Oracle Performance Tuning

WEEK 8

Challenge (VANZARI schema)

Bill from the “Sales” department reach you and he’s complaining about a report running slow.

You find the slow query:

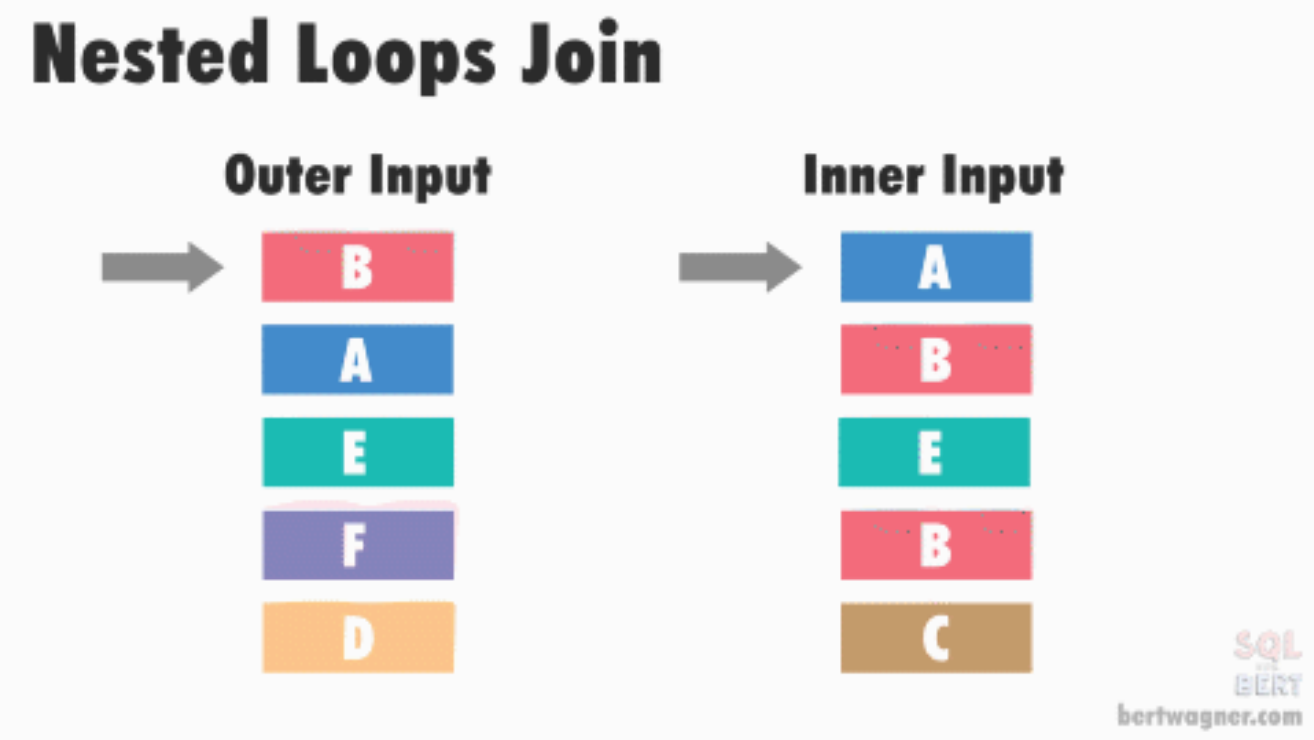
```
select * from facturi
where datafact between
to_date('01/08/2015',
'dd/mm/yyyy') and
to_date('30/09/2015',
'dd/mm/yyyy');
```

You can also see that the above query is concurrently executed by a lot of users.

Implement a solution to fix the performance problem and explain why you’ve chosen it.

Optimizing JOINs

HOW MANY JOIN TYPES DO YOU KNOW?



The Nested Loop Join

?!

```
SELECT /*+ use_nl(c cp) gather_plan_statistics*/ c.codcl, c.dencl, cp.loc
  FROM coduri_postale cp INNER JOIN clienti c ON CP.CodPost = C.CodPost;
```

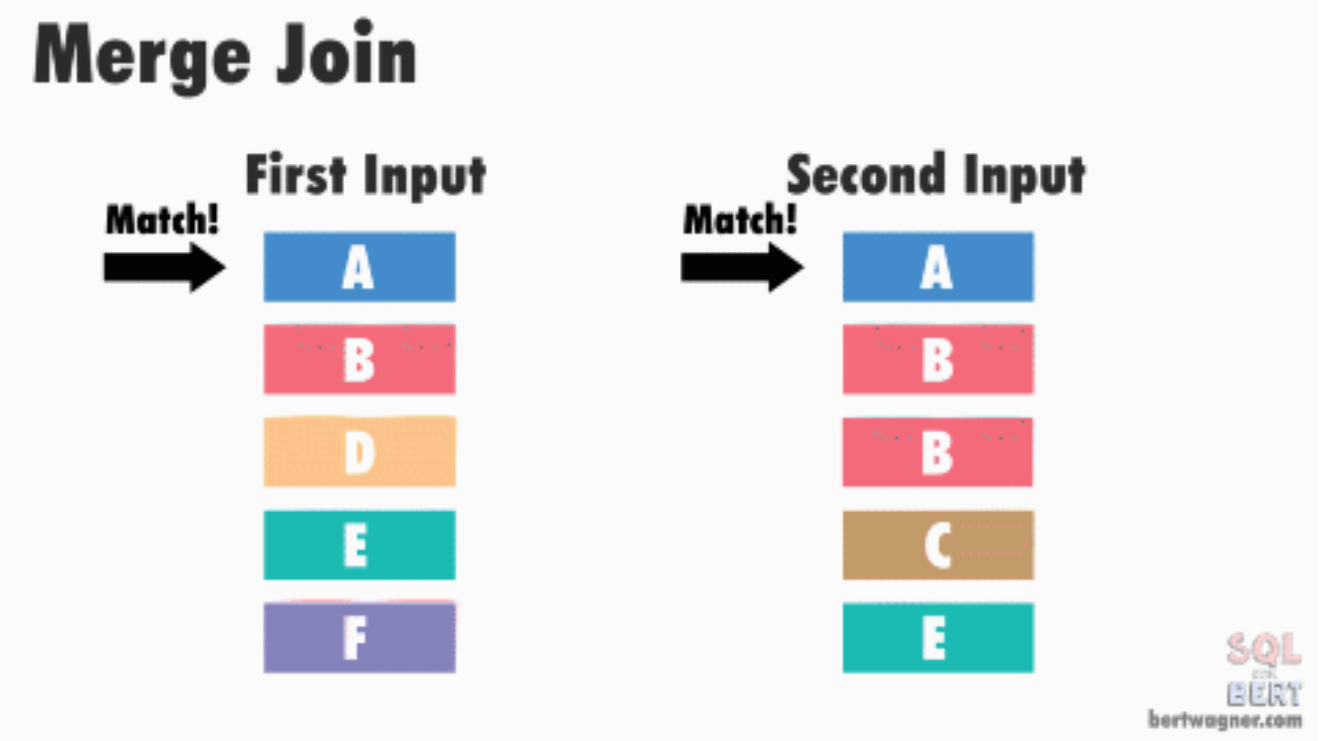
Query Result | Autotrace

SQL HotSpot | 0.803 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			
NESTED LOOPS		1000	
TABLE ACCESS (FULL)	CODURI_POSTALE	10	
TABLE ACCESS (FULL)	CLIENTI	100	
Filter Predicates			
CP.CODPOST=C.CODPOST			

V$STATNAME Name	V$MYSTAT Value
cluster key scans	9
consistent gets	233
consistent gets examination	71

NL Example



Sort Merge Join

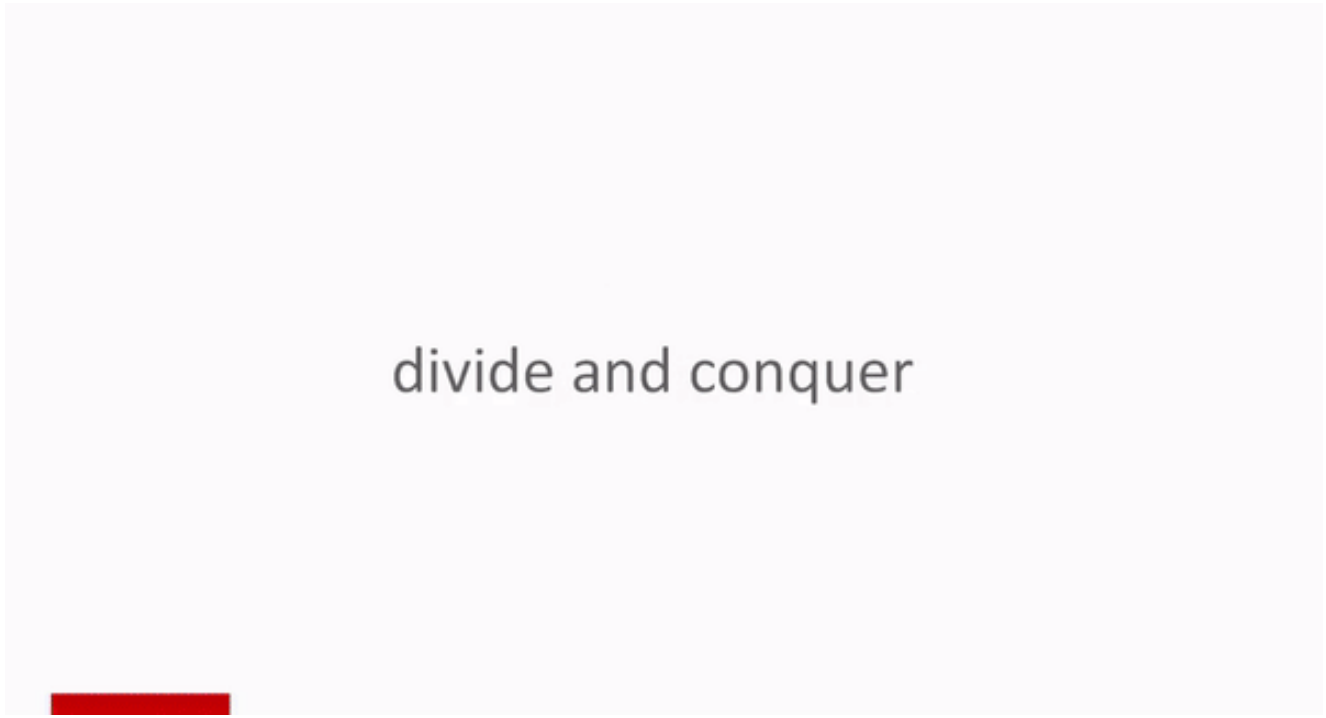
```
SELECT /*+ gather_plan_statistics use_merge(c f l p) */
  c.dencl, f.nrfact, f.datafact, p.denpr, l.cantitate
FROM clienti c INNER JOIN facturi f
ON c.codcl = f.codcl INNER JOIN liniifact l
ON f.nrfact = l.nrfact INNER JOIN produse p
ON  l.codpr = p.codpr
ORDER BY 1, 2, 3;
```

Query Result | Autotrace

SQL HotSpot | 2.565 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			10484
SORT (ORDER BY)		151393	10484
MERGE JOIN		151393	5961
SORT (JOIN)		151393	5957
MERGE JOIN		151393	2378
SORT (JOIN)		26280	685
MERGE JOIN		26280	276
SORT (JOIN)		1000	6
TABLE ACCESS (F	CLIENTI	1000	5
SORT (JOIN)		26280	270
Access Predicat			
Filter Predicates			
TABLE ACCESS (F	FACTURI	26280	22
Filter Predica			
SORT (JOIN)		151393	1693
Access Predicates			
Filter Predicates			
TABLE ACCESS (FULL)	LINIIFACT	151393	171
Filter Predicates			
SORT (JOIN)		1000	4
Access Predicates			
Filter Predicates			
TABLE ACCESS (FULL)	PRODUSE	1000	3

Sort MJ Example



HASH Joins

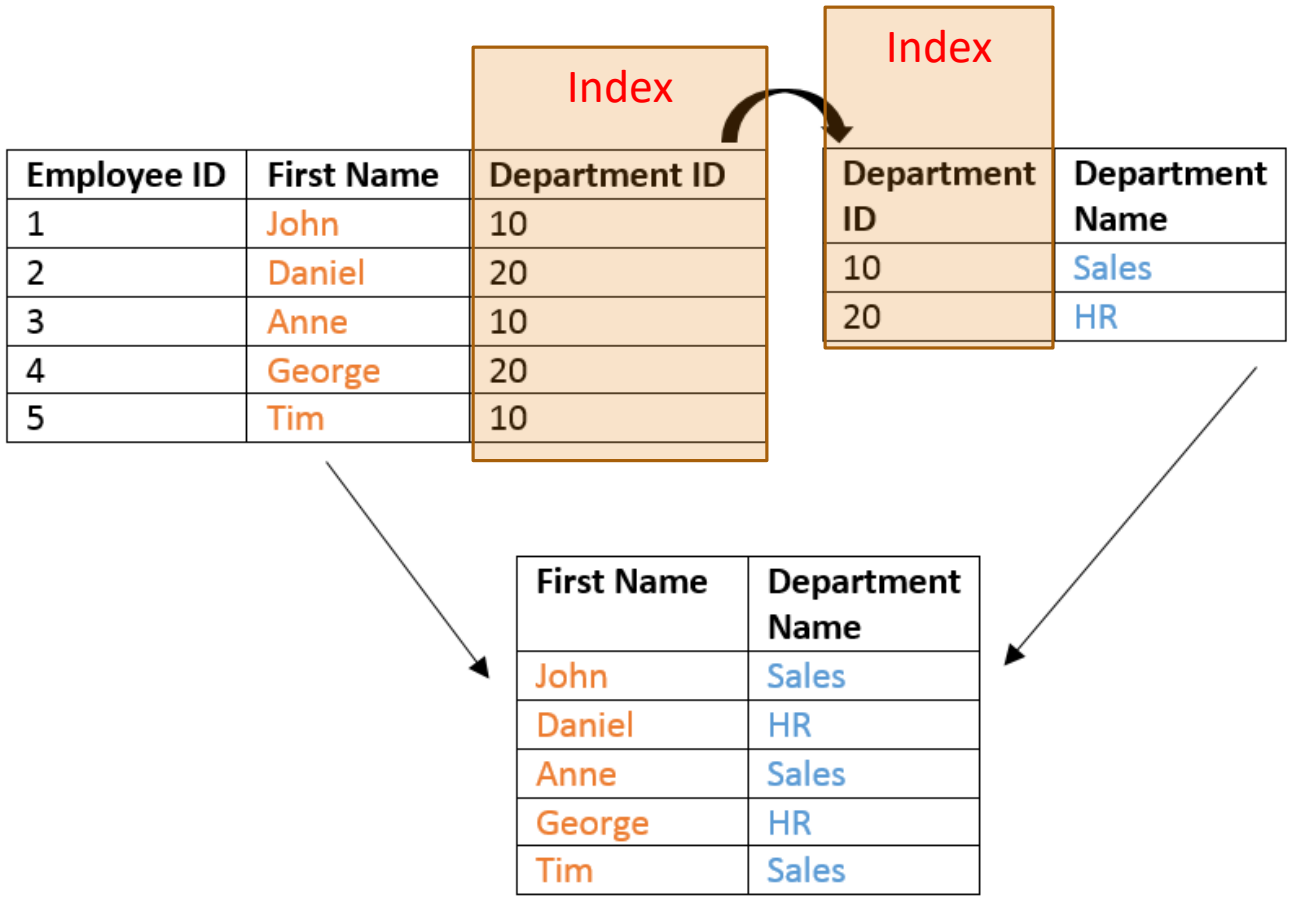
```
SELECT /*+ gather_plan_statistics use_hash(c f l p) */
  c.dencl, f.nrfact, f.datafact, p.denpr, l.cantitate
FROM clienti c INNER JOIN facturi f
ON c.codcl = f.codcl INNER JOIN liniifact l
ON f.nrfact = l.nrfact INNER JOIN produse p
ON l.codpr = p.codpr;
```

Query Result | Autotrace

SQL HotSpot | 0.587 seconds

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			627
HASH JOIN		151393	627
Access Predicates			
L.CODPR=P.CODPR			
TABLE ACCESS (FULL)	PRODUSE	1000	3
HASH JOIN		151393	624
Access Predicates			
C.CODCL=F.CODCL			
TABLE ACCESS (FULL)	CLIENTI	1000	5
HASH JOIN		151393	618
Access Predicates			
F.NRFACT=L.NRFACT			
TABLE ACCESS (FULL)	FACTURI	26280	22
Filter Predicates			
F.CODCL>1000			
TABLE ACCESS (FULL)	LINIIFACT	151393	171
Filter Predicates			
L.CODPR>0			

HJ Example



Indexing Joining Columns

Bitmap Join Indexes

EMPLOYEES table

Employee_id	Name	Department_id
1	Larry	1
2	Ken	1
3	Ray	2
4	Married	3
5	Married	3
6	Single	2
7	Married	2
.....	.....	.....

DEPARTMENTS table

Department_id	Name
1	Sales
2	R&D
3	Marketing

Sales	R&D	Marketing
1	0	1
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0
.....	.....	.....

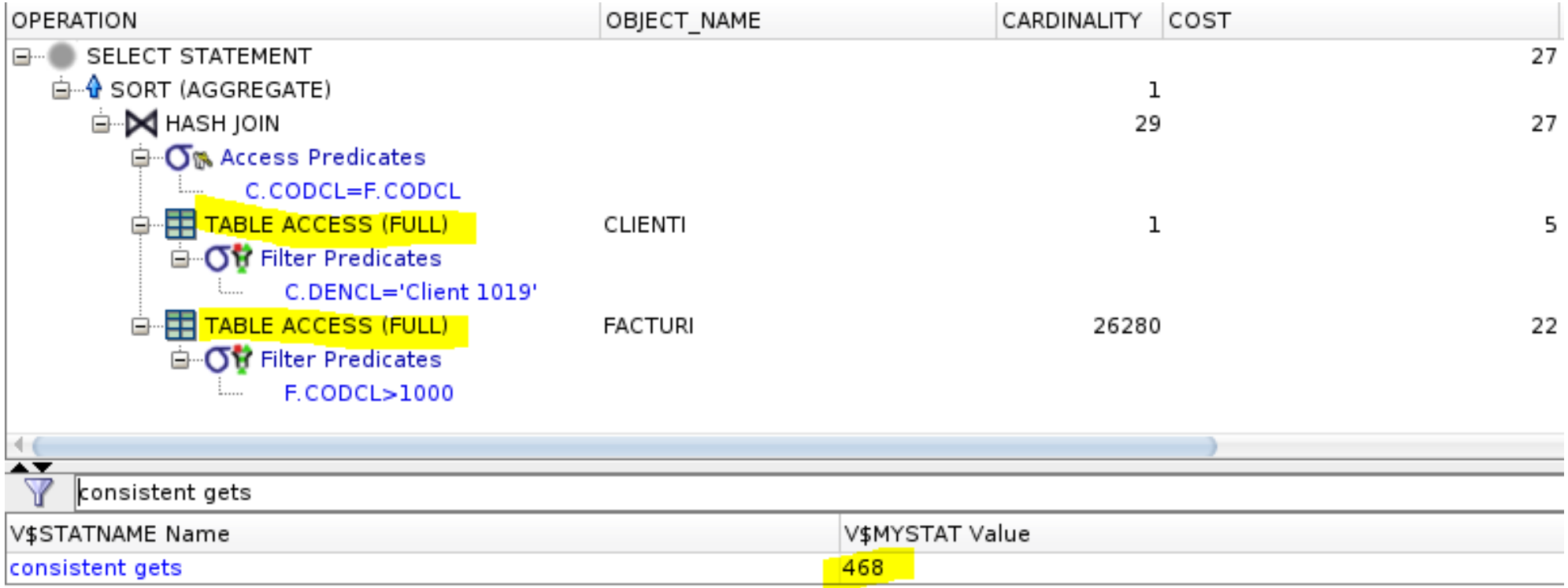
Bitmap Join Index

```
CREATE BITMAP INDEX bi_1 ON employees(d.department_name)
 FROM employees e, departments d WHERE d.department_id=e.department_id
```

Bitmap Join Example (part 1)

The Query:

```
SELECT COUNT(*) FROM CLIENTI C, FACTURI F
WHERE C.CODCL=F.CODCL   AND C.DENCL='Client 1019';
```



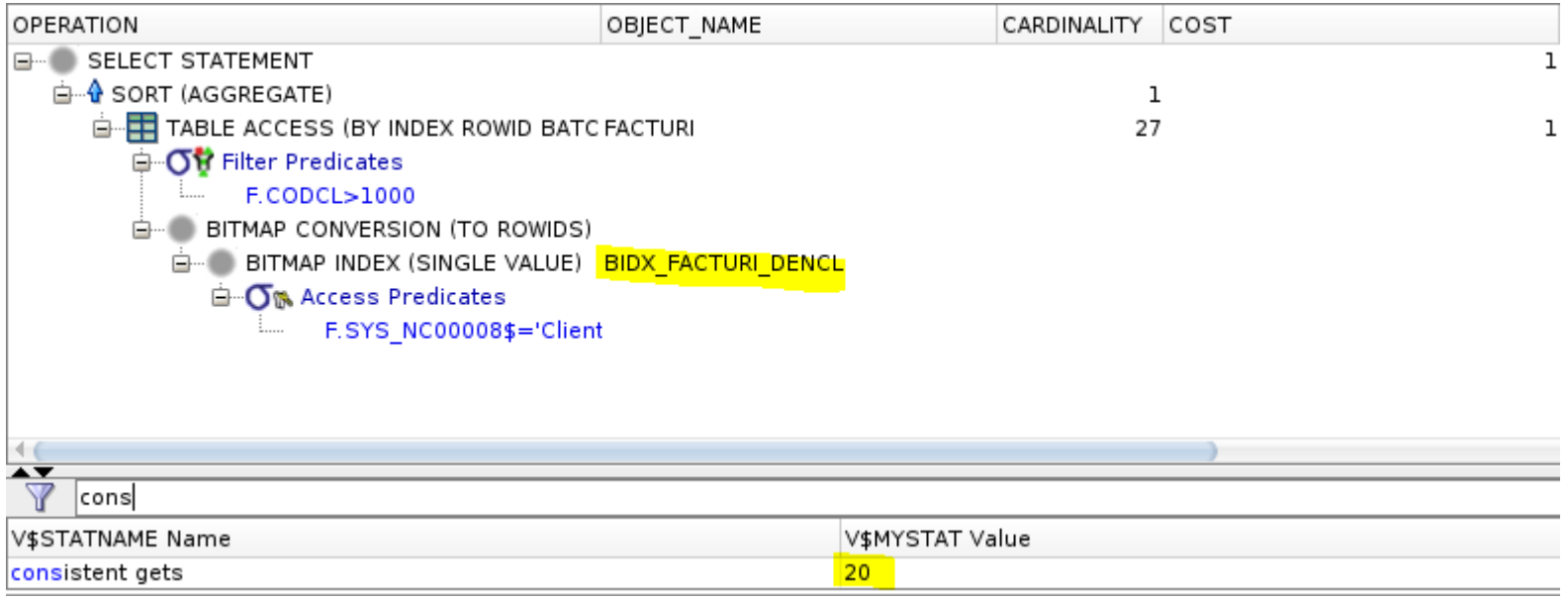
OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT			27
SORT (AGGREGATE)		1	
HASH JOIN		29	27
Access Predicates			
C.CODCL=F.CODCL			
TABLE ACCESS (FULL)	CLIENTI	1	5
Filter Predicates			
C.DENCL='Client 1019'			
TABLE ACCESS (FULL)	FACTURI	26280	22
Filter Predicates			
F.CODCL>1000			

consistent gets

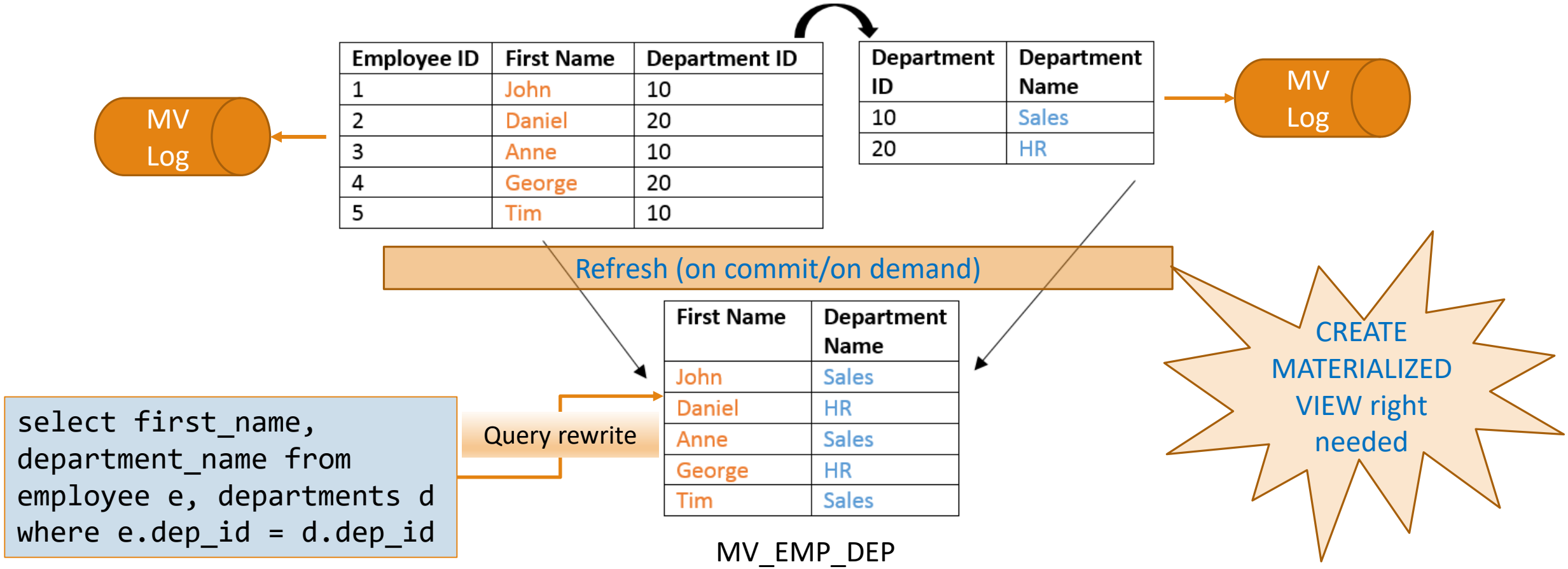
V$STATNAME Name	V$MYSTAT Value
consistent gets	468

Bitmap Join Example (part 2)

```
CREATE BITMAP INDEX BIDX_FACTURI_DENCL ON
FACTURI(C.DENCL) FROM FACTURI F, CLIENT C
WHERE C.CODCL = F.CODCL;
```



Materialized Views



There is an application which is using data from “VANZARI” schema

Some users are mad because one of their frequently used report is running slow

You find out that the following query is behind that report:

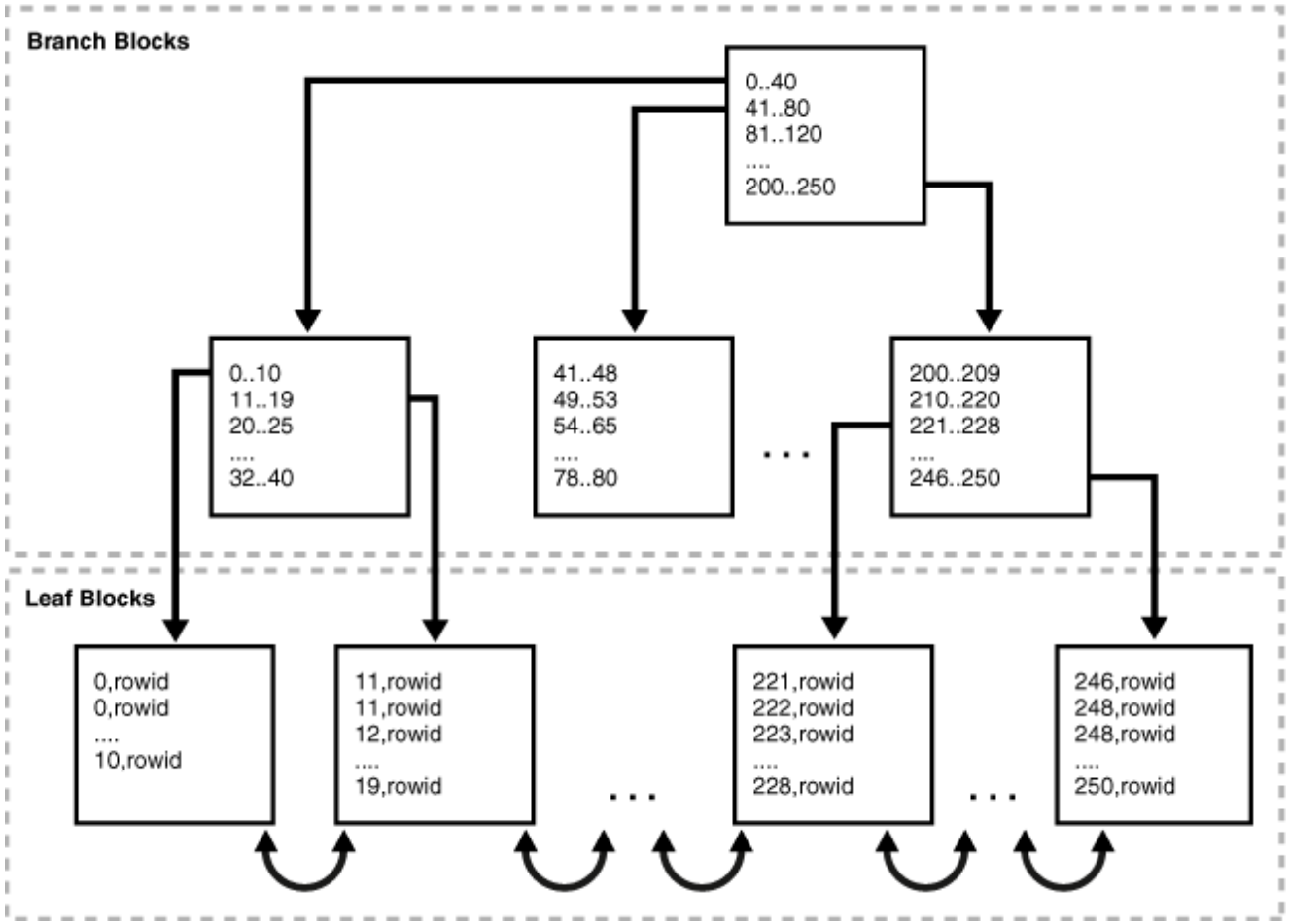
```
SELECT F.NRFACT, L.CODPR, L.CANTITATE FROM FACTURI F, LINIIFACT L WHERE F.NRFACT=L.NRFACT AND F.DATAFACT < TO_DATE('01/09/2015', 'DD/MM/YYYY')
```

Suggest and implement an improvement for the above query

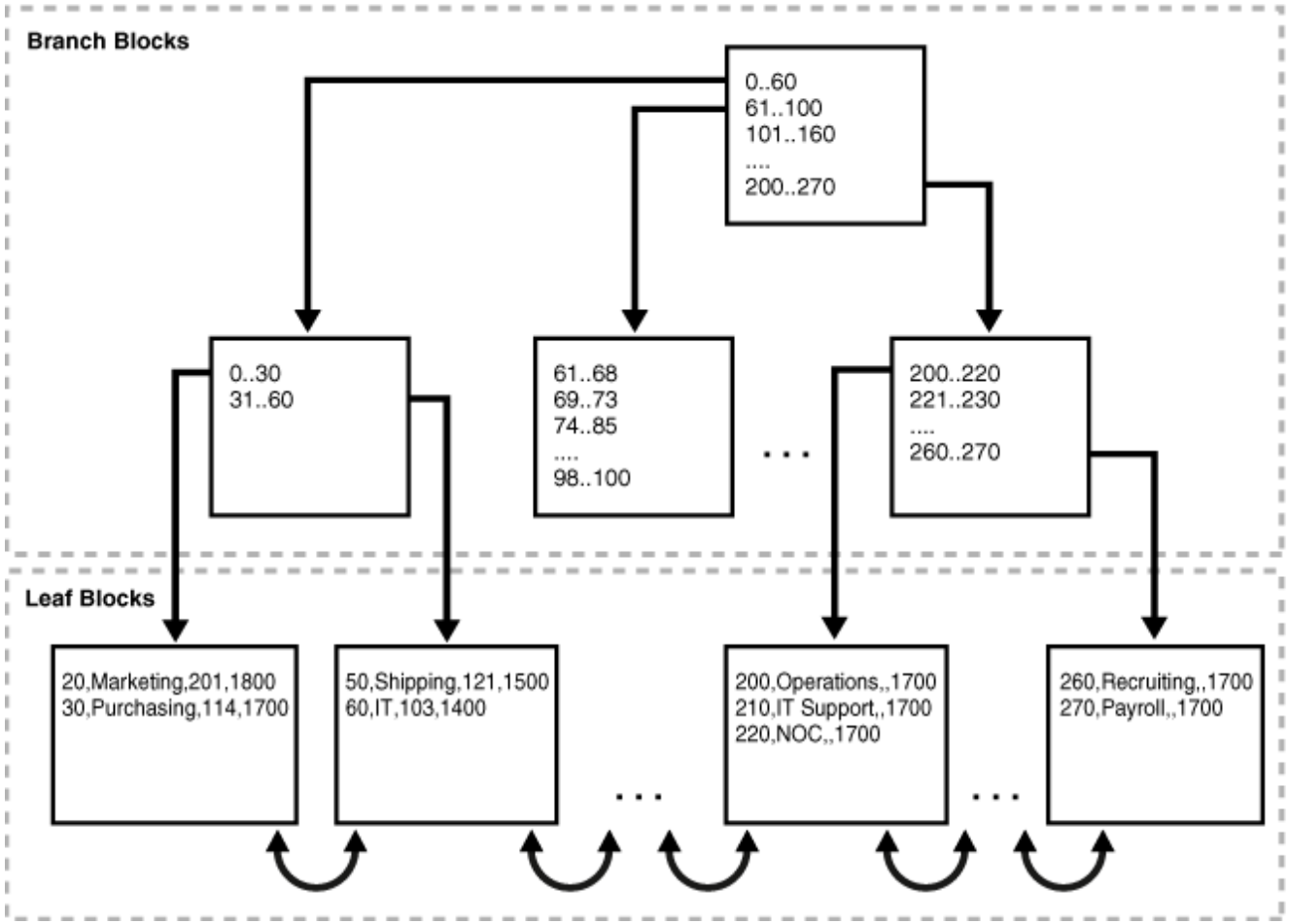
How can you tell that the fix you suggested did the job?

Hands-on Practice

Other SQL Tuning Techniques



Do you remember indexes?


Branch Blocks
0..60
61..100
101..160
....
200..270
0..30
31..60
61..68
69..73
74..85
....
98..100
200..220
221..230
....
260..270
Leaf Blocks
20,Marketing,201,1800
30,Purchasing,114,1700
50,Shipping,121,1500
60,IT,103,1400
200,Operations,,1700
210,IT Support,,1700
220,NOC,,1700
260,Recruiting,,1700
270,Payroll,,1700

Tabele IOT

Heap-Organized Table	Index-Organized Table
The **rowid** uniquely identifies a row. Primary key constraint may optionally be defined.	Primary key uniquely identifies a row. Primary key constraint must be defined.
Physical rowid in `ROWID` **pseudocolumn** allows building secondary indexes.	Logical rowid in `ROWID` pseudocolumn allows building secondary indexes.
Individual rows may be accessed directly by rowid.	Access to individual rows may be achieved indirectly by primary key.
Sequential **full table scan** returns all rows in some order.	A **full index scan** or fast full index scan returns all rows in some order.
Can be stored in a **table cluster** with other tables.	Cannot be stored in a table cluster.
Can contain a column of the `LONG` data type and columns of **LOB** data types.	Can contain LOB columns but not `LONG` columns.
Can contain **virtual columns** (only relational heap tables are supported).	Cannot contain virtual columns.

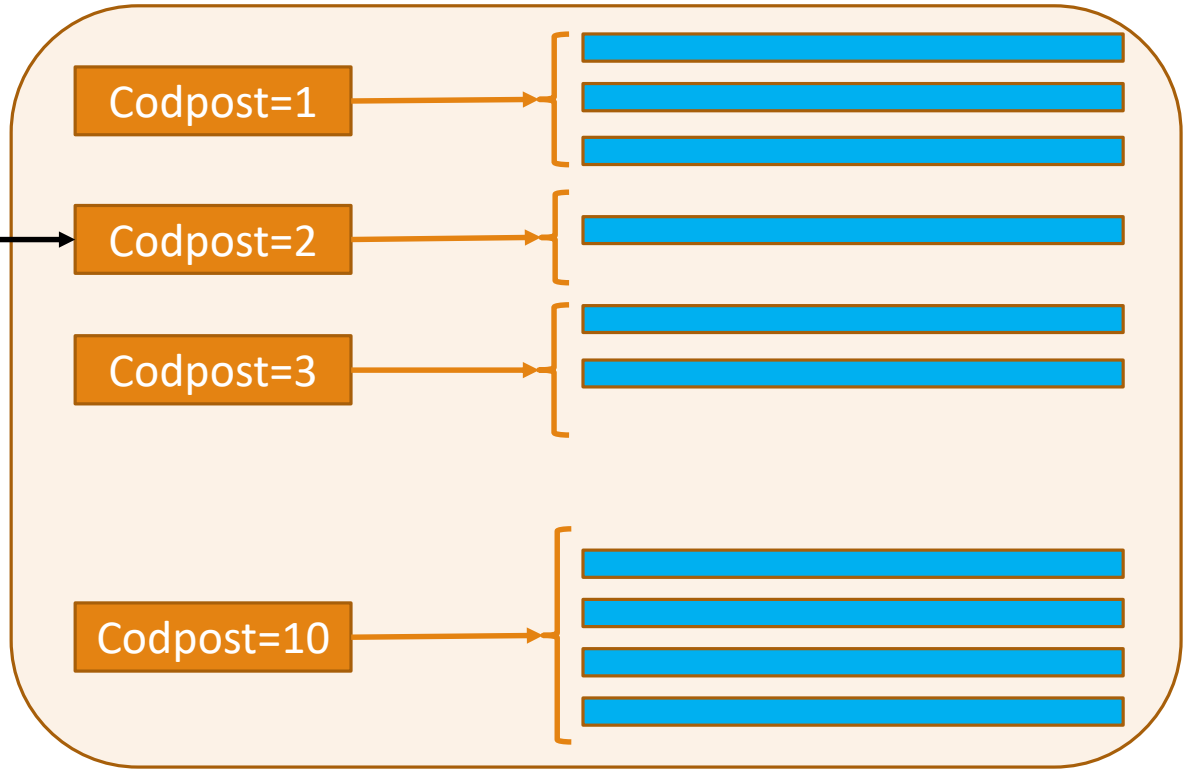
Regular tables vs. IOTs

Hash Clusters

```
create cluster coduri_postale_hash (
  codpost char(6)
)
hashkeys 10
size 8192;

create table clienti_cluster (
  codcl number(6),
  dencl varchar2(30),
  codfiscal char(9),
  adresa varchar2(40),
  codpost char(6),
  telefon varchar2(10)
) cluster coduri_postale_hash(codpost);
```

```
select * from
clienti_cluster where
codpost=2
```



That’s all folks!

THANK YOU AND SEE YOU NEXT WEEK…