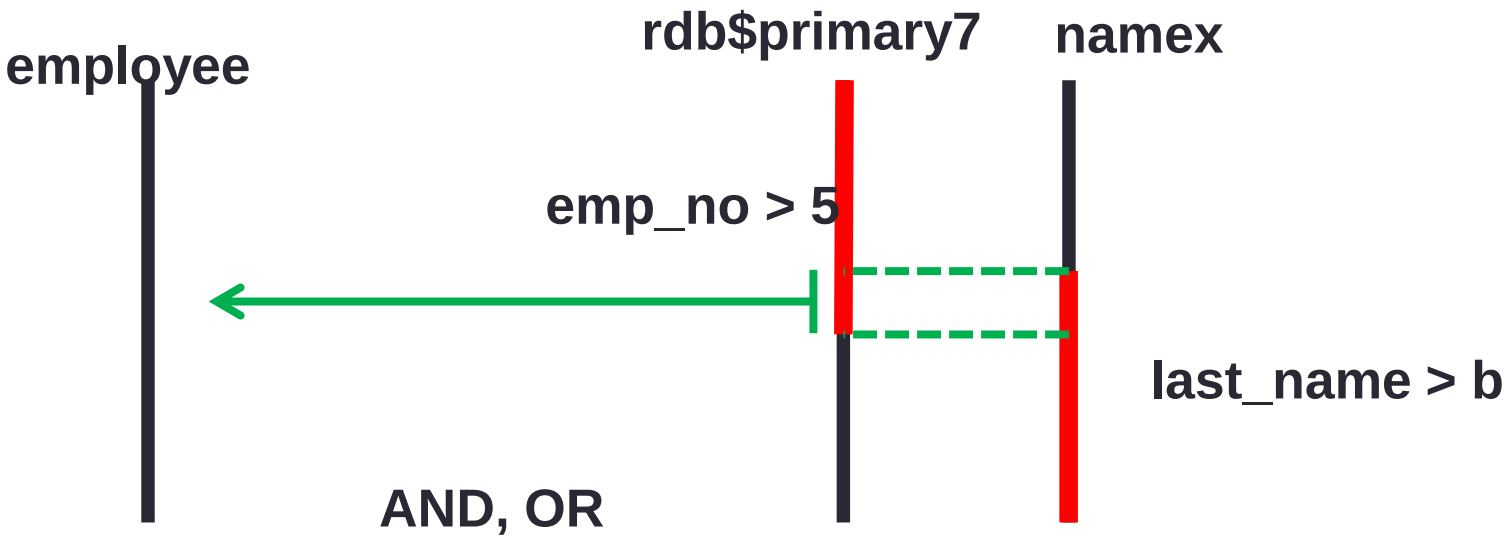
Index "RDB$INDEX_0" Range Scan (lower bound: 1/1)

- For composite indices > 1.
- 1st value – how many segments is used
- 2nd value – total number of segments
- 1/3 – one of 3 segments is used (bad)
- 2/3 – 2 of 3 are used (better)
- 3/3 – all segments are used (best)
- In case of 1/3, 2/3 – better use 1-segment indices

2 indices together: Index bitmap

- **select * from employee**
  where emp_no > 5 and last_name > 'b‘

PLAN (EMPLOYEE INDEX (RDB$PRIMARY7, NAMEX))



2 indices together: index bitmap

select * from a

where name > 'b' and a.id > 5

PLAN (A INDEX (ANAME, PK_A))

Select Expression

-> Filter

-> Table "A" Access By ID

-> **Bitmap And**

-> Bitmap

-> Index "ANAME" Range Scan (lower bound: 1/1)

-> Bitmap

-> Index "PK_A" Range Scan (lower bound: 1/1)

Full Scan or (TABLE ORDER INDEX)

select * from employee order by last_name;

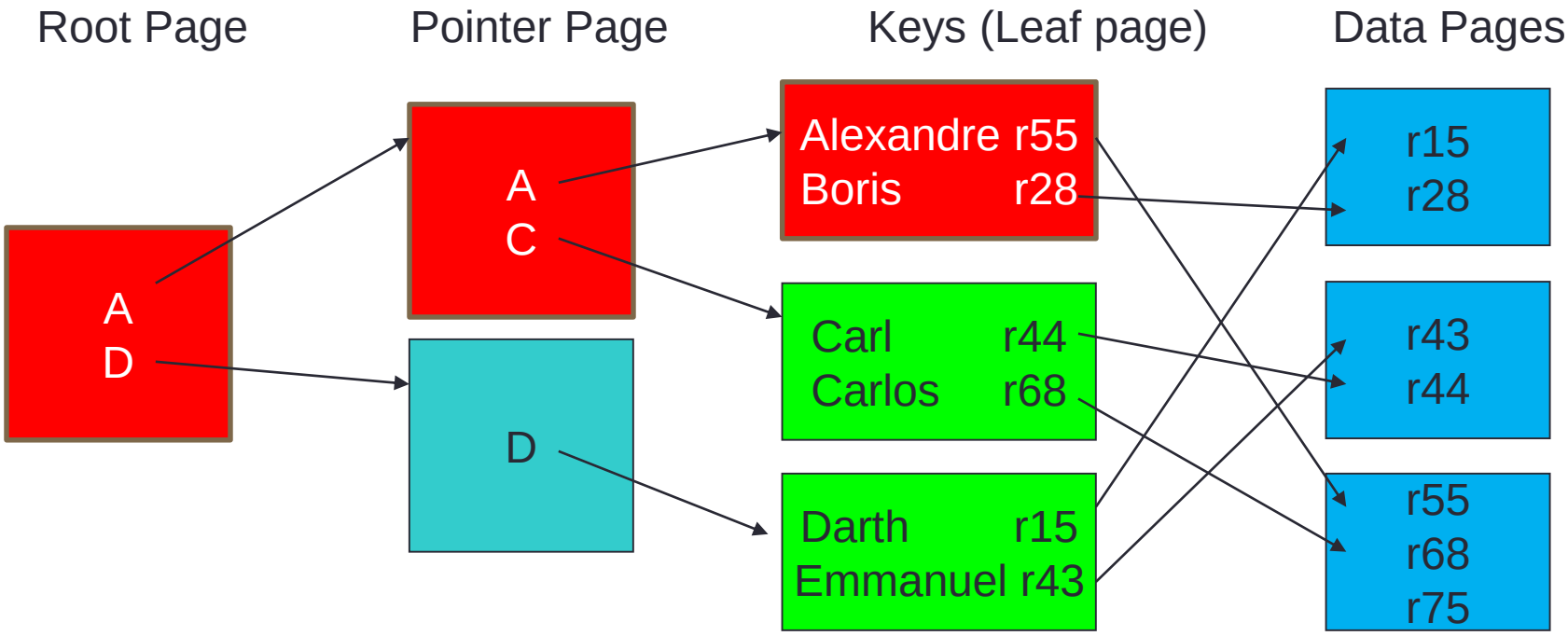
PLAN (EMPLOYEE ORDER NAMEX)

Select Expression

-> Table "EMPLOYEE" Access By ID

-> Index "NAMEX" **Full Scan**

Find the key corresponding to the condition



In this case, Firebird does at least 3 reads of index pages and 1 read of data page to read the first record.

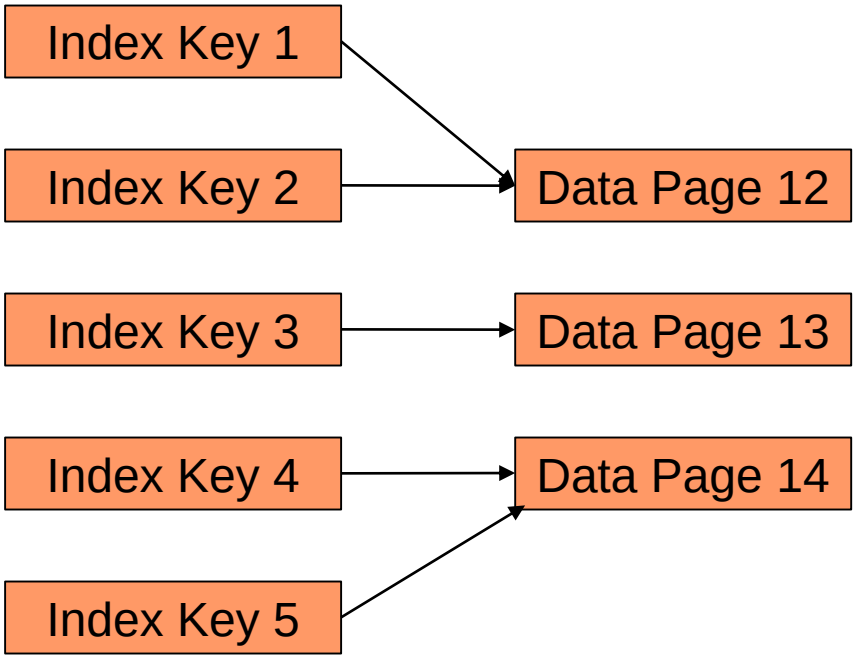
Example for Index Full Scan

- **select count(*) from table** (14mln записей)
  Execute time = 42s 500ms
  Buffers = 2048
  Reads = **118 792**
  Fetches = 28 814 893

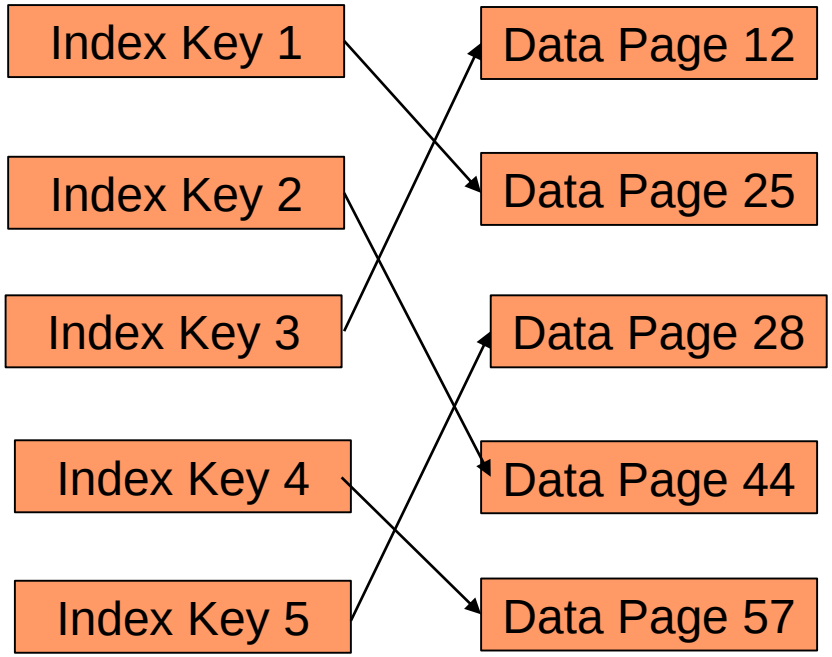
- **select a, count(a) from table**
  group by a
  PLAN (TABLE ORDER A)
  Execute time = 1m 12s 469ms
  Reads = 3 733 434
  every page was re-read to cache ~31 times
  Fetches = 42 869 143

Clustering factor

Good clustering factor:
INT/BIGINT autoincrement



Bad clustering factor:
GUID, random PK

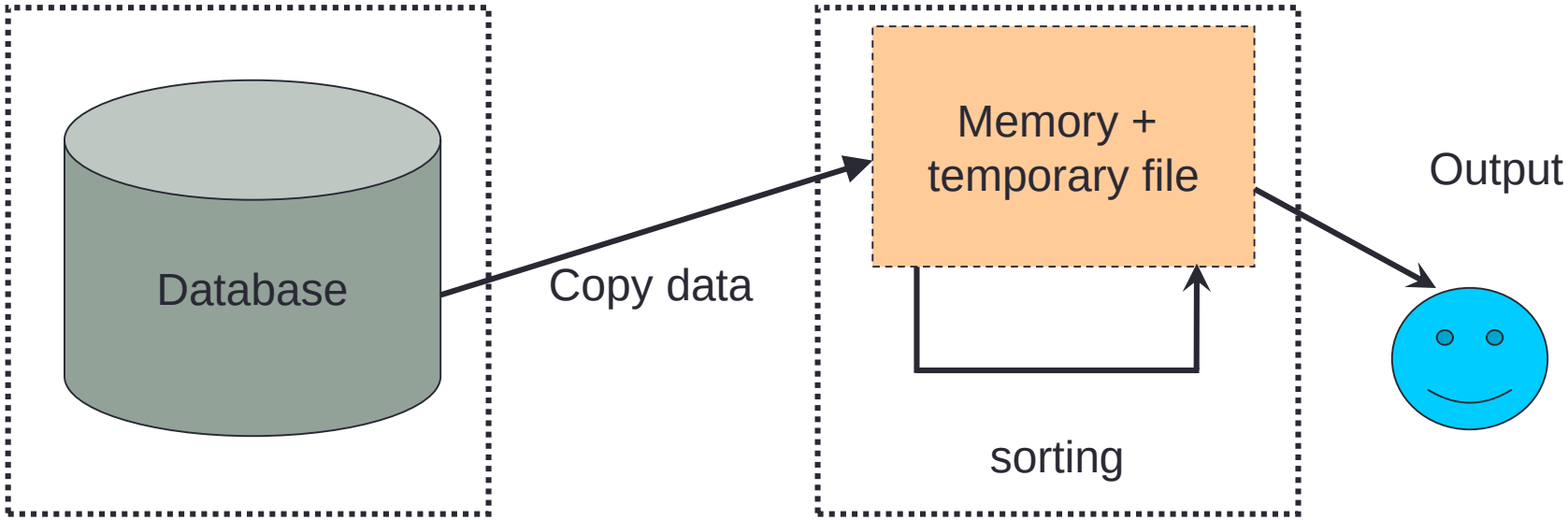


Summary for **table ORDER index**

- It returns the first records very fast, according the index
- Many jumps through index and data pages
  - High IO (3x more than natural per each record)
  - It leads to kicking pages from cache
- Only 1 index can be used – according ORDER BY or GROUP BY sequence
- Index Clustering factor is important to estimate the quality

PLAN SORT

- **select * from employee order by last_name||”**
  PLAN SORT ((EMPLOYEE NATURAL))
- Select Expression
- -> Sort (record length: 158, key length: 28)
- -> Table "EMPLOYEE" Full Scan



ORDER vs SORT

PLAN (TABLE ORDER A)
Execute time = 1m 12s
Buffers = *2 048*
Reads = 3 627 028
Fetches = 32 224 797

Execute time = 27s 518ms
Buffers = *150 000*
Reads = 124 663
Fetches = 32 224 797

PLAN SORT ((A NATURAL))
Execute time = 35 s
Buffers = 2 048
Reads = 119 915
Fetches 14 767 524

- If page cache is small, SORT will be faster
- To speed up SORT – TempCacheLimit, fast drive for temp files
- To speed up ORDER – increase cache (SuperServer only)

JOIN (recordset, recordset)

```
select e.last_name, p.proj_id
from employee e, employee_project p
where e.emp_no = p.emp_no
```

```
select e.last_name, p.proj_id
from employee e inner join employee_project p on
(e.emp_no = p.emp_no)
```

PLAN JOIN (P NATURAL, E INDEX (RDB$PRIMARY7))

JOIN PLAN variants

- JOIN (table1 NATURAL, table2 NATURAL)
- JOIN (table1 NATURAL, table2 INDEX indexname)
- JOIN (JOIN(table1 NATURAL, table 2 INDEX indexname), table indexname)

- LEFT/RIGHT, INNER, FULL JOIN

Join plan for INNER JOIN

```
select e.emp_no, d.department from employee e
inner join department d on (e.dept_no = d.dept_no);

Select Expression
    -> Nested Loop Join (inner)
        -> Table "DEPARTMENT" as "D" Full Scan
        -> Filter
            -> Table "EMPLOYEE" as "E" Access By ID
                -> Bitmap
                    -> Index "RDB$FOREIGN8" Range Scan (full match)
```

Join plan for LEFT/RIGHT

```
select e.emp_no, d.department from employee e
left join department d on (e.dept_no = d.dept_no);

Select Expression
    -> Nested Loop Join (outer)
        -> Table "EMPLOYEE" as "E" Full Scan
        -> Filter
            -> Table "DEPARTMENT" as "D" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY5" Unique Scan
```

INNER JOIN + LEFT JOIN

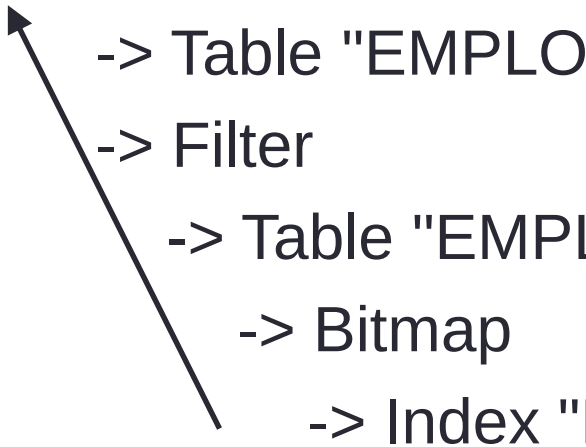
```
select e.emp_no, d.department
from employee e inner join employee_project p
        on (e.emp_no = ep.emp_no)
left join department d
        on (e.dept_no = d.dept_no);
```

Old plan

```
PLAN JOIN (JOIN (EP NATURAL, E INDEX (RDB$PRIMARY7)),
D INDEX (RDB$PRIMARY5))
```

PLAN for INNER JOIN+LEFT

```
Select Expression
    -> Nested Loop Join (outer)
        -> Nested Loop Join (inner)
            -> Table "EMPLOYEE_PROJECT" as "EP" Full Scan
            -> Filter
                -> Table "EMPLOYEE" as "E" Access By ID
                    -> Bitmap
                        -> Index "RDB$PRIMARY7" Unique Scan
        -> Filter
            -> Table "DEPARTMENT" as "D" Access By ID
                -> Bitmap
                    -> Index "RDB$PRIMARY5" Unique Scan
```



HASH JOIN

select e.* from employee e, employee_project p

where e.emp_no+0 = p.emp_no+0

PLAN HASH (E NATURAL, P NATURAL)

Select Expression

-> Filter

-> **Hash Join (inner)**

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

-> Record Buffer (record length: 25)

-> Table "EMPLOYEE_PROJECT" as "P" Full Scan

Summary for new plans

- Natural
  - FullScan
- Index
  - RangeScan
  - FullScan
- Sort
- JOIN
  - Inner
  - Outer
  - Hash

END OF PART 1