

Politecnico di Torino
Database Management Systems

Oracle Optimizer



Tania Cerquitelli
tania.cerquitelli@polito.it

A.A. 2014-2015




Optimizer objective

- A SQL statement can be executed in many different ways
- The query **optimizer** determines the most efficient way to execute a SQL statement after considering many factors (e.g., objects referenced, conditions specified in the query)
- The output from the optimizer is a **plan** that describes an optimum method of execution (i.e., minimum execution **cost**)
- The **cost** is an estimated value proportional to the expected **resource** use (i.e., I/O, CPU, and memory) needed to execute the statement with a particular plan



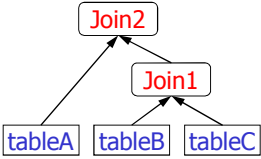
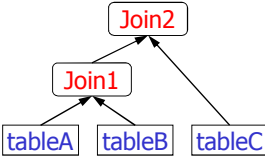
Database Management Systems


2




Plan Generator

- Its main function is to
 - try out different possible **plans** for a given query
 - and pick the one that has **the lowest cost**
- Many plans are possible because of combinations of different
 - access paths
 - join methods
 - join orders
- It uses an internal **cutoff** to reduce the number of plans explored
 - the cutoff is based on the cost of the **current best plan**
 - if cutoff is high, more plans are explored, and vice versa







Database Management Systems
6



Optimizer operations

<i>Operation</i>	<i>Description</i>
Evaluation of expressions and conditions	The optimizer first evaluates expressions and conditions containing constants as fully as possible
Statement transformation	For complex statements involving, for example, correlated sub-queries or views, the optimizer might transform the original statement into an equivalent join statement
Choice of optimizer goals	The optimizer determines the goal of optimization
Choice of access paths	For each table accessed by the statement, the optimizer chooses one or more of available access paths to obtain data
Choice of join orders	For a join statement that joins more than two tables, the optimizer chooses which pair of tables is joined first, and then which table is joined to the result, and so on
Choice of join methods	For a join statement that joins more than two tables, the optimizer chooses which join method is exploited to perform the required operation

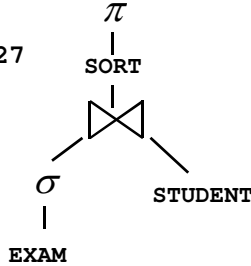

Database Management Systems
7




Example

STUDENT (SId, SSurname, SName)
COURSE (CCode, PId, Year, Semester)
EXAM (CCode, SId, Date, Score)


Query: SELECT SName, S.Sid
 FROM EXAM E, STUDENT S
 WHERE S.Sid=E.Sid and Score>=27
 ORDER BY SName



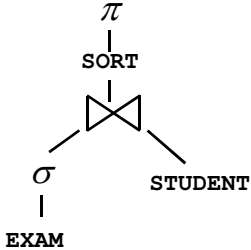


Database Management Systems

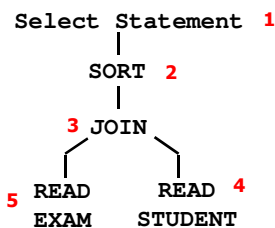
8




Example



Select Statement 1




Id	Pid	Operation	Cost
1	/	Select Statement	100
2	1	Sort	90
3	2	Join	70
4	3	Read STUDENT	40
5	3	Read EXAM + Selection	20




Database Management Systems

9




Access Paths for the Query Optimizer

- Access paths allow the **retrieval of data** from the database
 - **Index** access paths should be used for statements that retrieve a small subset of table rows
 - **Full scans** are more efficient when accessing a large portion of the table
- Data can be retrieved in any table by means of the following access paths
 - Full Table Scans
 - Index Scans
 - Rowid Scans




Database Management Systems 13




Full Table Scans

- This type of scan reads **all rows** from a table and filters out those that do not meet the selection criteria
- Each row is examined to determine whether it satisfies the statement's **WHERE** clause
- Physical blocks are adjacent and they are read **sequentially**
- Larger I/O calls are allowed, i.e., many blocks (multiblock) are read in a single I/O call
- **Multiblock** reads can be used to speed up the process
- The size of multiblock is initialized by the parameter **DB_FILE_MULTIBLOCK_READ_COUNT**





Database Management Systems 14



Assessing I/O for Blocks

- Oracle does I/O by **blocks**
 - Generally **multiple rows** are stored in each block. The total number of rows could be clustered together in a few blocks, or they could be spread out over a larger number of blocks.
- The optimizer decision to use full table scans is influenced by the **percentage of blocks** accessed, not rows. This is called the **index clustering factor**
- Although the clustering factor is a property of the index, the clustering factor actually relates to the spread of **similar indexed column values** within data blocks in the table
 - **Low** clustering factor: individual rows are **concentrated** within fewer blocks in the table.
 - **High** clustering factor: individual rows are **scattered** more randomly across blocks in the table. It costs more to use a range scan to fetch rows by rowid, because more blocks in the table need to be visited to return the data.


Database Management Systems16




Effects of Clustering Factor on Cost

- Assume the following situation
 - There is a table with **9 rows**
 - There is a non-unique **index** on column 1
 - Column 1 currently stores the **values A, B, and C**
 - Oracle stores the table using only **3 blocks**
- **Case 1.** The **index clustering factor** is **low** for the rows as they are arranged in the following diagram

Block 1	Block 2	Block 3
A A A	B B B	C C C


Database Management Systems17




Effects of Clustering Factor on Cost

- **Case 2.** If the same rows in the table are rearranged so that the index values are scattered across the table blocks (rather than clustered together), then the **index clustering factor** is **higher**, as in the following schema.


Block 1	Block 2	Block 3
-----	-----	-----
A B C	A B C	A B C


Database Management Systems18



When the Optimizer Uses Full Table Scans

- Lack of **index**
- Retrieval of a large amount of data stored in the target table
 - If the query will access **most of the blocks** in the table, the optimizer uses a full table scan, even though indexes might be available
 - Full table scans can use larger I/O calls, and making **fewer large I/O calls** is cheaper than making many smaller calls
- Small table
 - If a table has less than `DB_FILE_MULTIBLOCK_READ_COUNT` blocks it can be read in a **single I/O call**, then a full table scan might be cheaper than an index range scan

Database Management Systems19





Oracle indexes

- System indexes (secondary indexes) created automatically on the **primary key** attributes
 - SYS_#
- **Primary** indexes
 - Clustered Btree (physical sort)
 - Hash (bucket)
- **Secondary** indexes
 - Btree
 - Bitmap
 - Hash

```
CREATE INDEX IndexName ON Table (Column, ...);


DROP INDEX IndexName;
```


 Database Management Systems 20



Index Scans


- The index contains the indexed **value** and the **rowids** of rows in the table having that value
- An index scan retrieves data from an index based on the value of one or more columns in the index
 - Oracle searches the index for the indexed column **values accessed by the statement**
 - If the statement accesses only columns of the index, the indexed column values are read **directly** from the index, otherwise the rows in the table are accessed by means of the **rowid**
- An index scan can be one of the following types
 - Index Unique Scans
 - Index Range Scans
 - Index Full Scans
 - Fast Full Index Scans
 - Bitmap Indexes


 Database Management Systems 21



Index Unique Scans


- This scan returns at most a **single rowid** for each indexed value
- Oracle performs a unique scan if a statement contains a **UNIQUE** or a **PRIMARY KEY** constraint that guarantees that only a single row is accessed
- It is used when all columns of a unique (e.g., B-tree) index or an index created as a result of a primary key constraint are specified with **equality** conditions

Database Management Systems22



Index Range Scans

- An index range scan is a common operation for accessing **selective** data
- Data is returned in the **ascending order** of index columns. Multiple rows with identical values are sorted in ascending order by rowid
- The optimizer uses a range scan when it finds one or more **leading columns** of an index specified in conditions
 - col1 = :b1
 - col1 <= :b1
 - col1 >=:b1
 - and combinations of the preceding conditions for leading columns in the index
- Range scans can use unique or non-unique indexes
- Range scans **avoid sorting** when index columns constitute the **ORDER BY/GROUP BY** clause

Database Management Systems23



Index Full Scans

- A index full scan is available if a predicate references one of the **columns** in the index. The predicate does not need to be an index driver.
- It is also available when there is **no predicate**, if both the following conditions are met
 - **all** of the **columns** in the table referenced in the query are included in the index
 - at least one of the index columns is **not null**
- A full scan can be used to eliminate a **sort** operation (required by **GROUP BY**, **ORDER BY**, **MERGE JOIN**), because the data is **ordered** by the index key
- It reads the blocks singly (one by one)



Fast Full Index Scans

- Fast full index scans are an alternative to a full table scan when the index contains **all** the **columns** that are **needed** for the query, and at least one column in the index key has the **NOT NULL** constraint
- A fast full scan accesses the data in the index itself, **without accessing the table**
- It cannot be used to eliminate a **sort** operation, because the data is **not ordered** by the index key
- A fast full scan is **faster** than a normal full index scan
 - It reads the entire index using **multiblock** reads





Bitmap Indexes

- Bitmap indexes are most effective for queries that contain multiple conditions in the WHERE clause
- They are usually easier to destroy and re-create than to maintain
- A **bitmap join** uses a bitmap for key values and a mapping function that converts each bit position to a rowid



Database Management Systems

26




Rowid Scans

- The **rowid** of a row specifies the data **file** and data **block** (i.e., physical address) containing the row and the location of the row in that block
- Locating a row by its rowid is the **fastest** way to retrieve a **single row**
- To access a table by rowid (in Oracle)
 - Rowids of the selected rows are obtained through an **index scan** of one or more of the table's indexes
 - Each selected row is accessed in the table based on the **physical address** obtained by its rowid



Database Management Systems

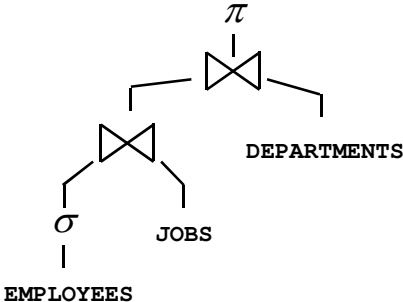
29





Rowid Scans: Example

```

EXPLAIN PLAN FOR
SELECT e.employee_id, j.job_title, e.salary, d.department_name
FROM employees e, jobs j, departments d
WHERE e.employee_id < 103
      AND e.job_id = j.job_id
      AND e.department_id = d.department_id;
    
```




Database Management Systems
35



Rowid Scans: Example

```


EXPLAIN PLAN FOR
SELECT e.employee_id, j.job_title, e.salary, d.department_name
FROM employees e, jobs j, departments d
WHERE e.employee_id < 103
      AND e.job_id = j.job_id
      AND e.department_id = d.department_id;
    
```


Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		3	189	10 (10)
1	NESTED LOOPS		3	189	10 (10)
2	NESTED LOOPS		3	141	7 (15)
* 3	TABLE ACCESS FULL	EMPLOYEES	3	60	4 (25)
4	TABLE ACCESS BY INDEX ROWID	JOBS	19	513	2 (50)
* 5	INDEX UNIQUE SCAN	JOB_ID_PK	1		
6	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	27	432	2 (50)
* 7	INDEX UNIQUE SCAN	DEPT_ID_PK	1		

Predicate Information (identified by operation id):

```


3 - filter("E"."EMPLOYEE_ID"<103)
5 - access("E"."JOB_ID"="J"."JOB_ID")
7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
    
```



Database Management Systems
36



JOIN


- Join Method
 - To join each **pair of row** sources, Oracle must perform a join operation
 - Join methods include
 - nested loop
 - sort merge
 - hash joins
- Join Order
 - To execute a statement that joins **more than two tables**, Oracle joins two of the tables and then joins the resulting row source to the next table
 - This process is continued until all tables are joined into the result.


Database Management Systems37



Understanding Statistics


- Optimizer statistics are a collection of data that **describe** more details about the database and the objects in the database
- Optimizer statistics, stored in the **data dictionary**, include the following:
 - Table statistics
 - Number of rows
 - Number of blocks
 - Average row length
 - Column statistics
 - Number of **distinct values** (NDV) in columns
 - Number of **nulls** in columns
 - Data distribution (**histogram**)
 - Index statistics
 - Number of leaf blocks
 - Levels
 - Clustering factor
 - System statistics
 - I/O performance and utilization
 - CPU performance and utilization

Database Management Systems46




Statistics on Tables, Indexes and Columns

- To view statistics in the data dictionary, query the appropriate data dictionary view (USER, ALL, or DBA). These DBA_* views include the following:
 - **DBA_TABLES**
 - **DBA_OBJECT_TABLES**
 - **DBA_TAB_STATISTICS**
 - **DBA_TAB_COL_STATISTICS**
 - **DBA_TAB_HISTOGRAMS**
 - **DBA_INDEXES**
 - **DBA_IND_STATISTICS**
 - **DBA_CLUSTERS**
 - **DBA_TAB_PARTITIONS**
 - **DBA_TAB_SUBPARTITIONS**
 - **DBA_IND_PARTITIONS**
 - **DBA_IND_SUBPARTITIONS**
 - **DBA_PART_COL_STATISTICS**
 - **DBA_PART_HISTOGRAMS**
 - **DBA_SUBPART_COL_STATISTICS**
 - **DBA_SUBPART_HISTOGRAMS**
- `describe table_name` allows to view the table schema




Database Management Systems 47




Column Statistics and Histograms

- When gathering statistics on a table, **DBMS_STATS** gathers information about the **data distribution** of the columns within the table (e.g., the maximum value and minimum value of the column)
- For **skewed** data distributions, **histograms** can also be created as part of the column statistics to describe the data distribution of a given column





Database Management Systems 51



Histograms

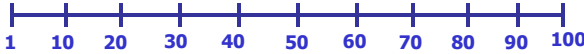
- Column statistics may be stored as histograms which provide **accurate** estimates of the distribution of column data.
- Histograms provide improved **selectivity** estimates in the presence of data skew, resulting in optimal execution plans with **non-uniform** data distributions
- Oracle uses two types of histograms for column statistics
 - Height-balanced histograms
 - Frequency histograms
- The type of histogram is stored in the **HISTOGRAM** column of the **USER/DBA_TAB_COL_STATISTICS** views


Database Management Systems
52





Height-Balanced Histograms


- In a height-balanced histogram, the column values are divided into **bands** so that each band contains approximately the **same number of rows**.
- The useful information that the histogram provides is where in the range of values the **endpoints** fall.
- Consider a column C with values between 1 and 100 and a histogram with 10 buckets



- If the data is not uniformly distributed, then the histogram might look similar to




Database Management Systems
53



Height-Balanced Histograms

```

SELECT column_name, num_distinct, num_buckets, histogram
  FROM USER_TAB_COL_STATISTICS
 WHERE table_name = 'INVENTORIES' AND column_name = 'QUANTITY_ON_HAND';
    
```


COLUMN_NAME	NUM_DISTINCT	NUM_BUCKETS	HISTOGRAM
QUANTITY_ON_HAND	237	10	HEIGHT BALANCED

```

SELECT endpoint_number, endpoint_value
  FROM USER_HISTOGRAMS
 WHERE table_name = 'INVENTORIES' and column_name = 'QUANTITY_ON_HAND'
 ORDER BY endpoint_number;
    
```


ENDPOINT_NUMBER	ENDPOINT_VALUE
0	0
1	27
2	42
3	57
4	74
5	98
6	123
7	149
8	175
9	202
10	353

Height-Balanced Histograms




Database Management Systems

54




Frequency Histograms

- In a frequency histogram, each **value** of the column corresponds to a single **bucket** of the histogram
- Each bucket contains the number of **occurrences** of that single value.
- Frequency histograms are automatically created instead of height-balanced histograms when the number of **distinct values** is less than or equal to the number of histogram **buckets** specified
- Frequency histograms can be viewed using the ***USER_HISTOGRAMS** tables



Database Management Systems

55



Frequency Histograms


```


SELECT column_name, num_distinct, num_buckets, histogram
  FROM USER_TAB_COL_STATISTICS
 WHERE table_name = 'INVENTORIES' AND column_name = 'WAREHOUSE_ID';

COLUMN_NAME                NUM_DISTINCT NUM_BUCKETS HISTOGRAM
-----
WAREHOUSE_ID                9              9 FREQUENCY

SELECT endpoint_number, endpoint_value
  FROM USER_HISTOGRAMS
 WHERE table_name = 'INVENTORIES' and column_name = 'WAREHOUSE_ID'
 ORDER BY endpoint_number;


ENDPOINT_NUMBER ENDPOINT_VALUE
-----
          36             1
         213             2
         261             3
         370             4
         484             5
         692             6
         798             7
         984             8
        1112             9
    
```



Database Management Systems
56



Choosing an Optimizer Goal

- Optimization for best **throughput**
 - Optimizer chooses the least amount of resources necessary to process **all rows** accessed by the statement
 - Throughput is more important in **batch** applications (e.g., Oracle Reports applications) because the user is only concerned with the time necessary for the application to complete
- Optimization for best **response time**
 - Optimizer uses the least amount of resources necessary to process the **first row** accessed by a SQL statement.
 - Response time is important in **interactive** applications (e.g., SQL*Plus queries)


Database Management Systems
57




OPTIMIZER_MODE Parameter Values

Value	Description
ALL_ROWS	The optimizer uses a cost-based approach for all SQL statements in the session. It optimizes with a goal of best throughput (minimum resource use to complete the entire statement). Default.
FIRST_ROWS_n	The optimizer uses a cost-based approach, optimizes with a goal of best response time to return the first n number of rows ; n can equal 1, 10, 100, or 1000
FIRST_ROWS	The optimizer uses a mix of cost and heuristics to find a best plan for fast delivery of the first few rows

- The following SQL statement changes the goal of the query optimizer for the current session to **best response time**

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_1;
```



Database Management Systems

58