

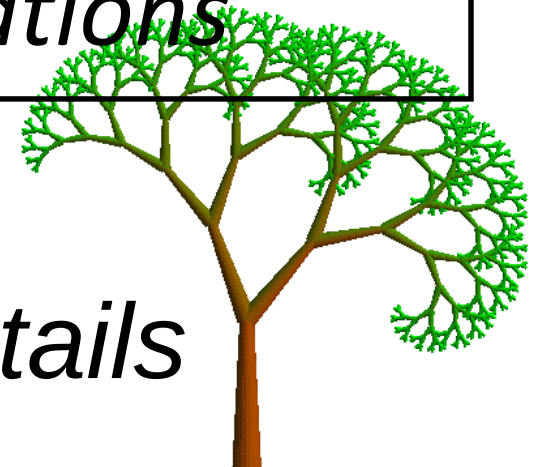


Query Tuning Foundations

Execution Plan

A - Row Source Operations

Branch and Leaf Details





Objectives

- After completing this lesson, you should be able to do the following:
 - Understand the meaning of usual operations
 - Become familiar with very simple execution plan



Row Source Operation Description

- Unary Operations
 - 1 - Access Path
 - 2 – Other Unary Operations
- Binary Operations
 - 3 – Joins
- N-ary Operations
 - 4 – Other N-ary Operations



1 Structures / Access Paths

STRUCTURES

1. Tables (1 - 2)
2. Indexes to access tables
B_tree (3 - 4 - 5 - 6 - 7 - 8)
Index bitmap (9 - 10)
3. Tables organized by :
Cluster
Hash cluster
IOT

ACCESS PATHS

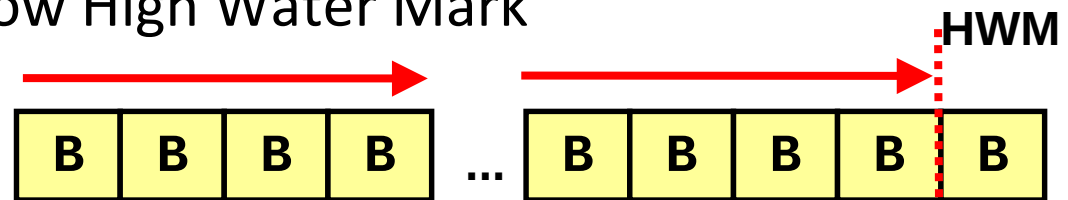
1. Full Table Scan
2. Rowid Scan
3. Index Scan (Unique)
4. Index Scan (Range)
5. Index Scan (Full)
6. Index Scan (Fast Full)
7. Index Scan (Skip)
8. Index Scan (Index Join)
9. Using Bitmap Indexes
10. Combining Bitmap Indexes

1.1 - Full Table Scan

- Performs Multi-block Reads
(here DB_FILE_MULTIBLOCK_READ_COUNT = 4)

- Reads All Blocks Below High Water Mark

- Filters rows



```
select * from emp where ename = 'KING';
```

Id	Operation	Name	Rows	Bytes	Cost	

0	SELECT STATEMENT		1	22	2	
* 1	TABLE ACCESS FULL	EMP	1	22	2	

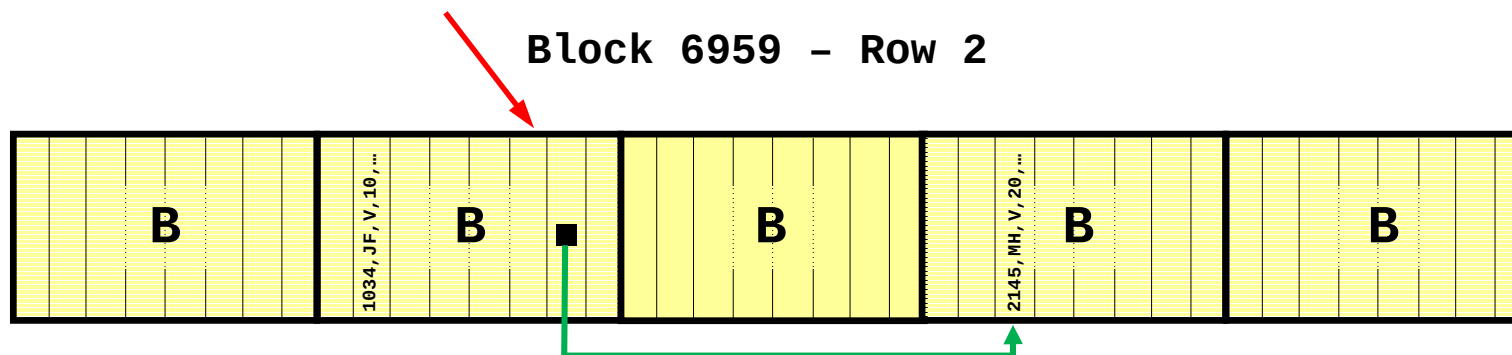
Predicate Information (identified by operation id):

```
-----  
1 - filter("EMP"."ENAME"='KING')
```

1.2 - Rowid Scan

```
SQL> explain plan for
  2 select * from emp where rowid='AAAJ5QAAFAAABsvAAC';
```

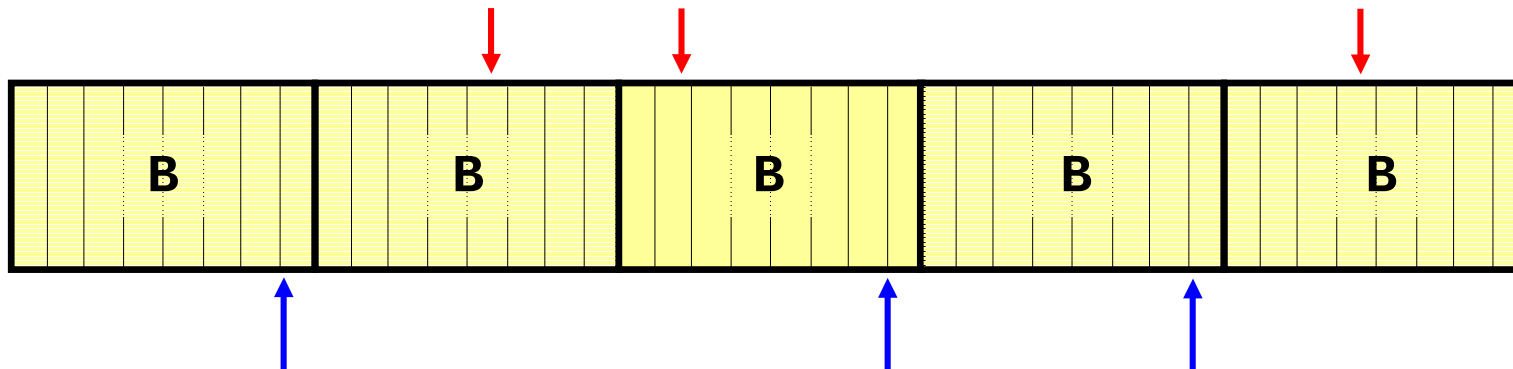
Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	1
1	TABLE ACCESS BY USER ROWID	EMP	1	1



Sample Table Scan

```
SELECT * FROM emp SAMPLE BLOCK (10) [SEED (1)];
```

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)
0	SELECT STATEMENT		4	99	2	(0)
1	TABLE ACCESS SAMPLE	EMP	4	99	2	(0)





Types of Indexes

- Index storage techniques:
 - **B*-tree**
 - Normal
 - Descending
 - Reverse key
 - Key compression
 - Function based
 - Index Organized Table
 - **Bitmap** (Stored in a B-Tree)
 - Domain indexes

B*-Tree Indexes

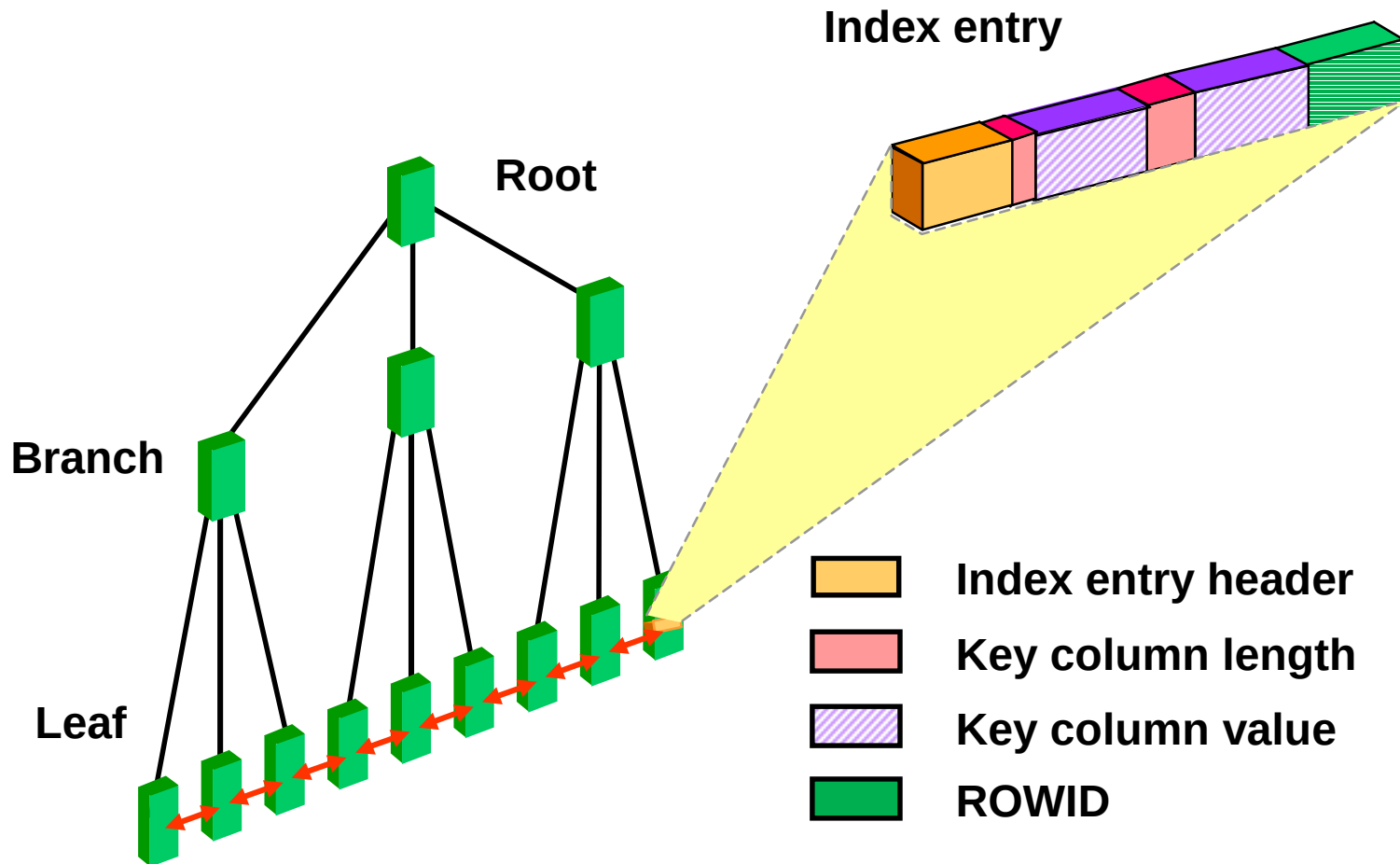
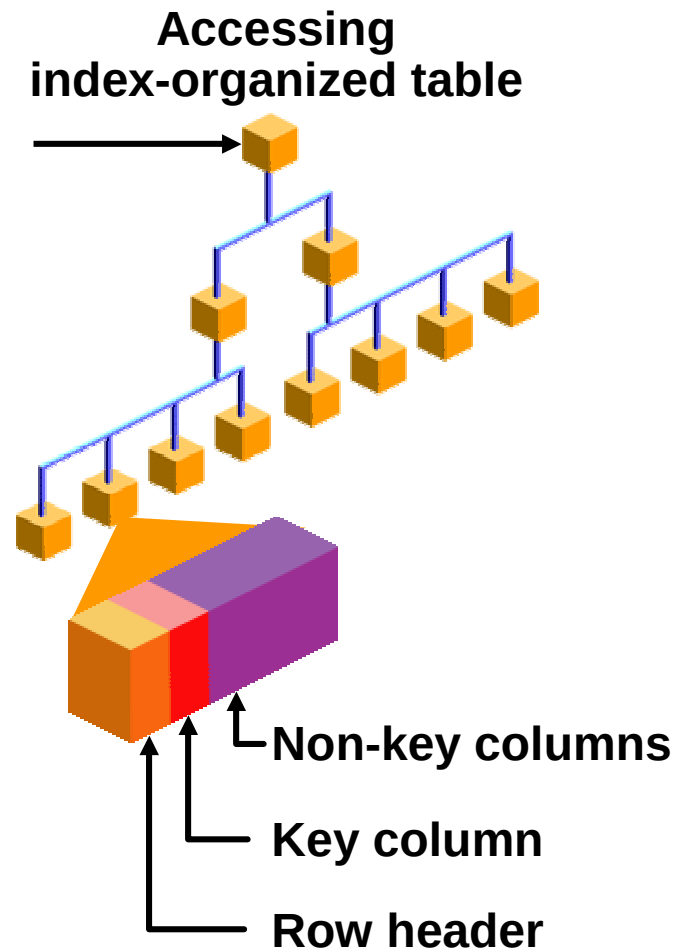
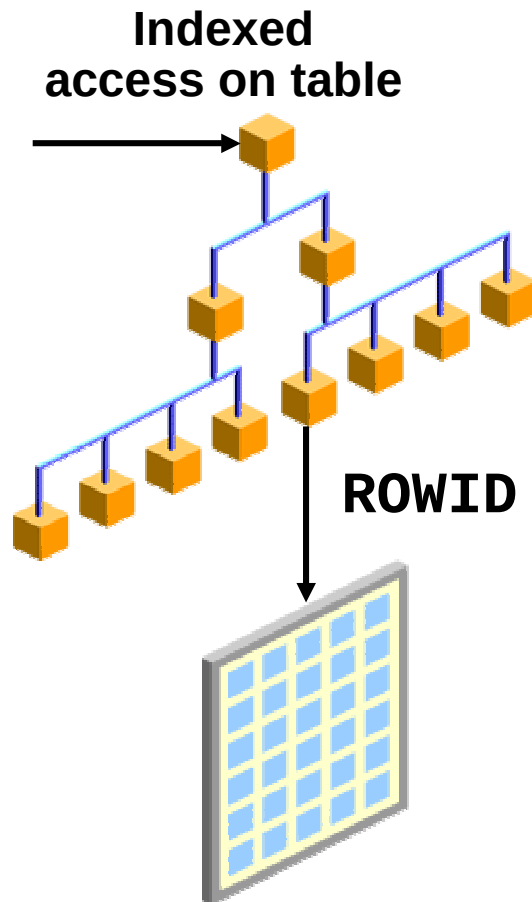


Table Data retrieved using Rowid

Index-Organized Tables





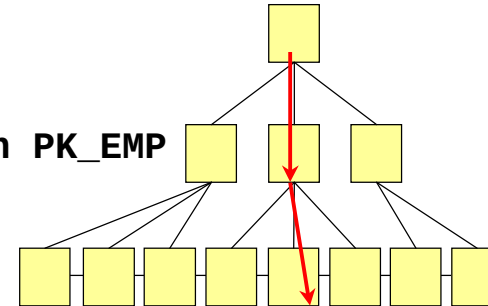
- Unique
- Range (Descending)
- Skip
- Full & Fast Full
- Index join

Table EMP



1.3 - Index Scan (Unique)

index UNIQUE Scan PK_EMP



```
create unique index PK_EMP on EMP(empno)
```

```
select * from emp where empno = 9999;
```

Id	Operation	Name	Rows	Bytes	Cost	

0	SELECT STATEMENT		1	32	1	
1	TABLE ACCESS BY INDEX ROWID	EMP	1	32	1	
* 2	INDEX UNIQUE SCAN	PK_EMP	1			

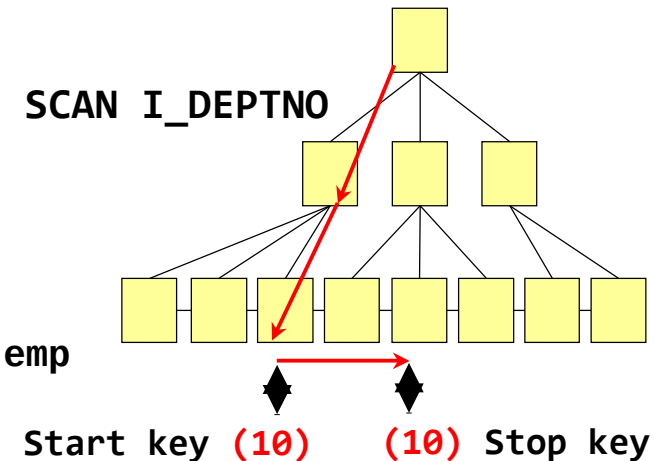
Predicate Information (identified by operation id):

2 - **access(EMP.EMPNO=9999)**

1.4 - Index Scan (Range)

```
create index I_DEPTNO on EMP(deptno)
```

```
select /*+ INDEX(EMP I_DEPTNO) */ * from emp
where deptno = 10 and sal > 1000;
```



Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		4	128	276
* 1	TABLE ACCESS BY INDEX ROWID	EMP	4	128	276
* 2	INDEX RANGE SCAN	I_DEPTNO	5		9

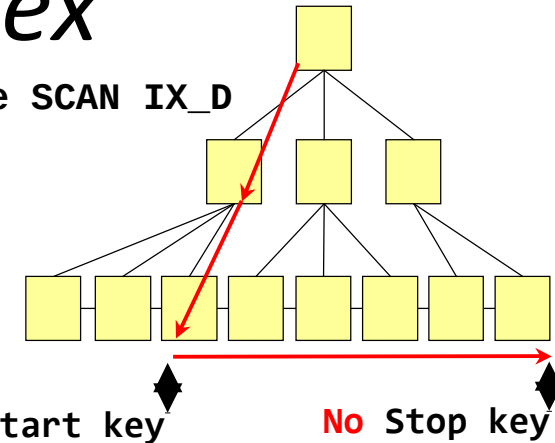
Predicate Information (identified by operation id):

- 1 - *filter(EMP.SAL>1000)* - Throw out non-matching rows
- 2 - *access(EMP.DEPTNO=10)* - means read data from Index, Table

Index Scan (Range)

Descending Index

Index Range SCAN IX_D



```
create index IX_D on EMP(deptno DESC)
```

```
select /*+ INDEX(EMP IX_D) */ * from emp
where deptno < 30 order by DESC;
```

Id	Operation	Name	Rows	Cost	
0	SELECT STATEMENT			1	
1	TABLE ACCESS BY INDEX ROWID	EMP	9	1	
* 2	INDEX RANGE SCAN	IX_D	1	2	

Predicate Information (identified by operation id):

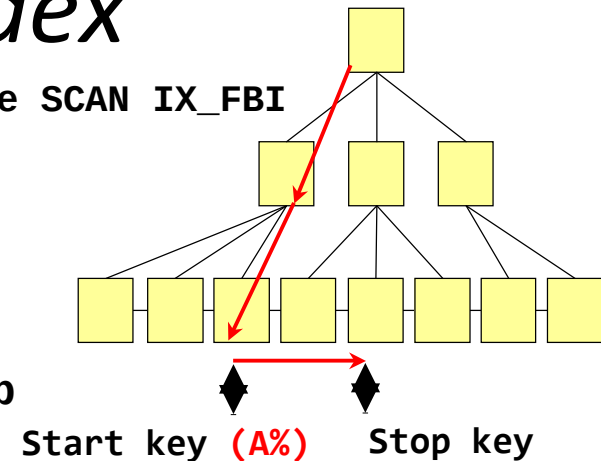
```
2 - access(SYS_OP_DESCEND(EMP.DEPTNO)>HEXTORAW('3EE0FF'))
    filter(SYS_OP_UNDESCEND(SYS_OP_DESCEND(EMP.DEPTNO))<30)
```

Index Scan (Range)

Function Based Index

```
create index IX_FBI on EMP(UPPER(ename));
```

```
select /*+ INDEX(EMP IX_FBI) */ * from emp
where upper(ENAME) like 'A%';
```



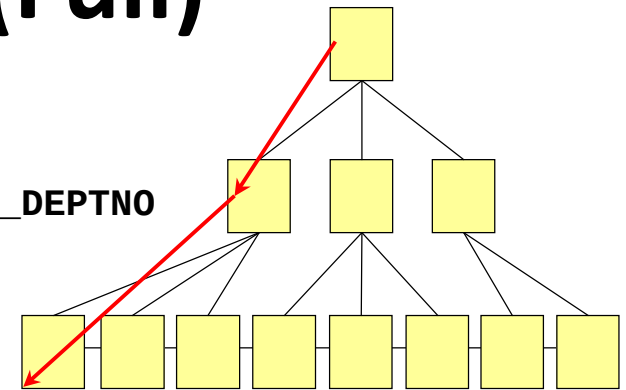
Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	1
1	TABLE ACCESS BY INDEX ROWID	EMP	1	1
* 2	INDEX RANGE SCAN	IX_FBI	1	2

Predicate Information (identified by operation id):

```
2 - access(UPPER(EMP.ENAME) LIKE 'A%')
    filter(UPPER(EMP.ENAME) LIKE 'A%')
```

1.5 - Index Scan (Full)

index Full Scan I_DEPTNO



```
select /*+ INDEX(EMP I_DEPTNO) */ * from emp
where sal > 1000
      and deptno is not null
order by deptno;
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		13	416	826
* 1	TABLE ACCESS BY INDEX ROWID	EMP	13	416	826
* 2	INDEX FULL SCAN	I_DEPTNO	14		26

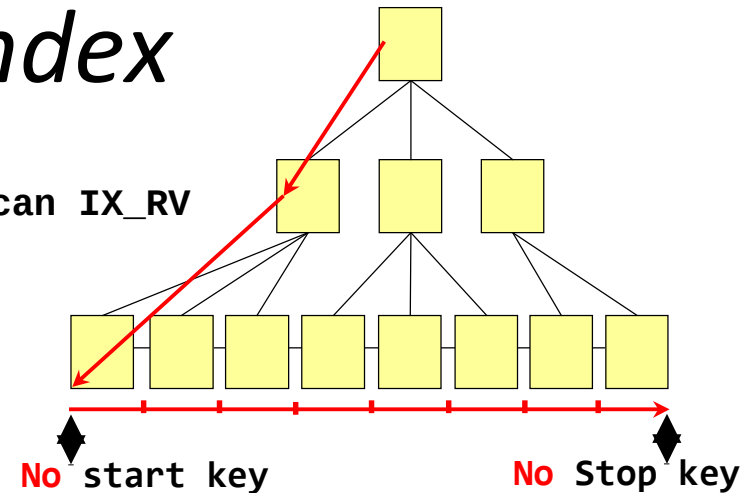
Predicate Information (identified by operation id):

- 1 - filter("EMP"."SAL">1000)
- 2 - filter("EMP"."DEPTNO" IS NOT NULL)

Index Scan (Full)

Reverse Key Index

index Full Scan IX_RV



```
select /*+ index(emp)*/ * from emp where empno > 7700;
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		6	358
1	TABLE ACCESS BY INDEX ROWID	EMP	6	358
* 2	INDEX FULL SCAN	IX_RV	6	26

Predicate Information (identified by operation id):

2 - filter(EMP.EMPNO>7700)

1.6 - Index Scan (Fast Full)

LEGEND:

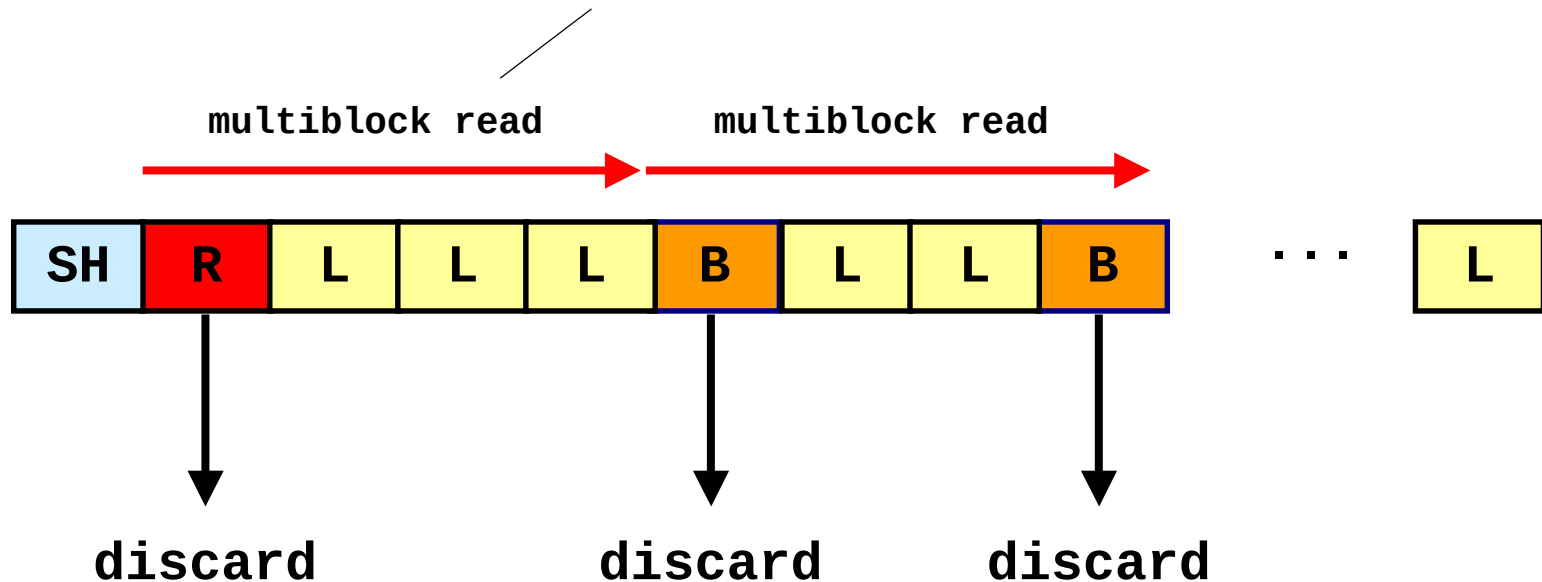
SH=segment header

R=root block

B=branch block

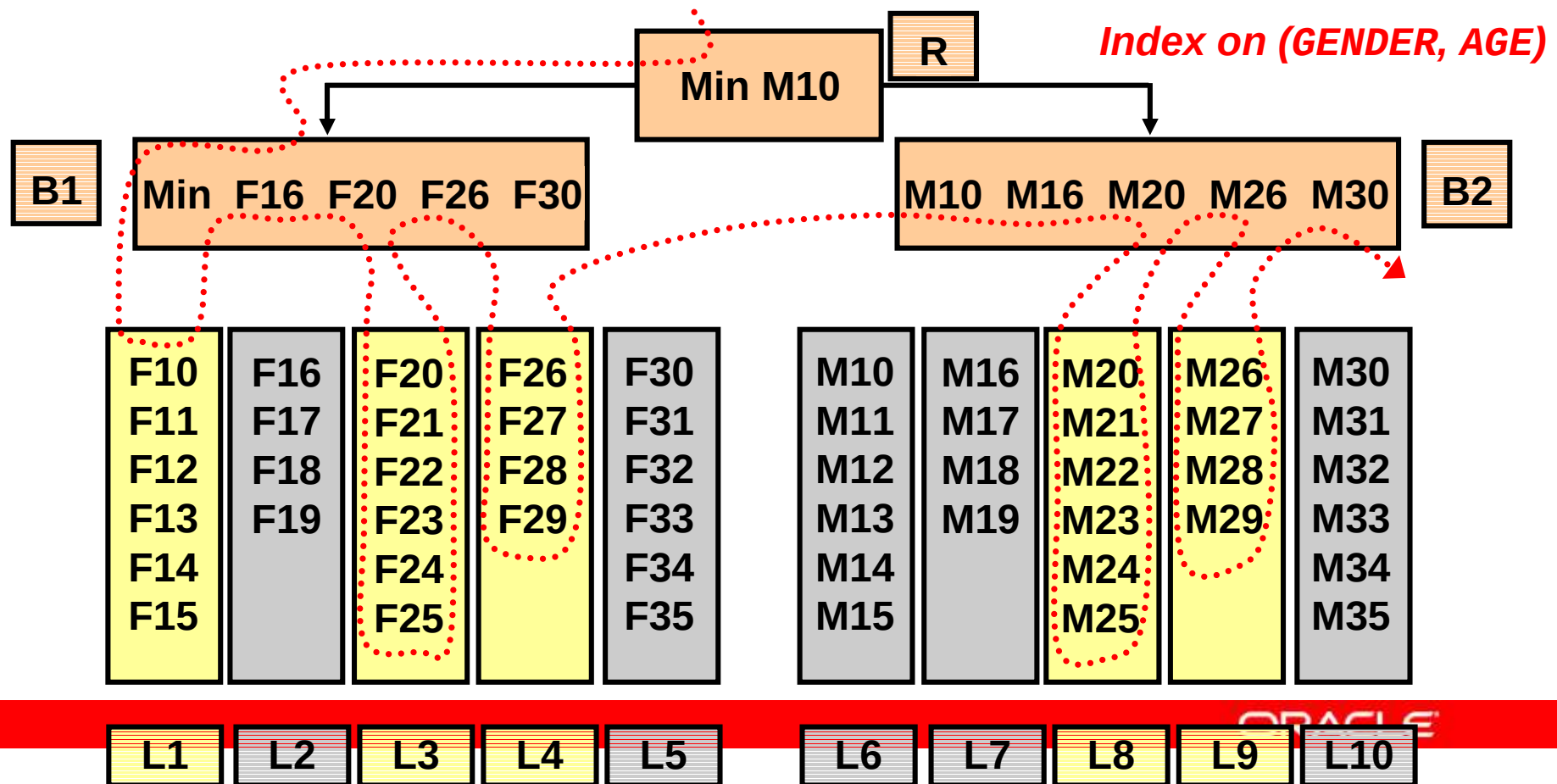
L=leaf block

db_file_multiblock_read_count = 4



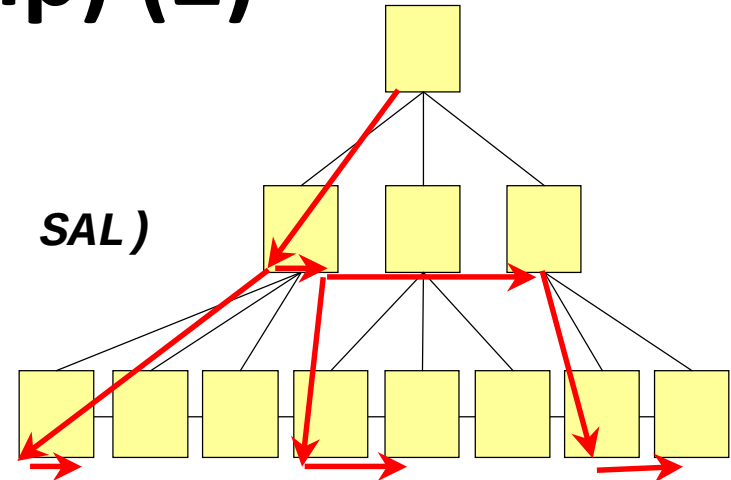
1.7 – Index Scan (Skip) (1)

SELECT * FROM employees WHERE age BETWEEN 20 AND 29



Index Scan (Skip) (2)

Index on (DEPTNO, SAL)



```
select /*+ index_ss(EMP IX_SS) */ * from emp where SAL < 1500;
```

Id	Operation	Name	A-Rows	Buffers
1	TABLE ACCESS BY INDEX ROWID	EMP	6	4
* 2	INDEX SKIP SCAN	IX_SS	6	2

```
2 - access("SAL"<1500)
    filter("SAL"<1500)
```

1.8 - Index Scan (Index Join)

```
create index I_ENAME on EMP(ename)
create index I_SAL on EMP(sal)
```

```
select /*+ INDEX_JOIN(e) */ ename, sal from emp e;
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		409	66
1	VIEW	index\$_join\$_001	409	66
* 2	HASH JOIN		409	
3	INDEX FAST FULL SCAN	I_ENAME	409	33
4	INDEX FAST FULL SCAN	I_SAL	409	33

Bitmap Indexes

Index Bitmap IX_B2
stored in B*Tree Index

Table
EMEA_PERF

ID	Name	G	C	Rowid
1	Jan Boonen	M	BE	41.389647.0
2	Andreas Walter	M	DE	41.389647.1
3	Lech Barwicki	M	FR	41.389647.2
4	Muriel FRY	F	FR	41.389647.3
5	Christian Boure	M	FR	41.389647.4
6	Thomas Guillemint	M	FR	41.389647.5
7	Dena Feinblat	F	IL	41.389647.6
8	Sijmen de Roo	M	NL	41.389647.7
9	Doody Atraktsi	M	IL	41.389647.8
10	James cremonini	M	UK	41.389648.0
11	Steve Dixon	M	UK	41.389648.1
12	Andrew Hambelton	M	UK	41.389648.2



Key	Start Rowid	End Rowid	Bitmap
BE	41.389647.0	41.389648.2	1 0 0 0 0 0 0 0 0 0 0 0
DE	41.389647.0	41.389648.2	0 1 0 0 0 0 0 0 0 0 0 0
FR	41.389647.0	41.389648.2	0 0 1 1 1 1 0 0 0 0 0 0
IL	41.389647.0	41.389648.2	0 0 0 0 0 0 1 0 1 0 0 0
NL	41.389647.0	41.389648.2	0 0 0 0 0 0 0 1 0 0 0 0
UK	41.389647.0	41.389648.2	0 0 0 0 0 0 0 0 0 1 1 1

Bitmap Indexes Example

Key Bitmap

BE	1	0	0	0	0	0	0	0	0	0	0	0	0
DE	0	1	0	0	0	0	0	0	0	0	0	0	0
FR	0	0	1	1	1	1	0	0	0	0	0	0	0
IL	0	0	0	0	0	0	0	1	0	1	0	0	0
NL	0	0	0	0	0	0	0	1	0	0	0	0	0
UK	0	0	0	0	0	0	0	0	0	1	1	1	0

← Bitmap for
key value FR

↑ Bit for 8th record is set because row
has value 'NL' in column COUNTRY

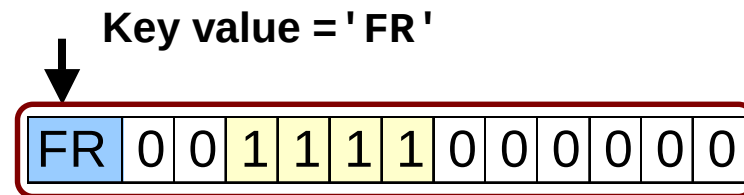
Key Bitmap

M	1	1	1	0	1	1	0	1	1	1	1	1	1
F	0	0	0	1	0	0	1	0	0	0	0	0	0

← Bitmap for
key value F

1.9 – Bitmap Index Access

```
SQL> SELECT *  
      2 FROM EMEA_PERF_TEAM T  
      3 WHERE country = 'FR';
```



```
SQL> SELECT * FROM EMEA_PERF_TEAM T WHERE country = 'FR';
```

Id	Operation	Name	Rows	Cost (%CPU)

0	SELECT STATEMENT			1 (100)
1	TABLE ACCESS BY INDEX ROWID	EMEA_PERF_TEAM	4	1 (0)
2	BITMAP CONVERSION TO ROWIDS			
*	BITMAP INDEX SINGLE VALUE	IX_B2		

Predicate Information (identified by operation id):

3 - access("COUNTRY"='FR')

1.10 - Combining Bitmap Indexes (1)

```
SQL> SELECT *  
2 FROM EMEA_PERF_TEAM T  
3 WHERE country = 'FR' or gender = 'F';
```

FR	0	0	1	1	1	1	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---	---

OR

F	0	0	0	1	0	0	1	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---



0	0	1	1	1	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---

1	TABLE ACCESS BY INDEX ROWID	EMEA_PERF_TEAM
2	BITMAP CONVERSION TO ROWIDS	
3	BITMAP OR	
*4	BITMAP INDEX SINGLE VALUE	IX_B1
*5	BITMAP INDEX SINGLE VALUE	IX_B2

4 - access("GENDER"='F')

5 - access("COUNTRY"='FR')

Combining Bitmap Indexes (2)

```
SQL> SELECT *  
2 FROM EMEA_PERF_TEAM T  
3 WHERE country = 'FR' and gender = 'M';
```

FR	0	0	1	1	1	1	0	0	0	0	0	0
----	---	---	---	---	---	---	---	---	---	---	---	---

AND

M	1	1	1	0	1	1	0	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---



0	0	1	0	1	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---

1	TABLE ACCESS BY INDEX ROWID	EMEA_PERF_TEAM
2	BITMAP CONVERSION TO ROWIDS	
3	BITMAP AND	
*4	BITMAP INDEX SINGLE VALUE	IX_B1
*5	BITMAP INDEX SINGLE VALUE	IX_B2

4 - access("GENDER"='M')

5 - access("COUNTRY"='FR')

Bitmap Operations

- BITMAP CONVERSION
 - Can be TO or FROM ROWIDS
- BITMAP INDEX
 - Single value lookup
 - Range scan
 - Full scan
- BITMAP MERGE
- BITMAP AND / OR
- BITMAP MINUS

Id	Operation

0	SELECT STATEMENT
1	TABLE ACCESS BY INDEX ROWID
2	BITMAP CONVERSION TO ROWIDS
3	BITMAP MINUS
4	BITMAP OR
*	BITMAP INDEX SINGLE VALUE
*	BITMAP INDEX SINGLE VALUE
*	BITMAP INDEX SINGLE VALUE



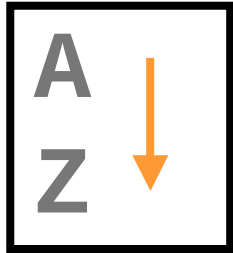
2 - Other Unary Operations

- SORT & HASH (GROUP BY – UNIQUE)
- BUFFER (SORT)
- IN LIST ITERATOR
- VIEW
- COUNT STOP KEY
- (MIN/MAX) / FIRST ROW



2.1 – SORT & HASH (1)

- **SORT**



- AGGREGATE: Single row from group function
- UNIQUE: To eliminate duplicates
- JOIN: Precedes a merge join
- GROUP BY, ORDER BY: For these operators

- **HASH**

- UNIQUE
- GROUP BY



SORT & HASH (2)

Don't rely on the **GROUP BY** or **DISTINCT** to **ORDER BY** returned rows.

If you want to have rows ordered, use **ORDER BY** clause.



2.2 – Buffer (sort)

- **BUFFER (SORT)**

```
select ename, emp.deptno, dept.deptno, dname
from   emp, dept
where  ename like 'A%';
```

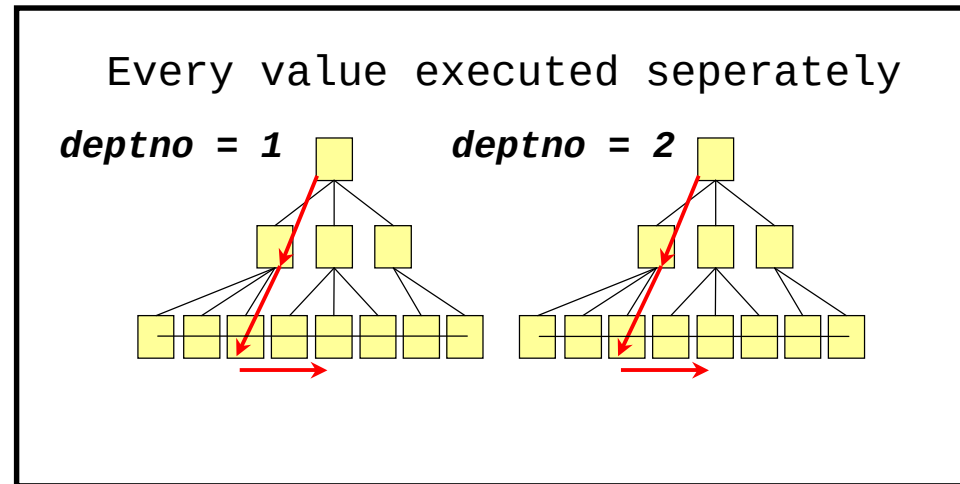
Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		21	10
1	MERGE JOIN CARTESIAN		21	10
* 2	TABLE ACCESS FULL	EMP	4	2
3	BUFFER SORT		5	8
4	TABLE ACCESS FULL	DEPT	5	2

```
2 - filter("EMP"."ENAME" LIKE 'A%')
```

2.3 – Inlist Iterator

```
create index I_DEPTNO
  on emp(deptno);

select * from emp
  where deptno in (1,2);
select * from emp
  where deptno = 1
     or deptno =2 ;
```



Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		5	2
1	INLIST ITERATOR			
2	TABLE ACCESS BY INDEX ROWID	EMP	5	2
* 3	INDEX RANGE SCAN	I_DEPTNO	5	1

Predicate Information (identified by operation id):

3 - access(DEPTNO=1 OR DEPTNO=2)

2.4 – View (1)

```
create view V as select /*+ NO_MERGE */ DEPTNO, sal
from emp ;

select * from V;
```

Id	Operation	Name	Rows	Cost (%CPU)

0	SELECT STATEMENT			3 (100)
1	VIEW	V	14	3 (0)
2	TABLE ACCESS FULL	EMP	14	3 (0)

View (2)

- View operation is executed separately
 - All rows from View are returned (Id=2)
 - This resultset is then joined to DEPT with nested loops

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT			7 (100)
1	NESTED LOOPS		3	7 (15)
2	VIEW		3	4 (25)
3	HASH GROUP BY		3	4 (25)
4	TABLE ACCESS FULL	EMP	14	3 (0)
5	TABLE ACCESS BY INDEX ROWID	DEPT	1	1 (0)
* 6	INDEX UNIQUE SCAN	PK_DEPT	1	1 (0)

Predicate Information (identified by operation id):

6 - access("V"."DEPTNO"="D"."DEPTNO")

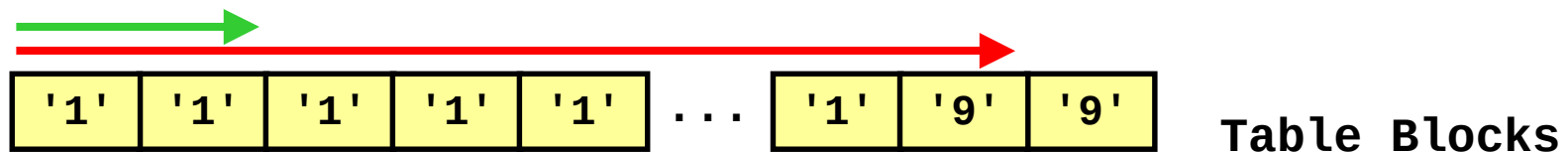
2.5 – Count Stop Key

- **COUNT STOPKEY**

- rownum < ~value~
- STOP previous operation when count is reached

- **EXAMPLE**

- Fill a table with plenty of '1' and only some '9' at the end.
- Run a query to output the ten first '1'
- Run same query with the ten first '9'
-  If the value in question is frequently occurring (e.g. '1') then only a small number of buffers are required (Buffers = 12)
-  If the value in question is NOT frequently occurring (e.g. '9'), then almost the whole table needs to be retrieved (Buffers = 3984)



2.6 – (Min/Max) / First Row

- Stop accessing the data after first value is returned
- That's an optimization introduced on 8i
- Works with Index Range / Full scan

```
select min(id) from t;          -> MIN(ID) = 1
```

Id	Operation	Name	A-Rows	Buffers
1	SORT AGGREGATE		1	5
2	INDEX FULL SCAN (MIN/MAX)	T_IND	1	5

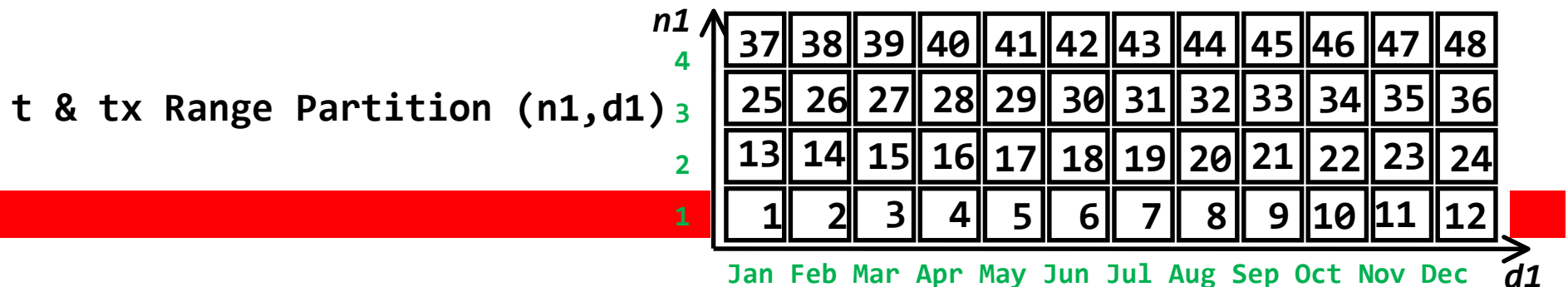


2.7 – Partition Access Path

- Only CBO seulement
- Partition (pruning)
 - compilation / execution
 - Depend on partition type
- Same Access Path for pruned partitions
- Partition wise join

2.7.1 – Partition Range (1)

- PARTITION RANGE SINGLE
 - WHERE `n1 = 3 AND d1 = to_date('2007-07-19',...);` → Pstart 31 Pstop 31
 - WHERE `n1 = :n AND d1 = to_date(:d1,'yyyy-MM-DD');` → Key Key
- PARTITION RANGE INLIST
 - WHERE `n1 in (1,3) AND d1 = to_date('2007-07-19',...);` → Key(I) Key(I)
- PARTITION RANGE ITERATOR
 - WHERE `n1 = 3` → 1 37
- PARTITION RANGE ALL
 - WHERE `n3 > 3000;` → 1 48
- PARTITION RANGE EMPTY
 - WHERE `n1 = 5 AND d1 = to_date('2007-07-19',...);` → INVALID INVALID
- PARTITION RANGE OR
 - WHERE `n1 = 3 OR d1 = to_date('2007-07-19',...);` → KEY(OR) KEY(OR)
- PARTITION RANGE SUBQUERY
 - WHERE `tx.d1 = t.d1 AND tx.n1 = t.n1 and tx.id=3;` → KEY(SQ) KEY(SQ)



2.7.1 – Partition Range (2)

- PARTITION RANGE SUBQUERY
 - WHERE **tx.d1 = t.d1 AND tx.n1 = t.n1** and tx.id=3; → Pstart Pstop
KEY(SQ) KEY(SQ)
- PARTITION RANGE ITERATOR
 - WHERE **n1 = 3** → 25 37
- PARTITION RANGE Multi Column → KEY(MC) KEY(MC)
 - WHERE **d1 = to_date('2007-07-19', 'yyyy-MM-DD')**;

t & tx

Range Partition (n1,d1)

n1 ↑	37	38	39	40	41	42	43	44	45	46	47	48
	25	26	27	28	29	30	31	32	33	34	35	36
	13	14	15	16	17	18	19	20	21	22	23	24
	1	2	3	4	5	6	7	8	9	10	11	12
	→ d1											

2.7.2 – Partition Hash

- PARTITION HASH SINGLE
 - WHERE n1 = 3 AND d1 = to_date('2007-07-19',...); → Pstart 23 Pstop 23
 - WHERE n1 = :n AND d1 = to_date(:d1,'yyyy-MM-DD'); → Key Key
- PARTITION HASH INLIST
 - WHERE n1 in (1,3) AND d1 = to_date('2007-07-19',...); → Key(I) Key(I)
 - WHERE n1 = 1 AND d1 in (to_date('2007-07-19',...);
- PARTITION HASH ALL → 1 32
 - WHERE n1 = 3 ;
- PARTITION HASH SUBQUERY → KEY(SQ) KEY(SQ)
 - WHERE tx.d1 = t.d1 AND tx.n1 = t.n1 and tx.id=3;

t & tx

PARTITION BY HASH (n1, d1)

PARTITIONS 32

25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

2.7.3 – Partition List

- | | | Pstart | Pstop |
|--|---|---------|---------|
| • PARTITION LIST SINGLE | | | |
| • WHERE n1 = 3 AND d1 = to_date('2007-07-19',...); | → | 31 | 31 |
| • WHERE n1 = :n AND d1 = to_date(:d1,'yyyy-MM-DD'); | → | Key | Key |
| • PARTITION LIST INLIST | | | |
| • WHERE n1 = (1,3) AND d1 = to_date('2007-07-19',...); | → | Key(I) | Key(I) |
| • PARTITION LIST ALL | | | |
| • WHERE n3 > 3000; | → | 1 | 48 |
| • PARTITION LIST SUBQUERY | | | |
| • WHERE tx.d1 = t.d1 AND tx.n1 = t.n1 and tx.id=3; | → | KEY(SQ) | KEY(SQ) |
| • PARTITION LIST EMPTY | | | |
| • WHERE n1 = 5 AND d1 < to_date('2007-07-19',...); | → | INVALID | INVALID |

t & tx List Partition (n1)



3 Joins

- A join defines the relationship between 2 row sources.
- A join is a method of combining data from 2 data sources
- It is controlled by join predicates which define how the objects are related

```
SELECT e.ename, d.dname  
FROM   emp e, dept d  
WHERE  e.deptno = d.deptno  
AND    (e.job = 'ANALYST'  
OR      e.empno = 9999);
```

Nonjoin predicate

Join predicate

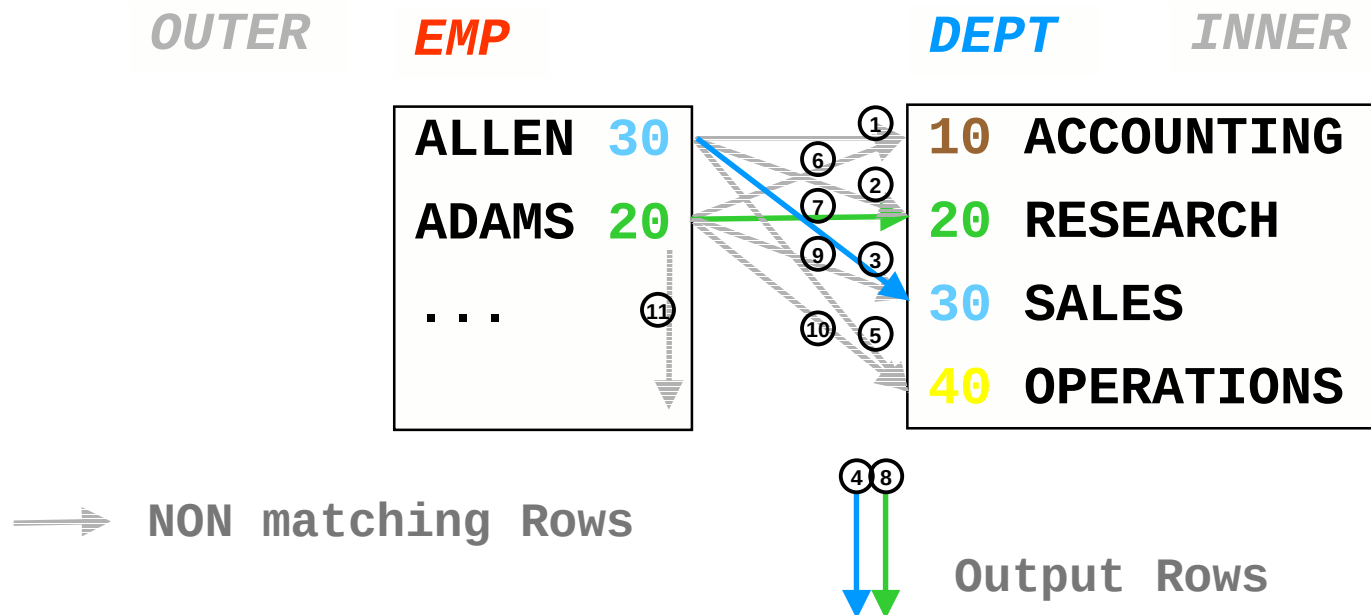


3.1 - Join Methods

- Nested Loops
- Merge Join
- Hash Join

3.1.1 - Nested Loop (1)

```
SQL> select ename, e.deptno, d.deptno, dname
2   from emp e, dept d
3   where e.deptno = d.deptno
4     and ename like 'A%'
```

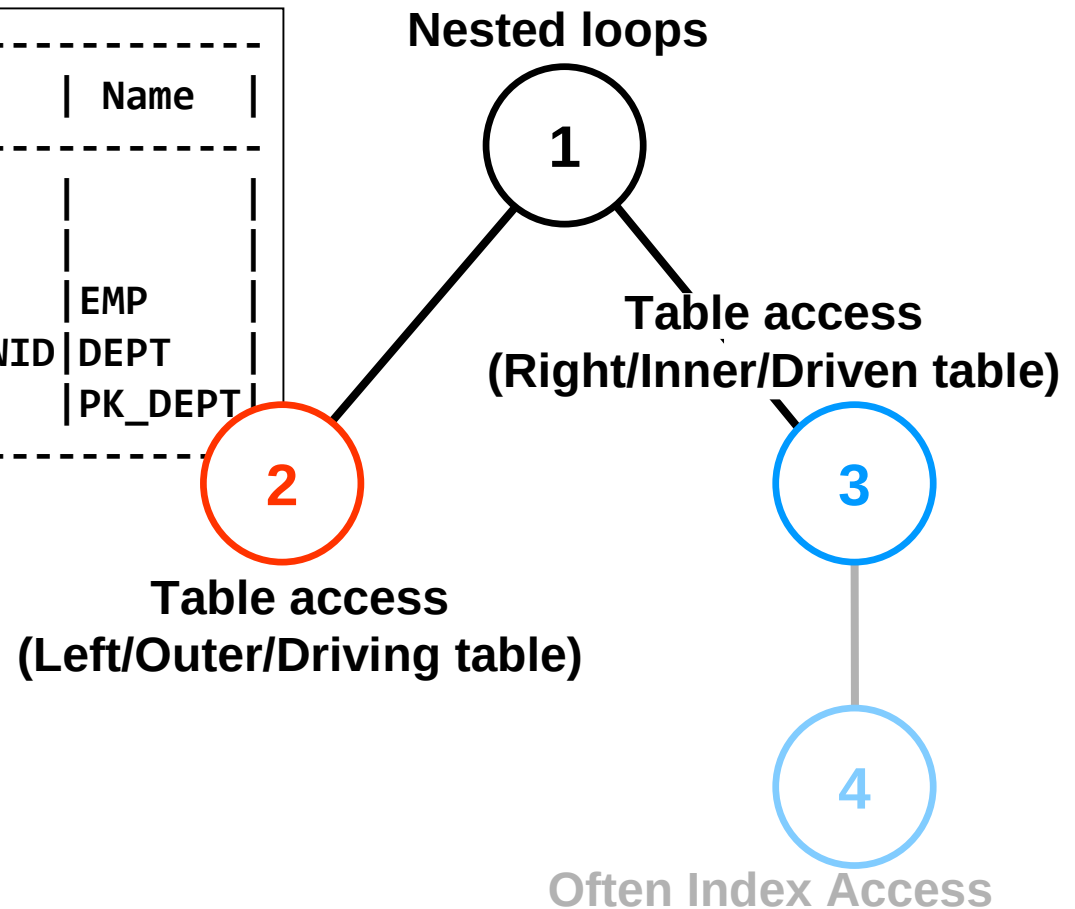


Nested Loop Join Plan (2)

Id	Operation	Name

0	SELECT STATEMENT	
1	NESTED LOOPS	
* 2	TABLE ACCESS FULL	EMP
3	TABLE ACCESS BY INDEX ROWID	DEPT
* 4	INDEX UNIQUE SCAN	PK_DEPT

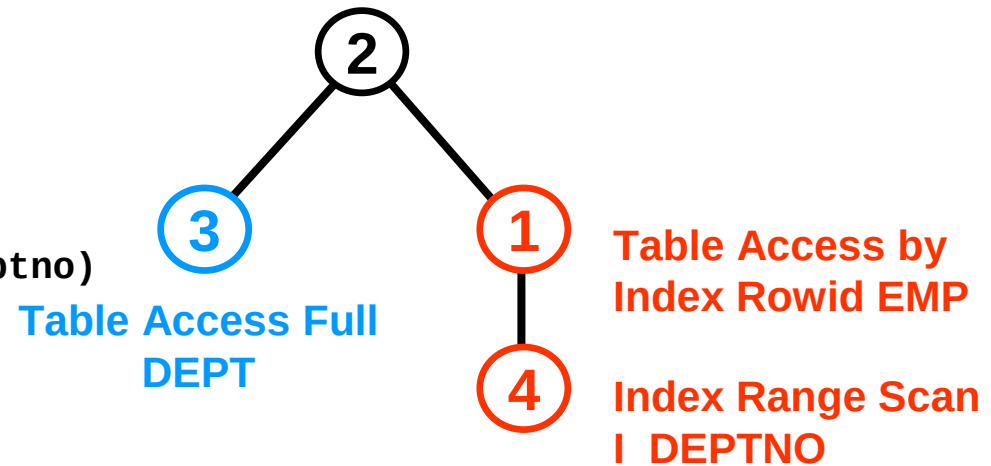
2 - filter(E.ENAME LIKE 'A%')		
4 - access(E.DEPTNO=D.DEPTNO)		



Nested Loop – Prefetching (3)

```
create index I_DEPTNO on EMP(deptno)
```

```
select empno, ename, dname
from emp e, dept d
where e.deptno= d.deptno ;
```



Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		14	6
1	TABLE ACCESS BY INDEX ROWID	EMP	4	1
2	NESTED LOOPS		14	6
3	TABLE ACCESS FULL	DEPT	4	2
* 4	INDEX RANGE SCAN	I_DEPTNO	5	

4 - access(EMP.DEPTNO=DEPT.DEPTNO)

3.1.2 - Sort-Merge (1)

EMP

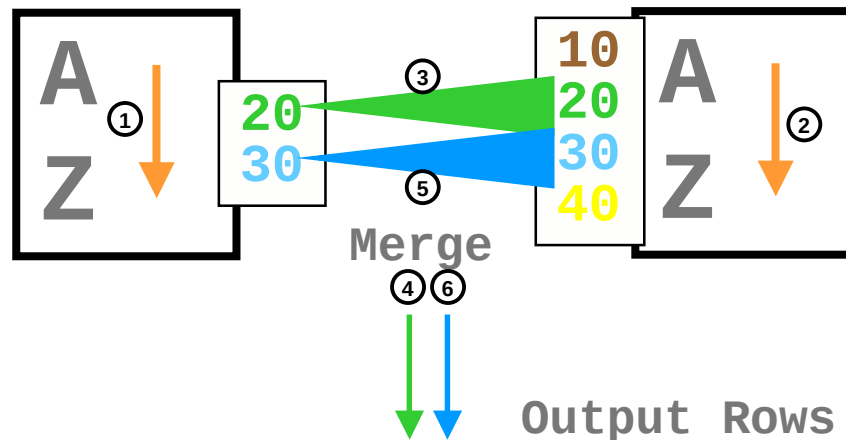
ALLEN	30
ADAMS	20
...	

Sort Join Key

DEPT

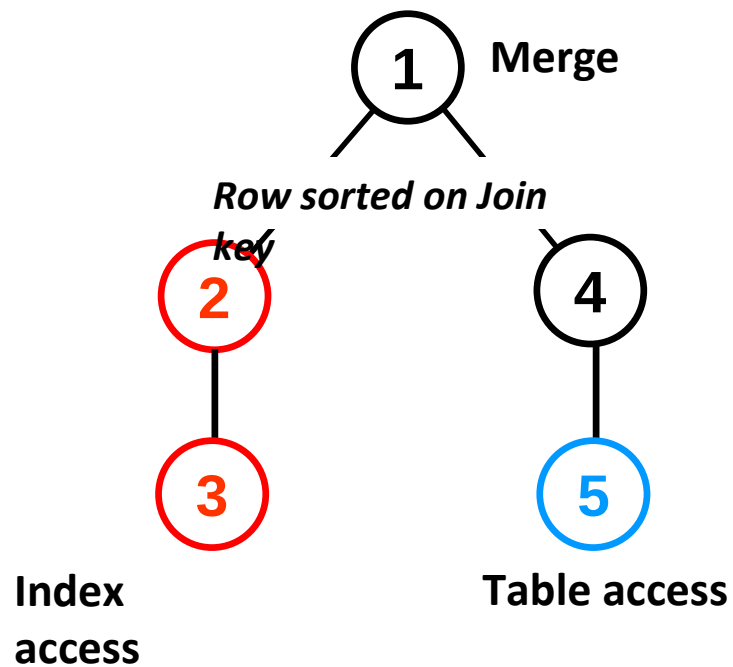
20	RESEARCH
30	SALES
10	ACCOUNTING
40	OPERATIONS

Sort Join Key



```
select ename
, d.deptno
, dname
from emp e
, dept d
where e.deptno = d.deptno
and ename like 'A%'
```

Sort-Merge Join Plan (2)

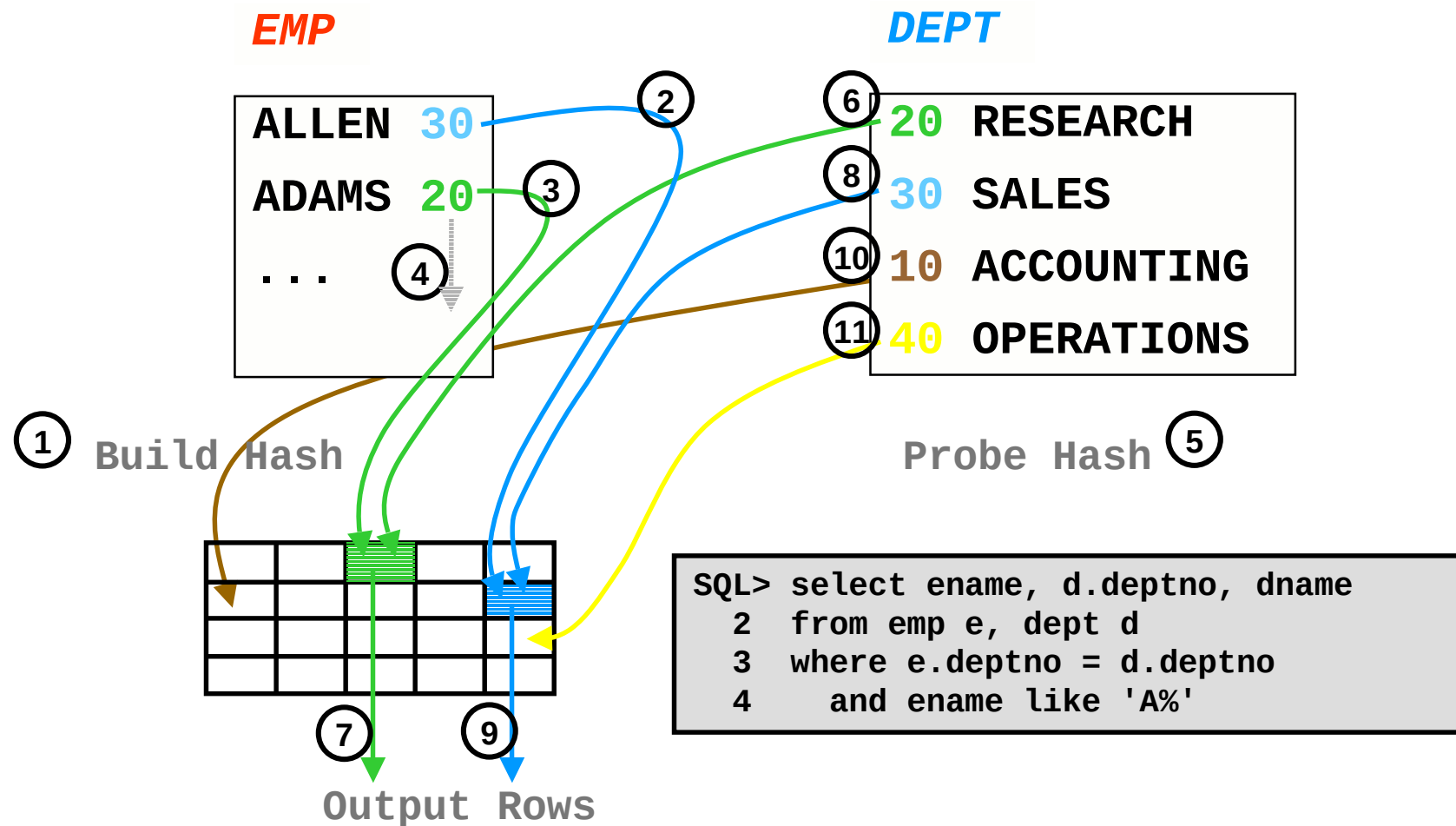


Id	Operation	Name
1	MERGE JOIN	
* 2	TABLE ACCESS BY INDEX ROWID	EMP
3	INDEX FULL SCAN	I_DEPTNO
* 4	SORT JOIN	
5	TABLE ACCESS FULL	DEPT

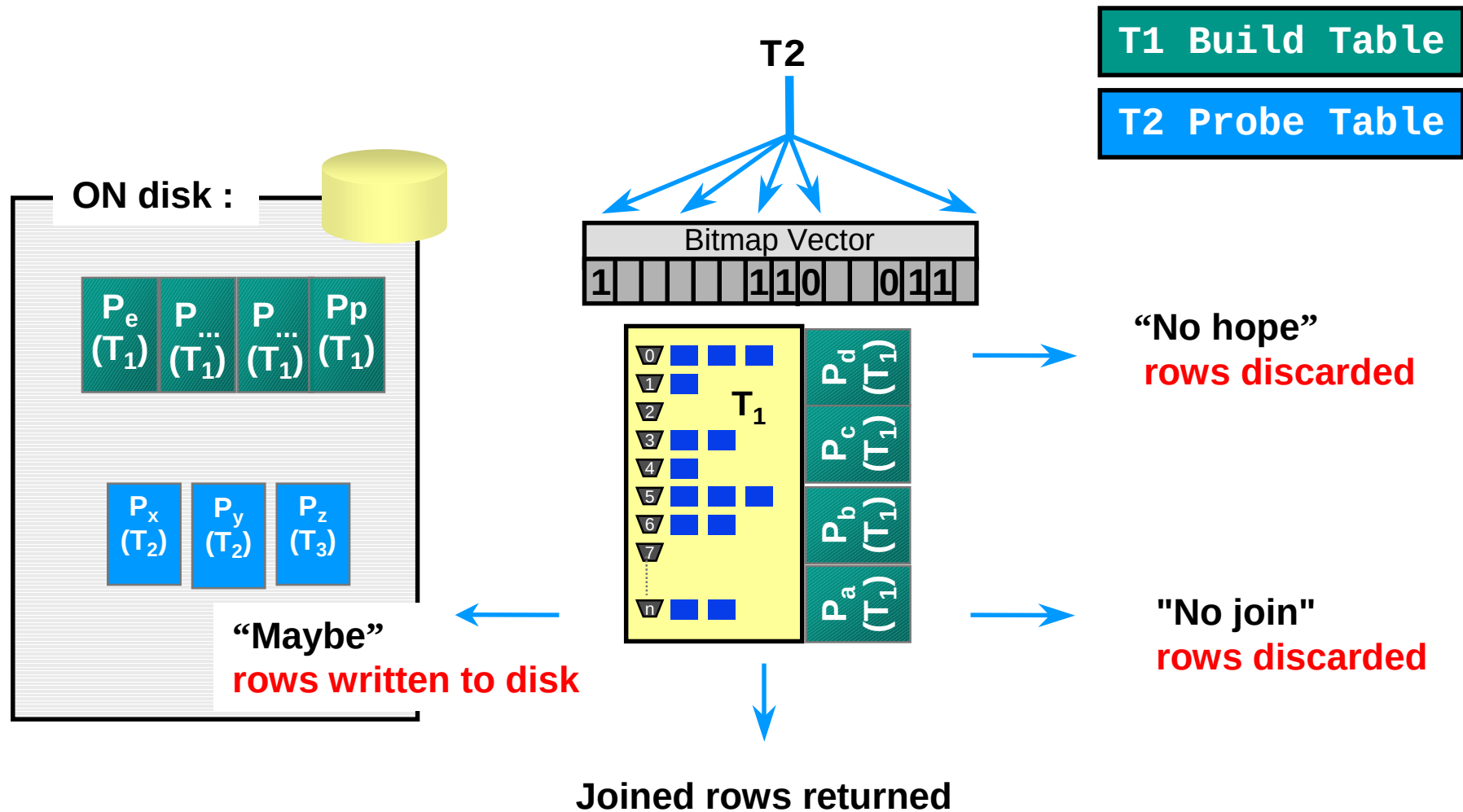
2 - filter(ENAME LIKE 'A%')

4 - access(E.DEPTNO=D.DEPTNO)
filter(E.DEPTNO=D.DEPTNO)

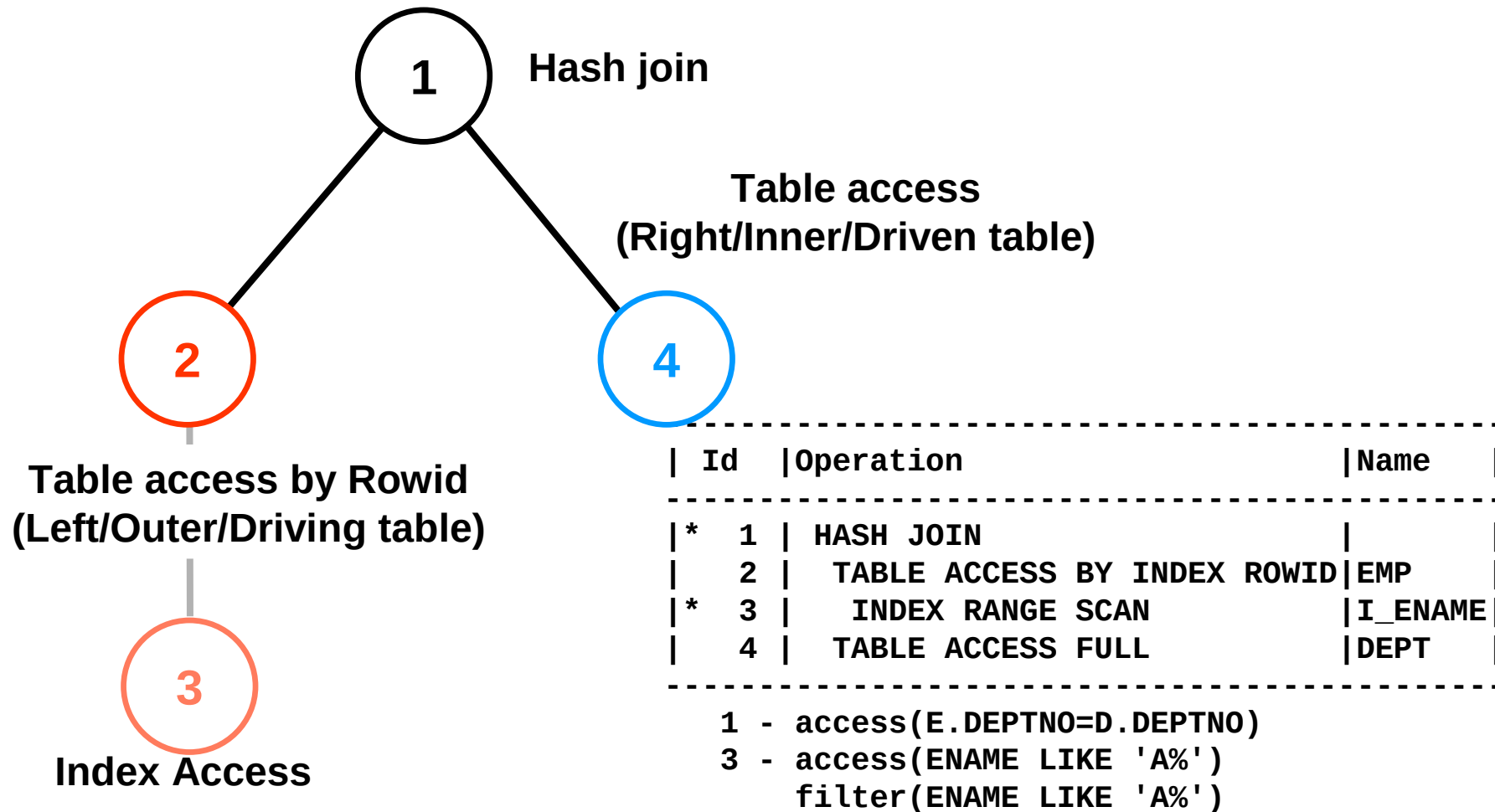
3.1.3 - Hash Join (1)



Hash Join Detail (2)

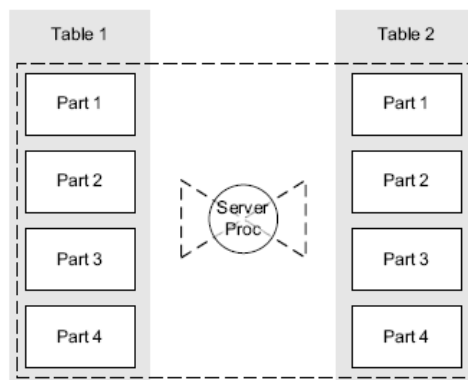


Hash Join Plan (3)

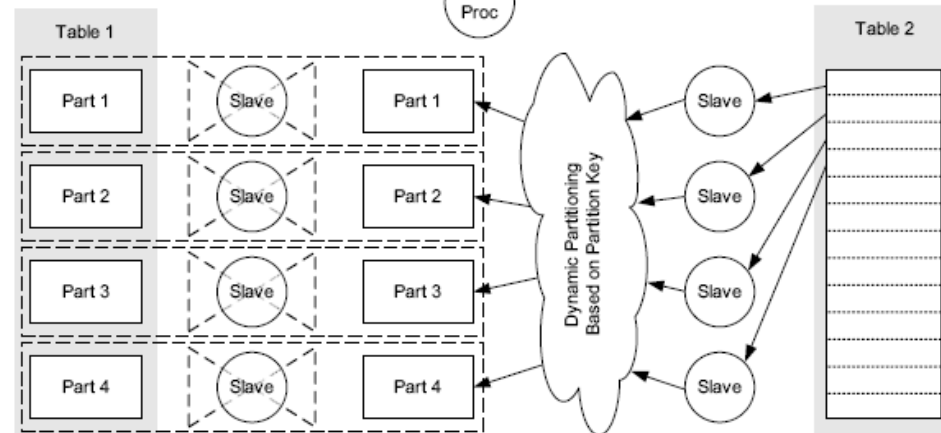


Partition-Wise Join

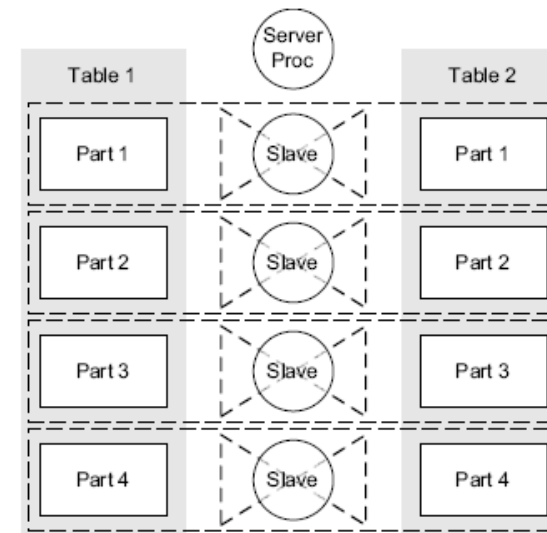
No Partition-Wise



Partial Partition-Wise



Full Partition-Wise



3.1.4 - Cartesian "Join"

```
SQL> select ename, e.deptno, d.deptno, dname  
2   from emp e, dept d where ename like 'A%'
```

EMP

BLAKE	30
ALLEN	30
ADAMS	20
SMITH	20
WARD	30
JONES	20
MARTIN	30
CLARK	10
SCOTT	20
KING	10
TURNER	30
JAMES	30
FORD	20
MILLER	10

DEPT

10	ACCOUNTING
20	RESEARCH
30	SALES
40	OPERATIONS

→ Returned Rows(8)

Id	Operation	Name
0	SELECT STATEMENT	
1	MERGE JOIN CARTESIAN	
* 2	TABLE ACCESS FULL	EMP
3	BUFFER SORT	
4	TABLE ACCESS FULL	DEPT
2 - filter(EMP.ENAME LIKE 'A%')		



3.2 - Join Types

- A join operation (whatever the join method) combines the output from two row sources and returns one resulting row source.
- Join operation types include the following :
 - Join (Equijoin – Non-equijoin)
 - Outer Join
 - Semi join - EXISTS Subquery
 - Anti Join - NOT IN Subquery
 - Star Join (Optimisation)



3.2.1 - Join

- Equi join

```
SELECT e.ename, d.dname FROM emp e,  
dept d where e.deptno = d.deptno;
```

- NON Equi join

```
SELECT e.ename, e.sal, s.grade FROM  
emp e, salgrade s where e.sal between  
s.losal and s.hisal;
```

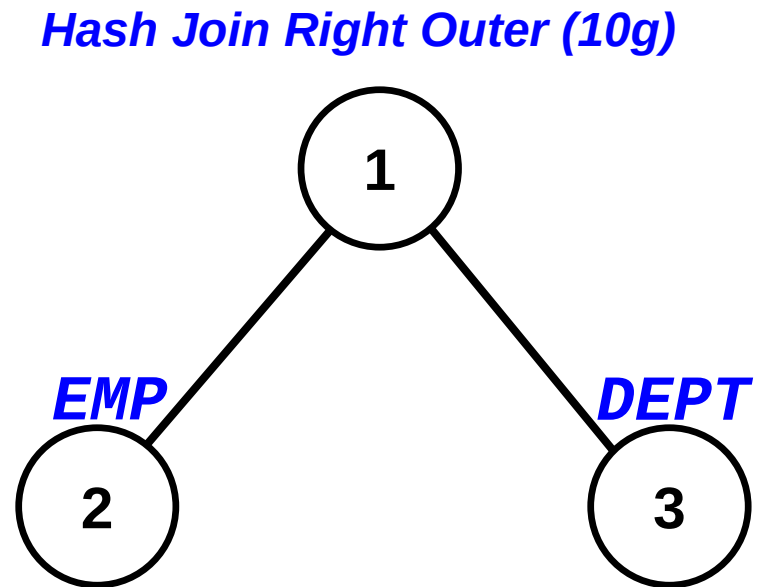
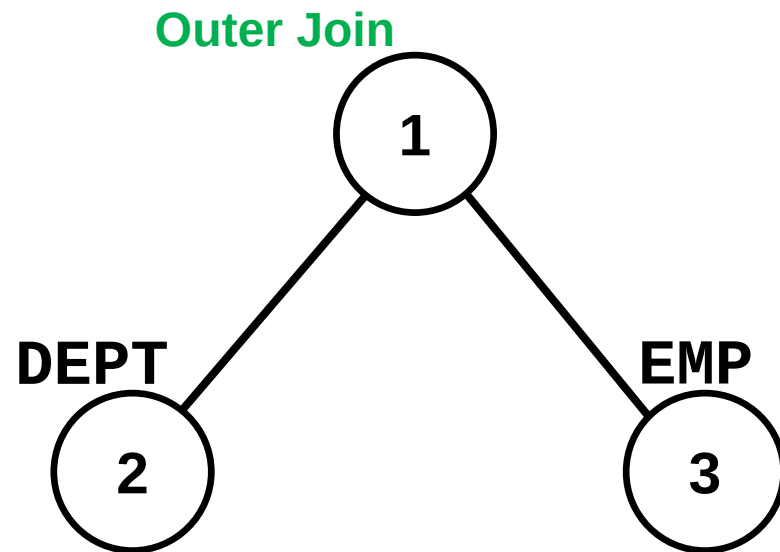
3.2.2 - Outer Joins

- A join returns rows where criteria match the specified join condition:
 - where emp.deptno = dept.deptno
- An outer join also returns a row if no match is found:
 - where emp.deptno(+)=dept.deptno

DEPTNO	DNAME	EMPNO	ENAME
10	ACCOUNTING	7782	CLARK
30	SALES	7900	JAMES
40	OPERATIONS		

1.3.2 - Outer Joins (2)

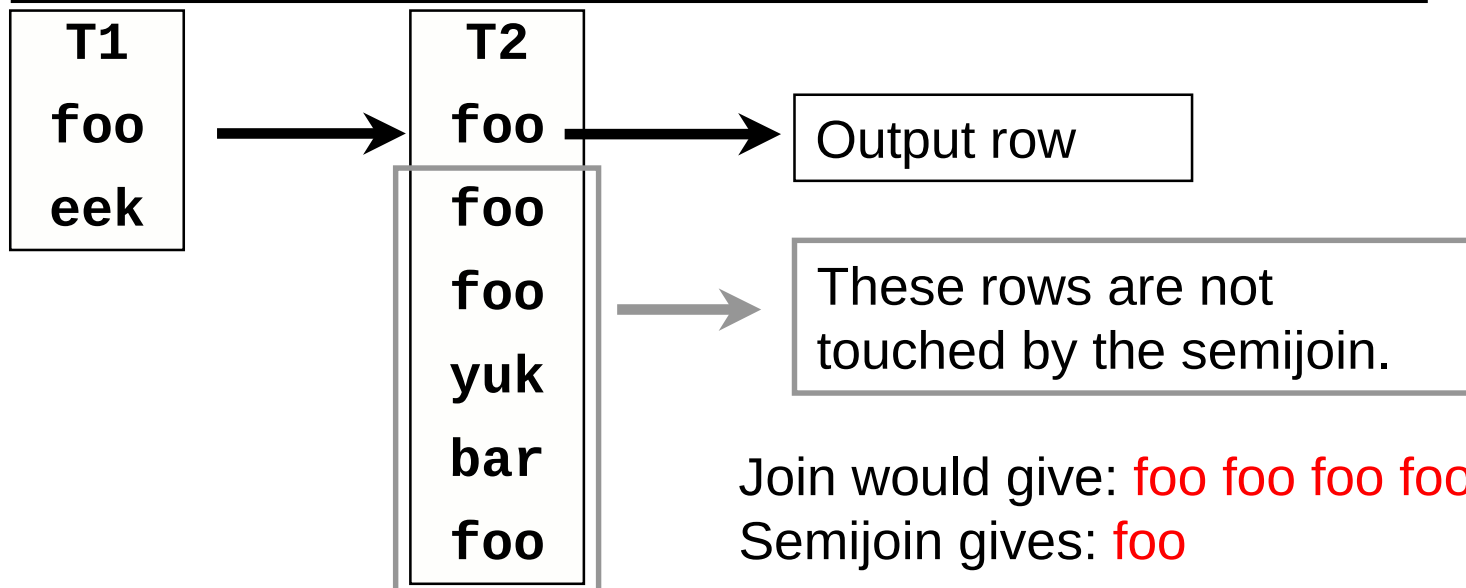
- where `emp.deptno(+) = dept.deptno`
 - outer join
 - Hash Join Right Outer



3.2.3 - Semi-Joins

Semijoins only look for the first match

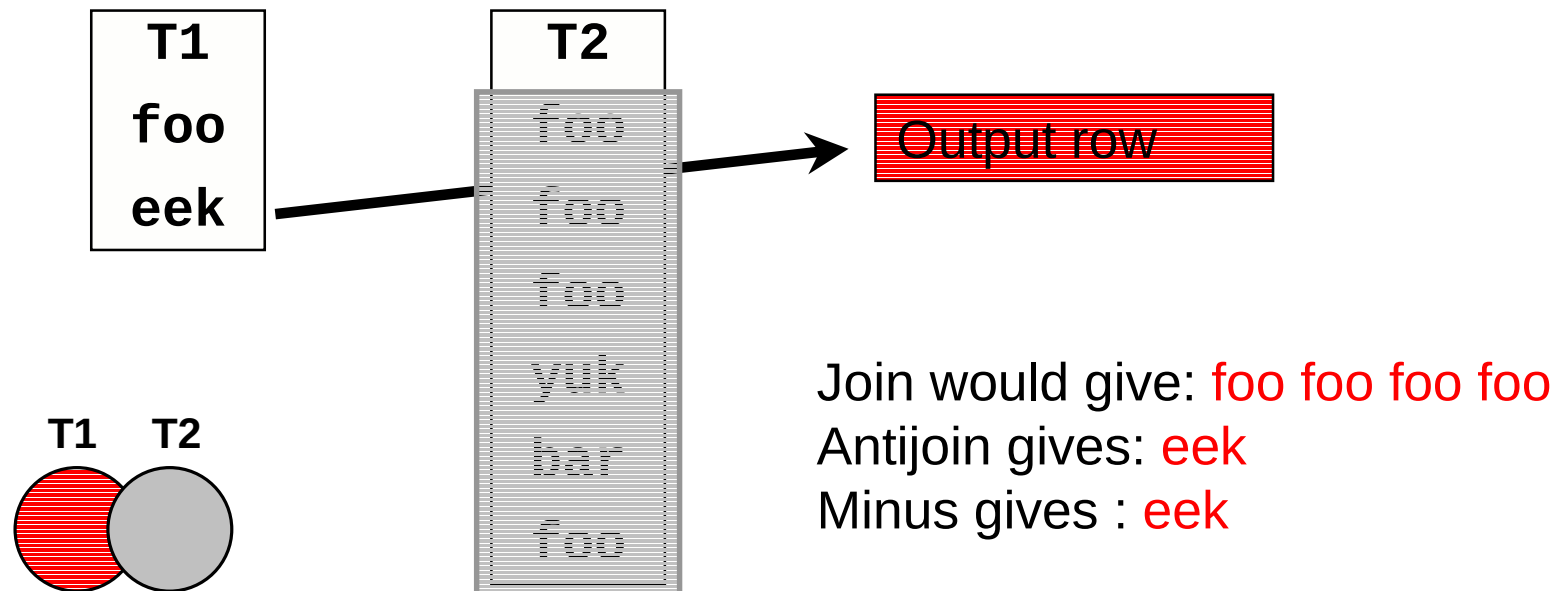
```
SQL> select c1 from t1
2   where c1 = 'foo' AND
3         exists (select null from t2
4                 where t1.c1 = t2.c1)
```



3.2.4 - Antijoins

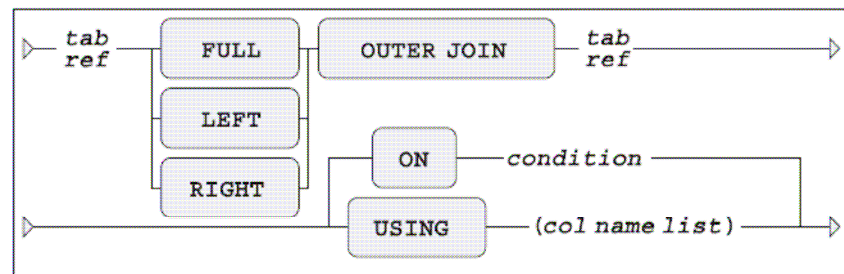
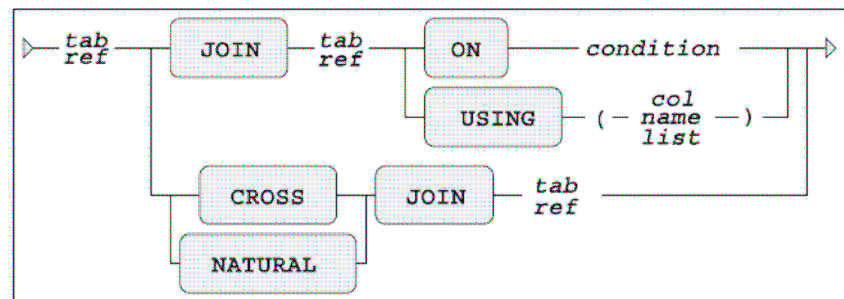
```
SQL> select c1 from t1  
2 where c1 not in (select c1 from t2)
```

Reverse of what would have been returned by a join: in case of a match the Oracle server does *not* return a row.



3.2.5 - ANSI Join Syntax

- Cross / Natural Join
- Left / Right / Full Outer Join



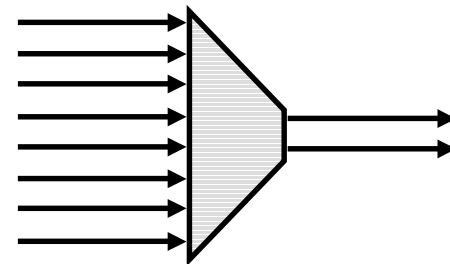


4 - Other N-ary Operations

- **FILTER**
- **CONCATENATION**
- **CONNECT BY**
- **UNION ALL / UNION**
- **INTERSECT**
- **MINUS**

4.1 – Filter (1)

- Accepts a set of rows
- Eliminates some of them
- Returns the rest

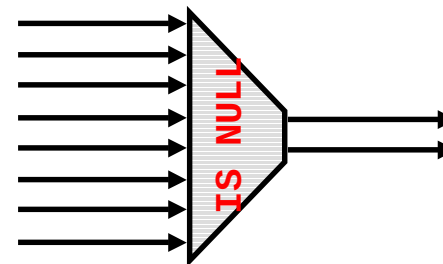


Filter (2)

```
select deptno, dname from dept
where not exists (select 1 from emp
                  where deptno=dept.deptno);
```

Id	Operation	Name	Starts	E-Rows	A-Rows
* 1	FILTER		1		1
2	TABLE ACCESS FULL	DEPT	1	4	4
* 3	INDEX RANGE SCAN	I_DEPTNO	4	2	3

1 - filter(**IS NULL**)
3 - access(EMP.DEPTNO=:B1)



filter is [**NOT EXISTS (select 1 from emp
where emp.deptno = :b1)**]

4.2 - Concatenation

```
create index I_SAL on emp(sal);
create index I_DEPTNO on emp(deptno);
```

```
select /*+ USE_CONCAT */ * from emp where deptno =1 or sal =2 ;
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		3	111	4
1	CONCATENATION				
2	TABLE ACCESS BY INDEX ROWID	EMP	1	37	2
* 3	INDEX RANGE SCAN	I_SAL	1		1
* 4	TABLE ACCESS BY INDEX ROWID	EMP	2	74	2
* 5	INDEX RANGE SCAN	I_DEPTNO	2		1

```
3 - access(SAL=2)
** 4 - filter(LNNVL(SAL=2))
5 - access(DEPTNO=1)
```

```
** LNNVL generated by 'OR'
```

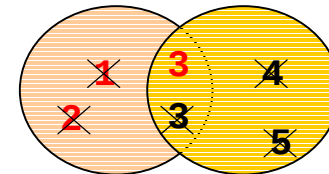


4.3 - Connect by

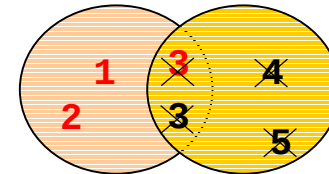
- connect by "without filtering"
 - 2 child operations
 - start with
 - Connect By
- connect by "with filtering"
 - 3 child operations
 - start with
 - Connect By
 - switch to no filtering, only if PGA is too small

4.4 - Intersect - Minus - Union (all)

- **INTERSECT**

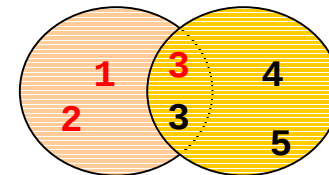


- **MINUS**



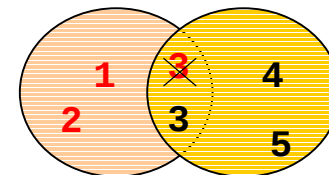
- **UNION ALL**

- Accepts sets of rows
- Applies union rules to determine the output
- Does not remove duplicates



- **UNION**

- Accepts sets of rows
- Applies union rules to determine the output
- removes duplicates





Summary

- At the end of the part, you should have learned :
 - The meaning of usual operations
 - Be familiar with very simple execution plan