

Other Optimizer Operators

Chapter 8

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

8

Other Optimizer Operators

ORACLE

Objectives

Objectives

After completing this lesson, you should be able to:

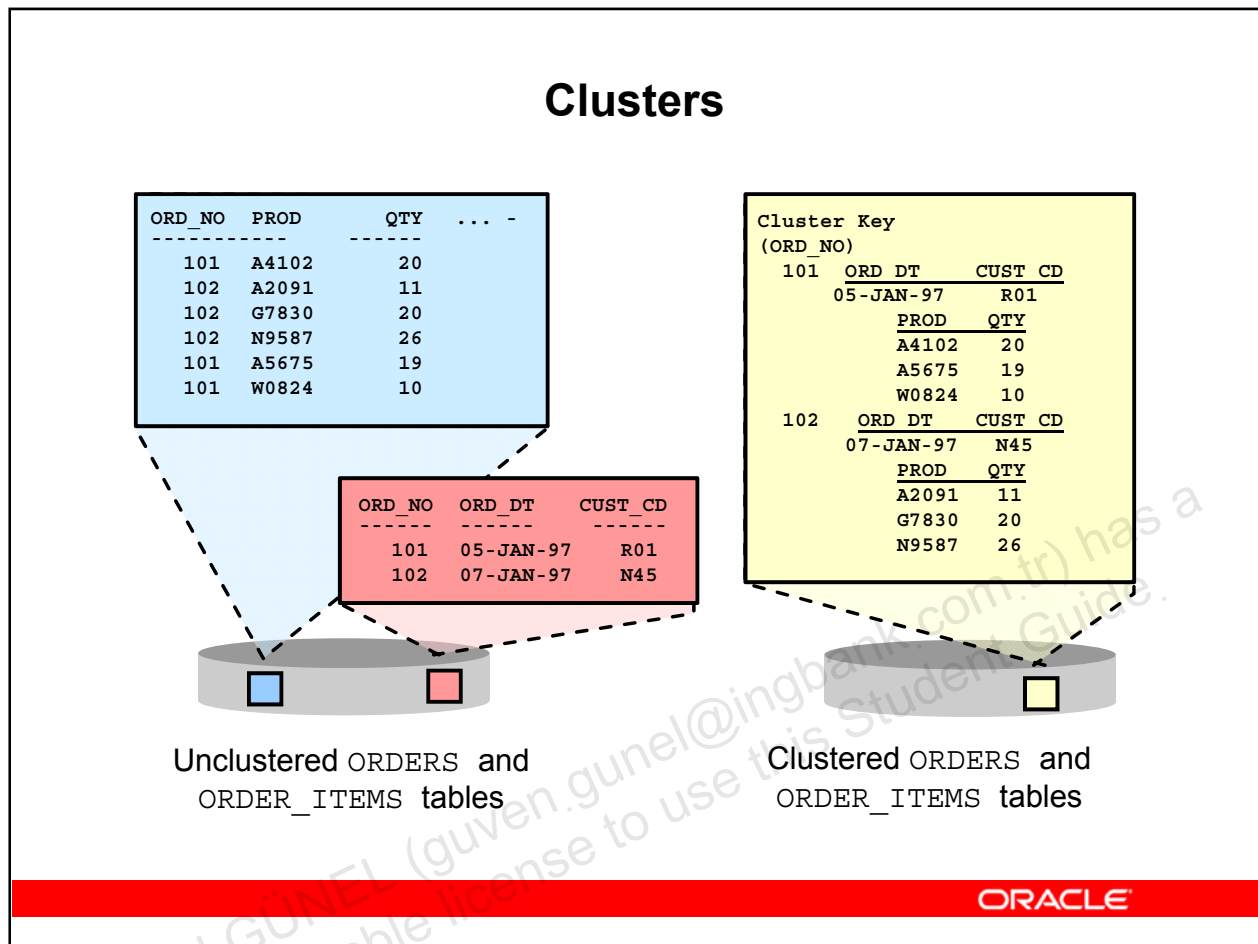
- Describe SQL operators for:
 - Clusters
 - In-List
 - Sorts
 - Filters
 - Set Operations
- Result Cache operators

ORACLE

Objectives

This lesson helps you understand the execution plans that use common operators of other access methods.

Clusters



Clusters

Clusters are an optional method for storing table data. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. For example, the `ORDERS` and `ORDER_ITEMS` table share the `ORDER_ID` column. When you cluster the `ORDERS` and `ORDER_ITEMS` tables, the system physically stores all rows for each order from both the `ORDERS` and `ORDER_ITEMS` tables in the same data blocks.

Cluster index: A cluster index is an index defined specifically for a cluster. Such an index contains an entry for each cluster key value. To locate a row in a cluster, the cluster index is used to find the cluster key value, which points to the data block associated with that cluster key value. Therefore, the system accesses a given row with a minimum of two I/Os.

Hash clusters: Hashing is an optional way of storing table data to improve the performance of data retrieval. To use hashing, you create a hash cluster and load tables into the cluster. The system physically stores the rows of a table in a hash cluster and retrieves them according to the results of a hash function. The key of a hash cluster (just as the key of an index cluster) can be a single column or composite key. To find or store a row in a hash cluster, the system applies the hash function to the row's cluster key value; the resulting hash value corresponds to a data block in the cluster, which the system then reads or writes on behalf of the issued statement.

Note: Hash clusters are a better choice than using an indexed table or index cluster when a table is queried frequently with equality queries.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

When Are Clusters Useful?

When Are Clusters Useful?

- Index cluster:
 - Tables always joined on the same keys
 - The size of the table is not known
 - In any type of searches
- Hash cluster:
 - Tables always joined on the same keys
 - Storage for all cluster keys allocated initially
 - In either equality (=) or nonequality (<>) searches

ORACLE

When Are Clusters Useful?

- Index clusters allow row data from one or more tables that share a cluster key value to be stored in same block. You can locate these rows using a cluster index, which has one entry per cluster key value and *not* for each row. Therefore, the index is smaller and less costly to access for finding multiple rows. The rows with the same key are in a small group of blocks. This means that in an index cluster the clustering factor is very good and provides clustering for data from multiple tables sharing the same join key. The smaller index and smaller group of blocks reduce the cost of access by reducing block visits to the buffer cache. Index clusters are useful when the size of the tables is not known in advance (For example: Creating a new table rather than converting an existing one whose size is stable) because a cluster bucket is only created after a cluster key value is used. They are also useful for all filter operations or searches. Note that full table scans do not perform well on a table in a multiple table cluster as it has more blocks than the table would have if created as a heap table.
- Hash clusters allow row data from one or more tables that share a cluster key value to be stored in same block. You can locate these rows using a system-provided or user-provided hashing function or using the cluster key value assuming that this is already evenly distributed making the access to a row faster than using index clusters. Table rows with the same cluster key values hash into the same cluster buckets and can be stored in the same block or small group of blocks.

When Are Clusters Useful?

When Are Clusters Useful?

- Single-table hash cluster:
 - Fastest way to access a large table with an equality search
- Sorted hash cluster:
 - Only used for equality search
 - Avoid sorts on batch reporting
 - Avoid overhead probe on the branch blocks of an IOT

ORACLE

When Are Clusters Useful? (continued)

- This means that in a hash cluster the clustering factor is also very good and a row may be accessed by its key with one block visit only and without needing an index. Hash clusters allocate all the storage for all the hash buckets when the cluster is created, so they may waste space. They also do not perform well other than on equality searches or nonequality searches. Like index clusters if they contain multiple tables, full scans are more expensive for the same reason.
- Single-table hash clusters are similar to a hash cluster, but are optimized in the block structures for access to a single table, thereby providing the fastest possible access to a row other than by using a rowid filter. As they only have one table, full scans, if they happen, cost as much as they would in a heap table.
- Sorted hash clusters are designed to reduce costs of accessing ordered data by using a hashing algorithm on the hash key. Accessing the first row matching the hash key may be less costly than using an IOT for a large table because it saves the cost of a B*-tree probe. All the rows that match on a particular hash key (For example: Account number) are stored in the cluster in the order of the sort key or keys (For example: Phone calls), thereby, eliminating the need for a sort to process the order by clause. These clusters are very good for batch reporting, billing, and so on.

Cluster Access Path: Examples

Cluster Access Path: Examples

The first screenshot shows the query: `select * from bigemp_fact where deptno=10;` The Autotrace results are as follows:

OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		1	
TABLE ACCESS HASH	BIGEMP_FACT	1	

The second screenshot shows the query: `select * from emp,dept where emp.deptno=dept.deptno and emp.deptno > 800;` The Autotrace results are as follows:

OPERATION	OBJECT_NAME	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT		11520	
NESTED LOOPS		11520	69
TABLE ACCESS CLUSTER	DEPT	174	60
INDEX RANGE SCAN	EMP_DEPT_IN...	2	2
TABLE ACCESS CLUSTER	EMP	57	9

ORACLE

Cluster Access Path: Examples

The example in the slide shows you two different cluster access paths.

In the top one, a hash scan is used to locate rows in a hash cluster, based on a hash value. In a hash cluster, all rows with the same hash value are stored in the same data block. To perform a hash scan, the system first obtains the hash value by applying a hash function to a cluster key value specified by the statement. The system then scans the data blocks containing rows with that hash value.

The second one assumes that a cluster index was used to cluster both the `EMP` and `DEPT` tables. In this case, a cluster scan is used to retrieve, from a table stored in an indexed cluster, all rows that have the same cluster key value. In an indexed cluster, all rows with the same cluster key value are stored in the same data block. To perform a cluster scan, the system first obtains the `ROWID` of one of the selected rows by scanning the cluster index. The system then locates the rows based on this `ROWID`.

Note: You see examples of how to create clusters in the labs for this lesson.

Sorting Operators

Sorting Operators

- **SORT operator:**
 - AGGREGATE: Single row from group function
 - UNIQUE: To eliminate duplicates
 - JOIN: Precedes a merge join
 - GROUP BY, ORDER BY: For these operators
- **HASH operator:**
 - GROUP BY: For this operator
 - UNIQUE: Equivalent to SORT UNIQUE
- If you want ordered results, *always* use ORDER BY.

ORACLE

Sorting Operators

Sort operations result when users specify an operation that requires a sort. Commonly encountered operations include the following:

- SORT AGGREGATE does not involve a sort. It retrieves a single row that is the result of applying a group function to a group of selected rows. Operations such as COUNT and MIN are shown as SORT AGGREGATE.
- SORT UNIQUE sorts output rows to remove duplicates. It occurs if a user specifies a DISTINCT clause or if an operation requires unique values for the next step.
- SORT JOIN happens during a sort-merge join, if the rows need to be sorted by the join key.
- SORT GROUP BY is used when aggregates are computed for different groups in the data. The sort is required to separate the rows into different groups.
- SORT ORDER BY is required when the statement specifies an ORDER BY that cannot be satisfied by one of the indexes.
- HASH GROUP BY hashes a set of rows into groups for a query with a GROUP BY clause.

- `HASH UNIQUE` hashes a set of rows to eliminate duplicates. It occurs if a user specifies a `DISTINCT` clause or if an operation requires unique values for the next step. This is similar to `SORT UNIQUE`.

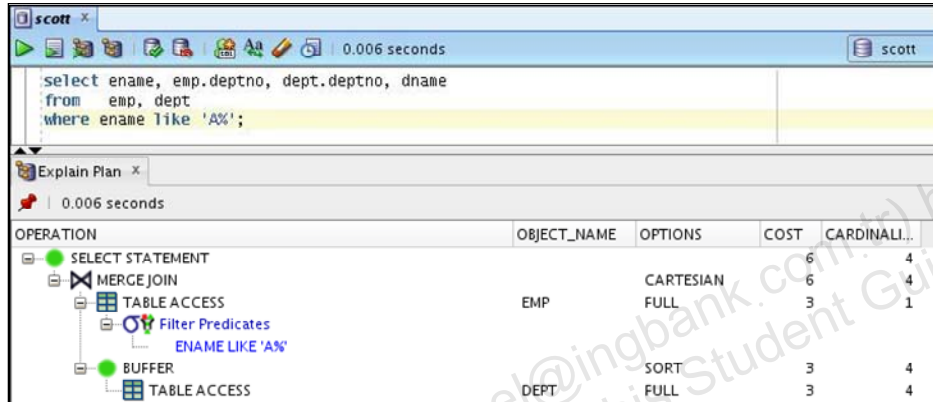
Note: Several SQL operators cause implicit sorts (or hashes since Oracle Database 10g, Release 2), such as `DISTINCT`, `GROUP BY`, `UNION`, `MINUS`, and `INTERSECT`. However, do not rely on these SQL operators to return ordered rows. If you want to have rows ordered, use the `ORDER BY` clause.

GÜVEN GÜNEL (guven.gunel@ingbank.com.tr) has a non-transferable license to use this Student Guide.

Buffer Sort Operator

Buffer Sort Operator

```
SELECT ename, emp.deptno, dept.deptno, dname
FROM emp, dept
WHERE ename like 'A%';
```



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			6	4
MERGE JOIN		CARTESIAN	6	4
TABLE ACCESS	EMP	FULL	3	1
Filter Predicates		ENAME LIKE 'A%'		
BUFFER		SORT	3	4
TABLE ACCESS	DEPT	FULL	3	4

ORACLE

Buffer Sort Operator

The **BUFFER SORT** operator uses a temporary table or a sort area in memory to store intermediate data. However, the data is not necessarily sorted.

The **BUFFER SORT** operator is needed if there is an operation that needs all the input data before it can start. (See “Cartesian Join.”)

So **BUFFER SORT** uses the buffering mechanism of a traditional sort, but it does not do the sort itself. The system simply buffers the data, in the User Global Area (UGA) or Program Global Area (PGA), to avoid multiple table scans against real data blocks.

The whole sort mechanism is reused, including the swap to disk when not enough sort area memory is available, but without sorting the data.

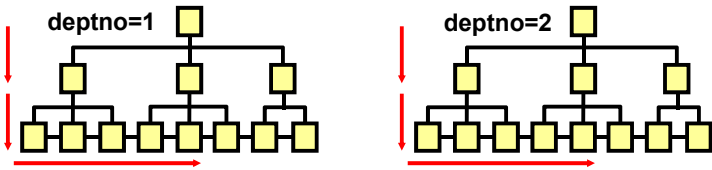
The difference between a temporary table and a buffer sort is as follows:

- A temporary table uses System Global Area (SGA).
- A buffer sort uses UGA.

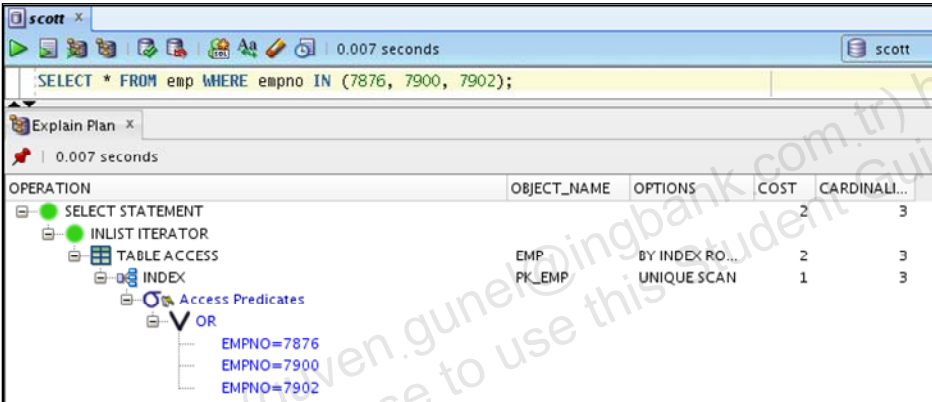
Inlist Iterator

Inlist Iterator

Every value executed separately



```
SELECT * FROM emp WHERE empno IN (7876, 7900, 7902);
```



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	3
INLIST ITERATOR				
TABLE ACCESS	EMP	BY INDEX ROWID	2	3
INDEX	PK_EMP	UNIQUE SCAN	1	3

Access Predicates:
OR
EMPNO=7876
EMPNO=7900
EMPNO=7902

ORACLE

Inlist Iterator

It is used when a query contains an `IN` clause with values or multiple equality predicates on the same column linked with `OR`s.

The `INLIST ITERATOR` operator iterates over the enumerated value list, and every value is executed separately.

The execution plan is identical to the result of a statement with an equality clause instead of `IN`, except for one additional step. The extra step occurs when `INLIST ITERATOR` feeds the equality clause with unique values from the list.

You can view this operator as a `FOR LOOP` statement in PL/SQL. In the example in the slide, you iterate the index probe over two values: 1 and 2.

Also, it is a function that uses an index, which is scanned for each value in the list. An alternative handling is `UNION ALL` of each value or a `FILTER` of the values against all the rows, this is significantly more efficient.

The optimizer uses an `INLIST ITERATOR` when an `IN` clause is specified with values, and the optimizer finds a selective index for that column. If there are multiple `OR` clauses using the same index, the optimizer selects this operation rather than `CONCATENATION` or `UNION ALL`, because it is more efficient.

View Operator

View Operator

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

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			3	14
VIEW	V		3	14
TABLE ACCESS	EMP	FULL	3	14

```
select v.*,d.dname from (select DEPTNO, sum(sal) SUM_SAL  
from emp group by deptno) v, dept d where v.deptno=d.deptno;
```

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			7	3
MERGE JOIN			7	3
TABLE ACCESS	DEPT	BY INDEX ...	2	4
INDEX	PK_DEPT	FULL SCAN	1	4
SORT		JOIN	5	3
Access Predicates				
V.DEPTNO=D.DEPT				
Filter Predicates				
V.DEPTNO=D.DEPT				
VIEW			4	3
HASH		GROUP BY	4	3
TABLE ACCESS	EMP	FULL	3	14

ORACLE

View Operator

Each query produces a variable set of data in the form of a table. A view simply gives a name to this set of data.

When views are referenced in a query, the system can handle them in two ways. If a number of conditions are met, they can be merged into the main query. This means that the view text is rewritten as a join with the other tables in the query. Views can also be left as standalone views and selected from directly as in the case of a table. Predicates can also be pushed into or pulled out of the views as long as certain conditions are met.

When a view is not merged, you can see the **VIEW** operator. The view operation is executed separately. All rows from the view are returned, and the next operation can be done.

Sometimes a view cannot be merged and must be executed independently in a separate query block. In this case, you can also see the **VIEW** operator in the explain plan. The **VIEW** keyword indicates that the view is executed as a separate query block. For example, views containing **GROUP BY** functions cannot be merged.

The second example in the slide shows a nonmergeable inline view. An inline view is basically a query within the **FROM** clause of your statement.

Basically, this operator collects all rows from a query block before they can be processed by higher operations in the plan.

Count Stop Key Operator

Count Stop Key Operator

```
SELECT count(*)  
FROM (SELECT /*+ NO_MERGE */ *  
      FROM emp WHERE empno = '1' and rownum < 10);
```

0.007 seconds

select count(*)
from (select /*+ NO_MERGE */ *
 from emp where empno = '1' and rownum < 10);

Autotrace x Script Output x Explain Plan x

0.007 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			0	1
SORT		AGGREGATE		1
VIEW			0	1
COUNT		STOPKEY		1
Filter Predicates				
ROWNUM<10				
INDEX	PK_EMP	UNIQUE SCAN	0	1
Access Predicates				
EMPNO=1				

ORACLE

Count Stop Key Operator

COUNT STOPKEY limits the number of rows returned. The limitation is expressed by the ROWNUM expression in the WHERE clause. It terminates the current operation when the count is reached.

Note: The cost of this operator depends on the number of occurrences of the values you try to retrieve. If the value appears very frequently in the table, the count is reached quickly. If the value is very infrequent, and there are no indexes, the system has to read most of the table's blocks before reaching the count.

Min/Max and First Row Operators

Min/Max and First Row Operators

```
SELECT MIN(quantity_on_hand)
FROM INVENTORIES
WHERE quantity_on_hand < 500;
```

The screenshot shows the Oracle SQL Developer interface. At the top, a status bar indicates '1.079 seconds'. Below it, the SQL query is displayed: `SELECT MIN(quantity_on_hand) FROM INVENTORIES WHERE quantity_on_hand < 500;`. The 'Script Output' tab is active, showing the execution plan. The plan consists of the following steps:

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
SORT		AGGREGATE	1	1
FIRST ROW			2	1
INDEX	INV_QTY_INDEX	RANGE SCAN...	2	1
Access Predicates		QUANTITY_ON_HAND<500		

ORACLE

Min/Max and First Row Operators

FIRST ROW retrieves only the first row selected by a query. It stops accessing the data after the first value is returned. This is an optimization introduced in Oracle 8i and it works with the index range scan and the index full scan.

In the example in the slide, it is assumed that there is an index on the `quantity_on_hand` column.

Other N-Array Operations

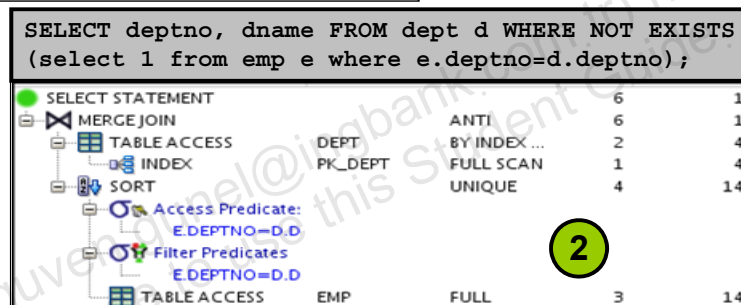
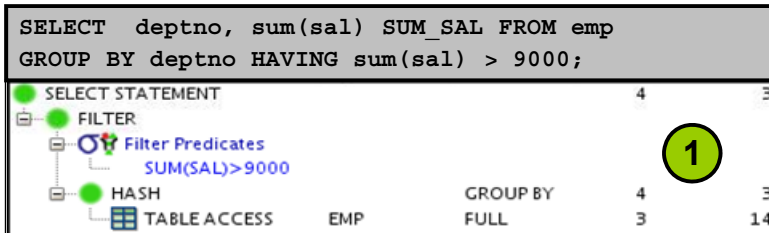
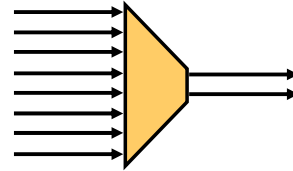
Other N-Array Operations

- FILTER
- CONCATENATION
- UNION ALL/UNION
- INTERSECT
- MINUS

ORACLE

FILTER Operations

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



ORACLE

FILTER Operations

A **FILTER** operation is any operation that discards rows returned by another step, but is not involved in retrieving the rows itself. All sorts of operations can be filters, including subqueries and single table predicates.

In the example 1, **FILTER** applies to the groups that are created by the **GROUP BY** operation.

In the example 2, **FILTER** is almost used in the same way as **NESTED LOOPS**. **DEPT** is accessed once, and for each row from **DEPT**, **EMP** is accessed by its index on **DEPTNO**. This operation is done as many times as the number of rows in **DEPT**.

The **FILTER** operation is applied, for each row, after **DEPT** rows are fetched. The **FILTER** discards rows for the inner query returned at least one row (`select 1 from emp e where e.deptno=d.deptno`) is **TRUE**.

Concatenation Operation

Concatenation Operation

```
SELECT * FROM emp WHERE deptno=1 or sal=2;
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		8	696
1	CONCATENATION			
2	TABLE ACCESS BY INDEX ROWID	EMP	4	348
3	INDEX RANGE SCAN	I_SAL	2	
4	TABLE ACCESS BY INDEX ROWID	EMP	4	348
5	INDEX RANGE SCAN	I_DEPTNO	2	

Predicate Information (identified by operation id):

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

ORACLE

Concatenation Operation

CONCATENATION concatenates the rows returned by two or more row sets. This works like UNION ALL and does not remove duplicate rows.

It is used with OR expansions. However, OR does not return duplicate rows, so for each component after the first, it appends a negation of the previous components (LNNVL):

CONCATENATION

- BRANCH 1 - SAL=2
- BRANCH 2 - DEPTNO = 1 AND NOT row in Branch 1

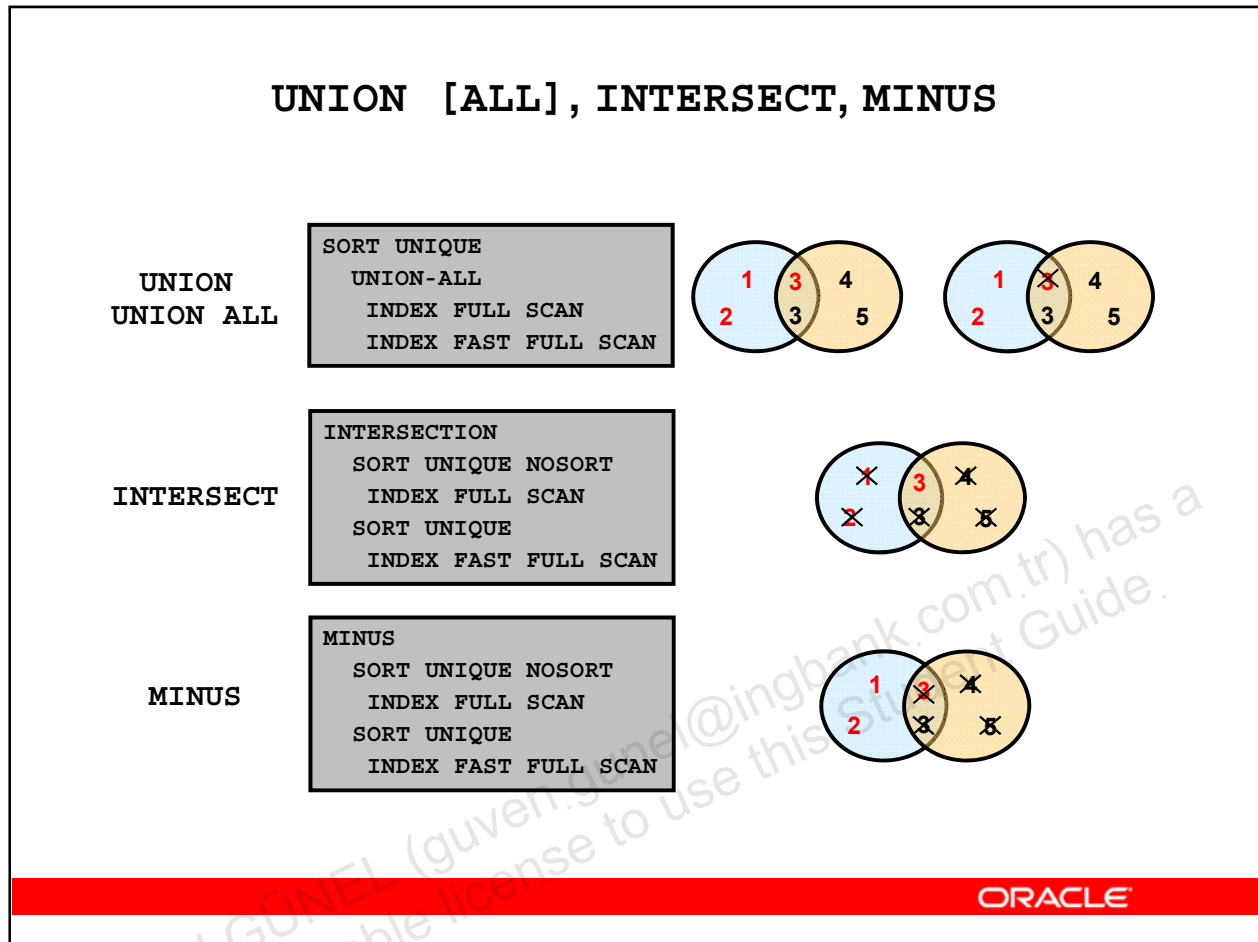
The LNNVL function is generated by the OR clause to process this negation.

The LNNVL() function returns TRUE, if the predicate is NULL or FALSE.

So filter (LNNVL(SAL=2)) returns all rows for which SAL != 2 or SAL is NULL.

Note: The explain plan in the slide is from Oracle Database 11g Release 1.

UNION [ALL], INTERSECT, MINUS



UNION [ALL], INTERSECT, MINUS

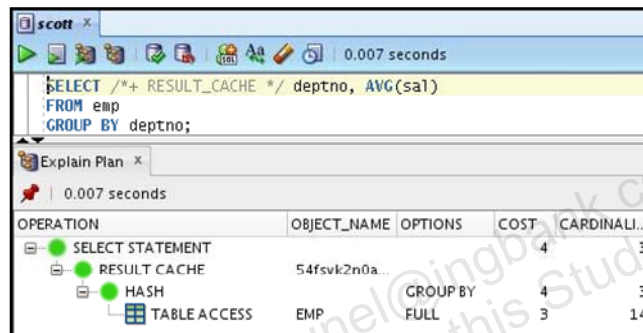
SQL handles duplicate rows with an **ALL** or **DISTINCT** modifier in different places in the language. **ALL** preserves duplicates and **DISTINCT** removes them. Here is a quick description of the possible SQL set operations:

- **INTERSECTION:** Operation accepting two sets of rows and returning the intersection of the sets, eliminating duplicates. Subrow sources are executed or optimized individually. This is very similar to sort-merge-join processing: Full rows are sorted and matched.
- **MINUS:** Operation accepting two sets of rows and returning rows appearing in the first set, but not in the second, eliminating duplicates. Subrow sources are executed or optimized individually. Similar to **INTERSECT** processing. However, instead of match-and-return, it is match-and-exclude.
- **UNION:** Operation accepting two sets of rows and returning the union of the sets, eliminating duplicates. Subrow sources are executed or optimized individually. Rows retrieved are concatenated and sorted to eliminate duplicate rows.
- **UNION ALL:** Operation accepting two sets of rows and returning the union of the sets, and not eliminating duplicates. The expensive sort operation is not necessary. Use **UNION ALL** if you know you do not have to deal with duplicates.

Result Cache Operator

Result Cache Operator

```
SELECT /*+ RESULT_CACHE */ deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			4	3
RESULT CACHE	54fsvk2n0a...			
HASH		GROUP BY	4	3
TABLE ACCESS	EMP	FULL	3	14

ORACLE

Result Cache Operator

The SQL query result cache enables explicit caching of query result sets and query fragments in database memory. A dedicated memory buffer stored in the shared pool can be used for storing and retrieving the cached results. The query results stored in this cache become invalid when data in the database objects that are accessed by the query is modified. Although the SQL query cache can be used for any query, good candidate statements are the ones that need to access a very high number of rows to return only a fraction of them. This is mostly the case for data warehousing applications.

If you want to use the query result cache and the `RESULT_CACHE_MODE` initialization parameter is set to `MANUAL`, you must explicitly specify the `RESULT_CACHE` hint in your query. This introduces the `ResultCache` operator into the execution plan for the query. When you execute the query, the `ResultCache` operator looks up the result cache memory to check whether the result for the query already exists in the cache. If it exists, the result is retrieved directly out of the cache. If it does not yet exist in the cache, the query is executed, the result is returned as output, and is also stored in the result cache memory.

If the `RESULT_CACHE_MODE` initialization parameter is set to `FORCE`, and you do not want to store the result of a query in the result cache, you must then use the `NO_RESULT_CACHE` hint in your query.

Quiz

Quiz

Hash clusters are a better choice than using an indexed table or index cluster when a table is queried frequently with equality queries.

- a. True
- b. False

ORACLE

Answer: a

Quiz

Quiz

The _____ operator uses a temporary table to store intermediate data.

- a. Buffer Sort Operator
- b. Inlist
- c. Min/Max
- d. N-Array

ORACLE

Answer: a

Quiz

Quiz

The following query uses the _____ operator:

```
SELECT * FROM emp WHERE empno IN (7876, 7900,  
7902) ;
```

- a. Buffer Sort Operator
- b. Inlist
- c. Min/Max
- d. N-Array

ORACLE

Answer: b

Quiz

Quiz

A `FILTER` operation retrieves rows returned by another statement.

- a. True
- b. False

ORACLE

Answer: b

Summary

Summary

In this lesson, you should have learned to:

- Describe SQL operators for:
 - Clusters
 - In-List
 - Sorts
 - Filters
 - Set Operations
- Result Cache operators

ORACLE

Practice 8: Overview

Practice 8: Overview

This practice covers the following topics:

- Using different access paths for better optimization
 - Case 14 to case 16
- Using the result cache

ORACLE