7

Optimizer: Table and Index Access Paths

Objectives

After completing this lesson, you should be able to:

- Describe the SQL operators for tables and indexes
- List the possible access paths
- Describe common observations

Row Source Operations

- Unary operations
  - Access Path
- Binary operations
  - Joins
- N-ary operations

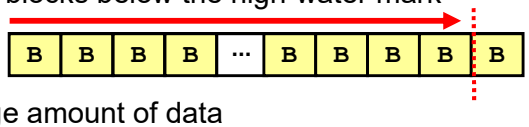
A row source is a set of rows returned by a step in the execution plan.

Main Structures and Access Paths

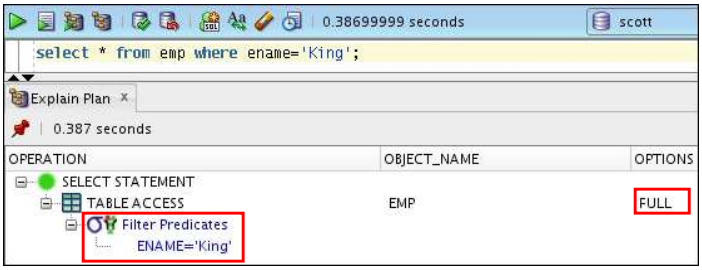
Structures	Access Paths
Tables	1. Full Table Scan 2. ROWID Scan 3. Sample Table Scan
Indexes	4. Index Scan (Unique) 5. Index Scan (Range) 6. Index Scan (Full) 7. Index Scan (Fast Full) 8. Index Scan (Skip) 9. Index Scan (Index Join) 10. Using Bitmap Indexes 11. Combining Bitmap Indexes

Full Table Scan

- Performs multiblock reads (here `DB_FILE_MULTIBLOCK_READ_COUNT = 4`)
- Reads all formatted blocks below the high-water mark



- May filter rows
- Is faster than index range scans for large amount of data



Full Table Scan Behavior in 11*g*R2

- Prior to Oracle Database 11*g*R2, FTS access path read all the blocks of a table (or index fast full scan) under the high-water mark into the buffer cache unless either the "_serial_direct_read" hidden parameter is set to true or the table/index have default parallelism set.
- In Oracle Database 11*g*R2, Oracle will automatically decide whether to use direct path reads bypassing buffer cache for serial FTS.
- Blocks read into the database cache as the result of a full scan of a large table are treated differently from other types of reads. The blocks are immediately available for reuse to prevent the scan from effectively cleaning out the buffer cache.

Full Table Scans

Common questions:

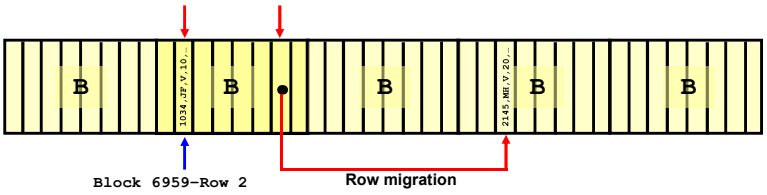
- Are all full table scans bad?
- At what percentage of data does the optimizer consider the Full Table Scan as the most efficient method to retrieve the data from a table (20%+, 30%+, or 50%+, and so on)?

When to use:

- No suitable index
- Low selectivity filters (or no filters)
- Small table
- High degree of parallelism
- Full table scan hint: `FULL (<table name>)`

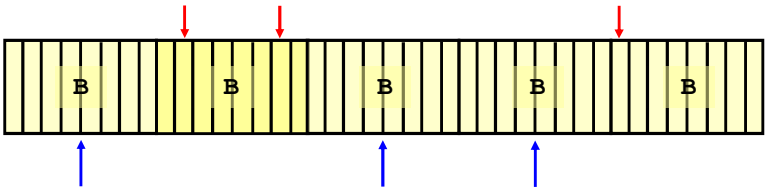
ROWID Scan

```
select * from scott.emp where rowid='AAAQ+LAAEAAAAAfAAJ';

-------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |    37 |     1|
|   1 |  TABLE ACCESS BY USER ROWID| EMP  |     1 |    37 |     1|
-------------------------------------------------------------------
```



Sample Table Scans

```
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)|
--------------------------------------------------------------------------
```



Quiz

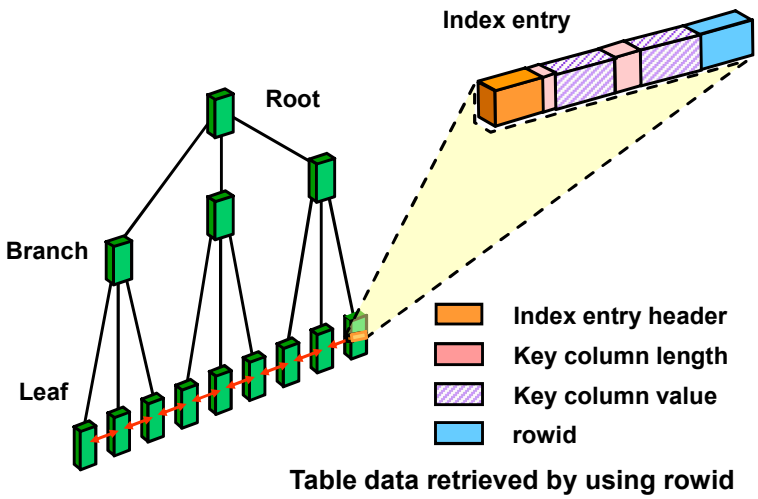
A full table scan sequentially reads all rows from a table and filters out those that do not meet the selection criteria.

a. True
b. False

Indexes: Overview

- Storage techniques:
  - B*-tree indexes: The default and the most common
    - Normal
    - Function based: Precomputed value of a function or expression
    - Index-organized table (IOT)
  - Bitmap indexes
  - Cluster indexes: Defined specifically for cluster
- Index attributes:
  - Key compression
  - Reverse key
  - Ascending, descending
- Domain indexes: Specific to an application or cartridge

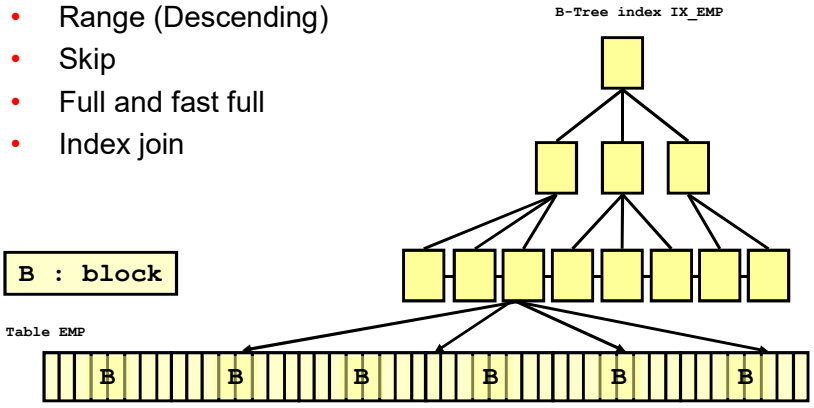
Normal B*-tree Indexes



Index Scans

Types of index scans:

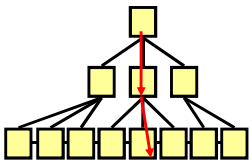
- Unique
- Range (Descending)
- Skip
- Full and fast full
- Index join



Index Unique Scan

The optimizer considers:

- If a SQL statement contains `UNIQUE` or a `PRIMARY KEY` constraint.
- When all the columns of a unique (B*-tree) index are specified with equality conditions.

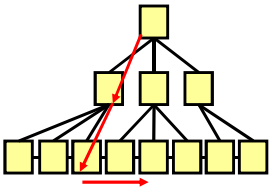



```
create unique index PK_EMP on EMP(empno)

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

Index Range Scan

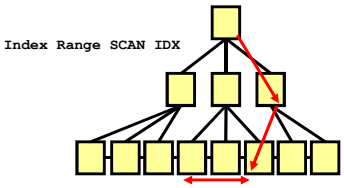
The optimizer considers:

- When the optimizer finds one or more leading columns of an index specified in conditions and any combination of the preceding conditions.
- It can use unique or nonunique indexes.
- It can avoid sorting when index columns constitute the `ORDER BY/GROUP BY` clause and the indexed columns are `NOT NULL` because otherwise they are not considered.




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

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

Index Range Scan: Descending



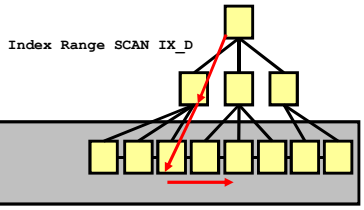

```
select * from emp where deptno>20 order by deptno desc;
```

Script Output | Explain Plan

0.426 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	7
TABLE ACCESS	EMP	BY INDEX ROWID	2	7
INDEX	I_DEPTNO	RANGE SCAN DESCENDING	1	7
Access Predicat				
DEPTNO>20				

Descending Index Range Scan




```
drop index I_Deptno;
create index IX_D on EMP(deptno desc);
select * from emp where deptno <30;
```

Script Output | Explain Plan

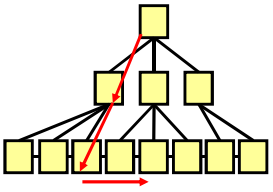
0.011 seconds

OPERATION	OBJECT_NA...	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	9
TABLE ACCESS	EMP	BY INDEX ROWID	2	9
INDEX	IX_D	RANGE SCAN	1	1
Access Predicates				
SYS_OP_DESCEND(DEPTNO)>HEXTORAW('3EE0FF')				
Filter Predicates				
SYS_OP_UNDESCEND(SYS_OP_DESCEND(DEPTNO))<30				

Index Range Scan: Function-Based

The optimizer considers:

- When frequently executed SQL statements include transformed columns, or columns in expressions, in a `WHERE` or `ORDER BY` clause.

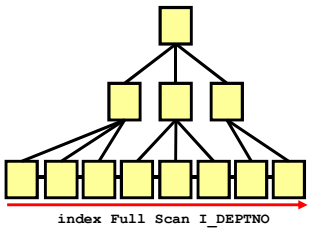



```
create index IX_FBI on EMP(UPPER(ename));

select * from emp where upper(ENAME) like 'A%';
```

Index Full Scan

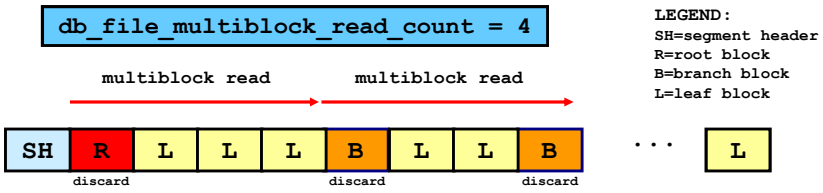
The optimizer considers:

- If a predicate references one of the columns in the index
- When there is no predicate, if both conditions are met: All columns in the table and referenced in the query are included in the index and at least one of the index columns is not null.

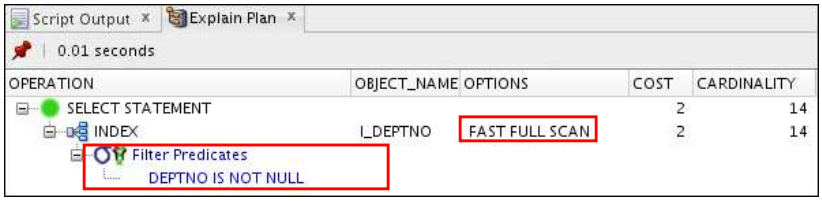



```
create index I_DEPTNO on EMP(deptno);

select *
from emp
where sal > 1000 and deptno is not null
order by deptno;
```

Index Fast Full Scan

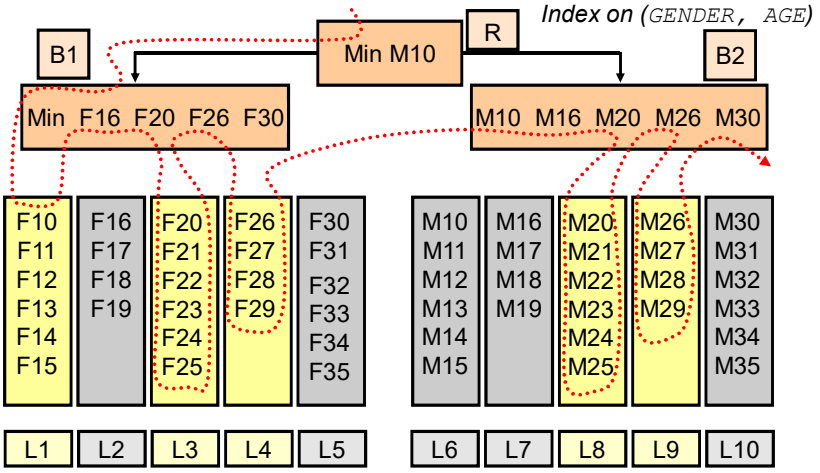



```
select /*+ INDEX_FFS(EMP I_DEPTNO)  */ deptno from emp
where deptno is not null;
```



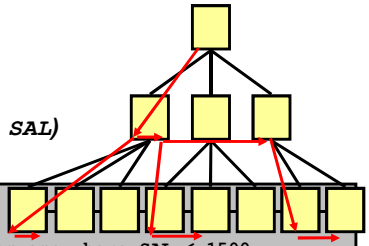
Index Skip Scan

```
SELECT * FROM employees WHERE age BETWEEN 20 AND 29
```



Index Skip Scan: Example

Index on (DEPTNO, SAL)



```
create index IX_SS on EMP(DEPTNO,SAL);

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

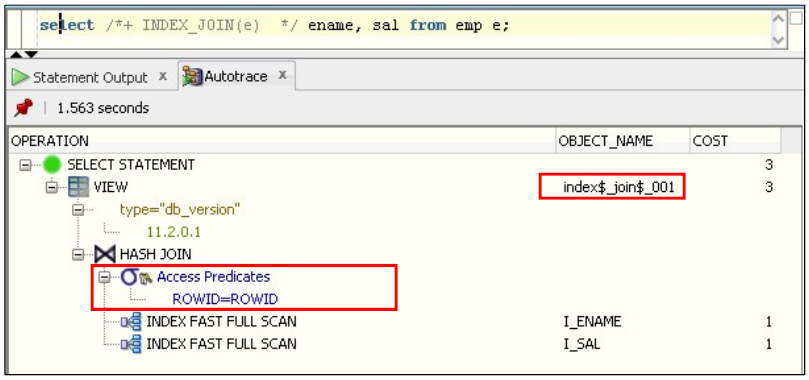
Script Output | Explain Plan

0.01 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			4	2
TABLE ACCESS	EMP	BY INDEX ROWID	4	2
INDEX	IX_SS	SKIP SCAN	3	2
Access Predicates				
SAL<1500				
Filter Predicates				
SAL<1500				

Index Join Scan

```
alter table emp modify (SAL not null, ENAME not null);
create index I_ENAME on EMP(ename);
create index I_SAL on EMP(sal);
```



B*-tree Indexes and Nulls

```
create table nulltest ( col1 number, col2 number not null);
create index nullind1 on nulltest (col1);
create index notnullind2 on nulltest (col2);
```

Using Indexes: Considering Nullable Columns

Column	Null?
SSN	Y
FNAME	Y
LNAME	N

PERSON

```
CREATE UNIQUE INDEX person_ssn_ix
    ON person(ssn);
```

```
SELECT COUNT(*) FROM person;

  SELECT STATEMENT    |
   SORT AGGREGATE     |
    TABLE ACCESS FULL| PERSON
```

```
DROP INDEX person_ssn_ix;
```

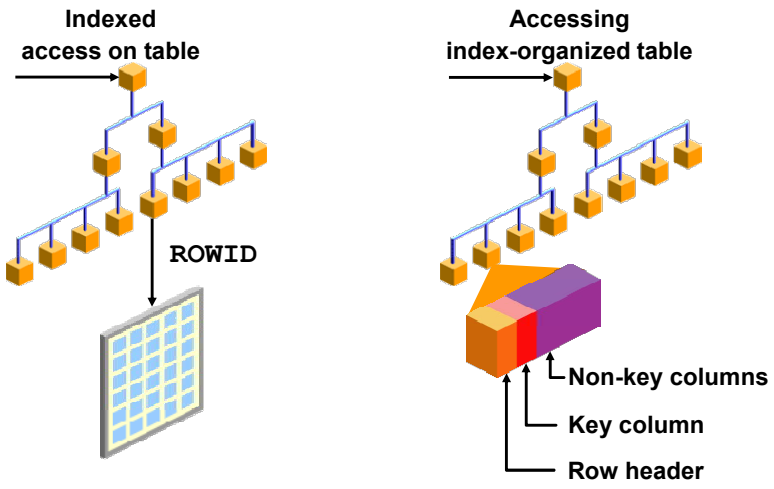
Column	Null?
SSN	N
FNAME	Y
LNAME	N

PERSON

```
ALTER TABLE person ADD CONSTRAINT pk_ssn
    PRIMARY KEY (ssn);
```

```
SELECT /*+ INDEX(person) */ COUNT(*) FROM
    person;

  SELECT STATEMENT       |
   SORT AGGREGATE        |
    INDEX FAST FULL SCAN| PK_SSN
```

Index-Organized Tables



Index-Organized Table Scans

```
create table iotemp
 ( empno number(4) primary key, ename varchar2(10) not null,
   job varchar2(9), mgr number(4), hiredate date,
   sal number(7,2) not null,  comm number(7,2), deptno number(2))
organization index;
```

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

Script Output | Explain Plan

0.734 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	SYS_IOT_TOP_7..	UNIQUE SCAN	1	1
Access Predicates				
EMPNO=9999				

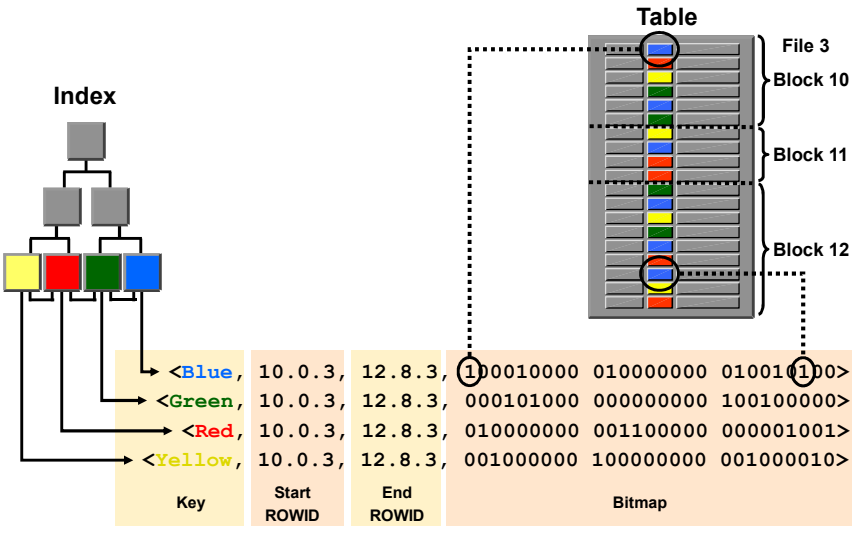
```
select * from iotemp where sal>1000;
```

Script Output | Explain Plan

0.007 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
INDEX	SYS_IOT_TOP_7...	FAST FULL SCAN	2	1
Filter Predicates				
SAL>1000				

Bitmap Indexes



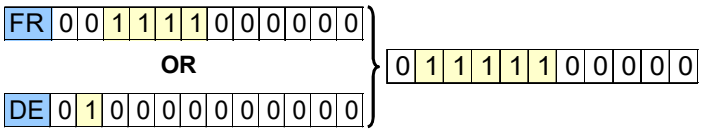
Bitmap Index Access: Examples

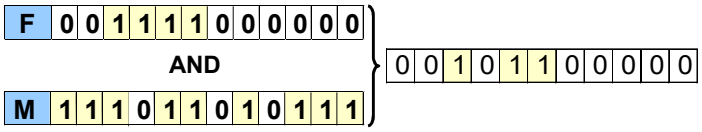
```
SELECT * FROM PERF_TEAM WHERE country='FR';
-------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | PERF_TEAM |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS |           |       |       |
|   3 |    BITMAP INDEX SINGLE VALUE  | IX_B2     |       |       |
-------------------------------------------------------------------------
Predicate:  3 - access("COUNTRY"='FR')
```

```
SELECT * FROM PERF_TEAM WHERE country>'FR';
-------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | PERF_TEAM |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS |           |       |       |
|   3 |    BITMAP INDEX RANGE SCAN    | IX_B2     |       |       |
-------------------------------------------------------------------------
Predicate: 3 - access("COUNTRY">'FR') filter("COUNTRY">'FR')
```

Combining Bitmap Indexes: Examples

```
SELECT * FROM PERF_TEAM WHERE country in('FR','DE');
```






```
SELECT * FROM EMEA_PERF_TEAM T WHERE country='FR' and gender='M';
```

Combining Bitmap Index Access Paths

```
SELECT * FROM PERF_TEAM WHERE country in ('FR','DE');
-------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes |
|   0 | SELECT STATEMENT              |           |     1 |    45 |
|   1 |  INLIST ITERATOR              |           |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID | PERF_TEAM |     1 |    45 |
|   3 |    BITMAP CONVERSION TO ROWIDS|           |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE | IX_B2     |       |       |
Predicate: 4 - access("COUNTRY"='DE' OR "COUNTRY"='FR')
```

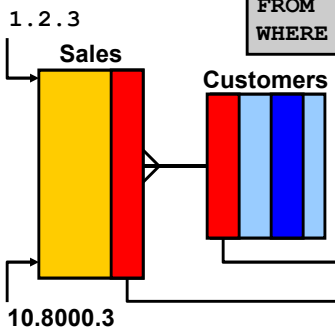
```
SELECT * FROM PERF_TEAM WHERE country='FR' and gender='M';
-------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes |
|   0 | SELECT STATEMENT              |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID | PERF_TEAM |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS|           |       |       |
|   3 |    BITMAP AND                |           |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE| IX_B1     |       |       |
|   5 |     BITMAP INDEX SINGLE VALUE| IX_B2     |       |       |
Predicate: 4 - access("GENDER"='M') 5 - access("COUNTRY"='FR')
```

Bitmap Operations

- BITMAP CONVERSION:
  - TO ROWIDs
  - FROM ROWIDs
  - COUNT
- BITMAP INDEX:
  - SINGLE VALUE
  - RANGE SCAN
  - FULL SCAN
- BITMAP MERGE
- BITMAP AND/OR
- BITMAP MINUS
- BITMAP KEY ITERATION

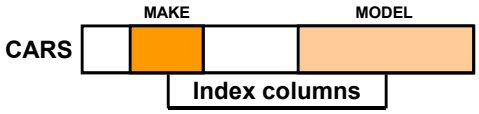
Bitmap Join Index

```
CREATE BITMAP INDEX cust_sales_bji
ON    sales(c.cust_city)
FROM  sales s, customers c
WHERE c.cust_id = s.cust_id;
```




```
        <Rognes, 1.2.3, 10.8000.3, 1000100100101001…>
<Aix-en-Provence, 1.2.3, 10.8000.3, 0001010001000001…>
      <Marseille, 1.2.3, 10.8000.3, 0100000010000011…>
```

Composite Indexes

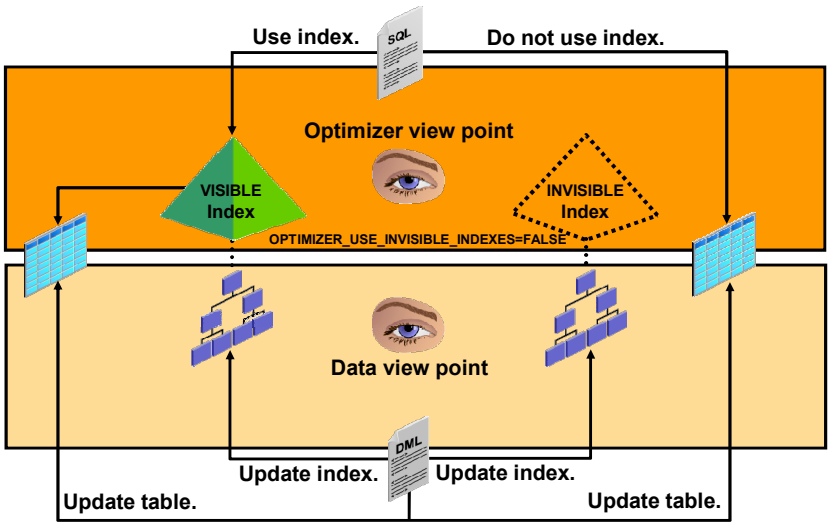



```
create index cars_make_model_idx on cars(make, model);
```

```
select *
from cars
where make = 'CITROËN' and model = '2CV';
```

```
-------------------------------------------------------------------
| Id  | Operation                    | Name                       |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                            |
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS                  |
|*  2 |   INDEX RANGE SCAN           | CARS_MAKE_MODEL_IDX        |
-------------------------------------------------------------------
```

Invisible Index: Overview



Invisible Indexes: Examples

- Index is altered as not visible to the optimizer:

```
ALTER INDEX ind1 INVISIBLE;
```

- Optimizer does not consider this index:

```
SELECT /*+ index(TAB1 IND1) */ COL1 FROM TAB1 WHERE …;
```

- Optimizer considers this index:

```
ALTER INDEX ind1 VISIBLE;
```

- You initially create an index as invisible:

```
CREATE INDEX IND1 ON TAB1(COL1) INVISIBLE;
```

Guidelines for Managing Indexes

- Create indexes after inserting table data.
- Index the correct tables and columns.
- Order index columns for performance.
- Limit the number of indexes for each table.
- Drop indexes that are no longer required.
- Specify the tablespace for each index.
- Consider parallelizing index creation.
- Consider creating indexes with `NOLOGGING`.
- Consider costs and benefits of coalescing or rebuilding indexes.
- Consider cost before disabling or dropping constraints.

Quiz

Assuming that the column `EMAIL` has an index, the following query results in an index range scan:

```
Select employee_name from employees where
email like '%A';
```

a. True
b. False

Quiz

To get optimum result from indexes:

a. Create indexes before inserting table data.
b. Do not order index columns.
c. Limit the number of indexes for each table.
d. Do not specify the tablespace for each index.

Common Observations

Review the following common observations:

- Why is full table scan used?
- Why is full table scan not used?
- Why is index scan not used?

Why Is Full Table Scan Used?

Review the following common causes:

- You set parameters that affect the optimizer's cost estimation.
- A large volume of business data must be processed.

Why Is Full Table Scan Not Used?

Review the following common causes:

- `INDEX FULL SCAN` is used to avoid a sort operation.
- You set parameters that affect the optimizer's cost estimation.
- Optimizer mode or hint is set to `FIRST_ROWS` or `FIRST_ROWS_n`.
- Query has a `USE_NL` hint that is not appropriate.
- Query has a `USE_NL`, `FIRST_ROWS`, or `FIRST_ROWS_n` hint that is favoring nested loop join.
- No parallel slaves are available for the query.

Why Is Index Scan Not Used?

Review the following common causes:

- There are functions being applied to the predicate.
- There is a data type mismatch.
- Statistics are old.
- The column can contain null.
- Using the index would actually be slower than not using it.
- You set parameters that affect the optimizer's cost estimation.
- The optimizer costs a full table scan cheaper than a series of index range scans.
- No index is available for columns in the predicate.

Why Is Index Scan Not Used?

Review the following common causes:

- Available indexes are too unselective.
- The index's cluster factor is too high.
- The query has a hint that is preventing the use of indexes.
- The index hint is being ignored.
- The incorrect `OPTIMIZER_MODE` is being used.
- A filter predicate is missing.

Summary

In this lesson, you should have learned to:

- Describe the SQL operators for tables and indexes
- List the possible access paths
- Describe common observations

Practice 7: Overview

This practice covers using different access paths for better optimization (case 1 through case 13).