Oracle Hints
Version 11.1

Note: The demos on this page are intended to show valid syntax but it is far easier to use a hint to make a mess of things than it is to improve things. So many, if not most, of the demos will increase the cost.

Join methods:

In the **loop join** algorithm, an outer loop is formed that is composed of a few entries that are to be selected. Then, for each entry in the outer loop, a look-up is performed for matching entries, in the inner loop.

In the **merge join** algorithm, both tables are accessed in the same order. If there's a sorted index on the matching column, on both tables, then no sorting is needed. All we have to do is read the rows in the order presented by the index. The reason it's called a merge join is that the algorithm, in detail, looks much like the algorithm for merging two (sorted) data streams together.

Let's say we got two tables, ORDERS and ORDER_ITEMS. Let's say we have sorted indexes on ORDER_NUMBER on both tables. Naturally, the index on ORDERS can forbid duplicates, while the index on ORDER_ITEMS has to permit duplicates.

Now, in this case, which algorithm is faster? It depends.

Let's say we want to look up a single order. This happens in OLTP systems a lot. The loop join is probably faster. The outer loop will find a single order number, and that means the inner loop will have to probe the index on ORDER_ITEMS just once. This is true even if we have to scan the order table, based on CUSTOMER_ID and ORDER_DATE.

Now let's say we want a report for all the reports, with details for April. The merge join is probably faster. With hundreds of orders to process, walking the index on ORDER_ITEMS once beats the heck out of doing hundreds of probes.

Fully Hinting Comment by Jonathan Lewis on USENET.

Consider, for example:

```
SELECT /*+ index(t1 t1_abc) index(t2 t2_abc) */ COUNT(*)
FROM t1, t2
WHERE t1.col1 = t2.col1;
```

For weeks, this may give you the plan:

```
 NESTED LOOP
     table access by rowid t1
       index range scan t1_abc
     table access by rowid t2
       index range scan t2_abc
```

Then, because of changes in statistics, or init.ora parameters, or nullity of a column, or a few other situations that may have slipped my mind at the moment, this might change to

```
 HASH JOIN
      table access by rowid t2
        index range scan t2_abc
      table access by rowid t1
        index range scan t1_abc
```

Your hints are still obeyed, the plan has changed. On the other hand, if you had specified

```
SELECT /*+ no_parallel(t1) no_parallel(t2) no_parallel_index(t1)
no_parallel_index(t2)
ordered use_nl(t2) index(t1 t1_abc) index(t2 t2_abc) */ COUNT(*)
FROM t1, t2
WHERE t1.col1 = t2.col1;
```

Then I think you could be fairly confident that there was no way that Oracle could obey the hints whilst changing the access path.

Optimizer Approaches

ALL_ROWS

The ALL_ROWS hint explicitly chooses the cost-based approach to optimize a statement block with a goal of best throughput (that is, minimum total resource consumption).

```
/*+ ALL_ROWS */
```

```
conn / as sysdba

set linesize 121
col name format a30
col value format a30

SELECT name, value
FROM gv$parameter
WHERE name LIKE '%optimizer%';

ALTER SYSTEM SET optimizer_mode=RULE SCOPE=MEMORY;

set autotrace traceonly explain

SELECT table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;

SELECT /*+ ALL_ROWS */ table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;

ALTER SYSTEM SET optimizer_mode=ALL_ROWS SCOPE=MEMORY;
```

FIRST_ROWS(n)

The FIRST_ROWS hint explicitly chooses the cost-based approach to optimize a statement block with a goal of best response time (minimum resource usage to return first row).

This hint causes the optimizer to make these choices:

- If an index scan is available, the optimizer may choose it over a full table scan.
- If an index scan is available, the optimizer may choose a nested loops join over a sort-merge join whenever the associated table is the potential inner table of the nested loops.
- If an index scan is made available by an ORDER BY clause, the optimizer may choose it to avoid a sort operation.
- The optimizer ignores this hint in DELETE and UPDATE statement blocks and in SELECT statement blocks that contain any of the following: UNION, INTERSECT, MINUS, UNION ALL, GROUP BY, FOR UPDATE, aggregating function and the DISTINCT operator.

```
/*+ FIRST_ROWS(<integer>) */
```

```
set autotrace trace exp

SELECT table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;

SELECT /*+ FIRST_ROWS(10) */ table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;

-- the differences are subtle so look closely
```

RULE

Disables the use of the optimizer. This hint is not supported and should not be used.

```
/*+ RULE */
```

```
set autotrace trace exp
```

```
SELECT table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;

SELECT /*+ RULE */ table_name
FROM dba_tables
WHERE owner = 'SYS'
AND table_name LIKE '%$'
ORDER BY 1;
```

General

APPEND	Instructs the optimizer to use direct-path INSERT if your database is running in serial mode. Your database is in serial mode if you are not using Enterprise Edition. Conventional INSERT is the default in serial mode, and direct-path INSERT is the default in parallel mode. In direct-path INSERT, data is appended to the end of the table, rather than using existing space currently allocated to the table. As a result, direct-path INSERT can be considerably faster than conventional INSERT. When you use the **APPEND** hint for INSERT, data is simply appended to a table above the HWM which has the effect of not creating UNDO. Existing free space in blocks is not used. `/*+ APPEND */`

```
CREATE TABLE t AS
SELECT *
FROM servers
WHERE 1=2;

INSERT /*+ NO_APPEND */ INTO t
SELECT * FROM servers;

SELECT COUNT(*) FROM t;

INSERT INTO t
SELECT * FROM servers;

SELECT COUNT(*) FROM t;

INSERT /*+ APPEND */ INTO t
SELECT * FROM servers;

SELECT COUNT(*) FROM t;

COMMIT;

SELECT COUNT(*) FROM t;
```

NOAPPEND	Instructs the optimizer to use conventional INSERT by disabling parallel mode for the duration of the INSERT statement. Conventional INSERT is the default in serial mode, and direct-path INSERT is the default in parallel mode. `/*+ NOAPPEND */`

```
See APPEND Demo Above
```

CACHE	Instructs the optimizer to place the blocks retrieved for the table at the most recently used end of the LRU list in the buffer cache when a full table scan is performed. This hint is useful for small lookup tables. `/*+ CACHE([@queryblock] <tablespec>) */`

```
conn hr/hr

set autotrace traceonly exp

SELECT /*+ FULL (hr_emp) CACHE(hr_emp) */ last_name
FROM employees hr_emp;

SELECT /*+ FULL(hr_emp) NOCACHE(hr_emp) */ last_name
FROM employees hr_emp;
```

NOCACHE	Specifies that the blocks retrieved for this table are placed at the least recently used end of the LRU list in the buffer cache when a full table scan is performed. This is the normal behavior of blocks in the buffer

	cache. /*+ NOCACHE([@queryblock] <tablespec>]) */
	See CACHE Demo Above
CURSOR_SHARING_EXACT	Oracle can replace literals in SQL statements with bind variables, when it is safe to do so. This replacement is controlled with the CURSOR_SHARING initialization parameter. The CURSOR_SHARING_EXACT hint instructs the optimizer to switch this behavior off. In other words, Oracle executes the SQL statement without any attempt to replace literals with bind variables. /*+ CURSOR_SHARING_EXACT */
	conn / as sysdba ALTER SYSTEM SET cursor_sharing='SIMILAR' SCOPE=BOTH; ALTER SYSTEM FLUSH SHARED_POOL; ALTER SYSTEM FLUSH SHARED_POOL; -- as the client run two similar SQL statements SELECT latitude FROM uwclass.servers WHERE srvr_id = 1; SELECT latitude FROM uwclass.servers WHERE srvr_id = 2; SELECT latitude FROM uwclass.servers WHERE srvr_id = 3; -- as SYS look in the shared pool set linesize 121 col sql_text format a50 SELECT address, child_address, sql_text, sql_id FROM gv$sql WHERE sql_fulltext LIKE '%uwclass%'; SELECT /*+ CURSOR_SHARING_EXACT */ latitude FROM uwclass.servers WHERE srvr_id = 3; SELECT address, child_address, sql_text, sql_id FROM gv$sql WHERE sql_fulltext LIKE '%uwclass%';
DRIVING_SITE	Forces query execution to be done at a user selected site rather than at a site selected by the database. This hint is useful if you are using distributed query optimization. /*+ DRIVING_SITE([@queryblock] <tablespec>) */
	SELECT p1.first_name, p2.first_name, p2.last_name FROM person p1, person@psoug_user p2 WHERE p1.person_id = p2.person_id AND p1.first_name <> p2.first_name; SELECT /*+ DRIVING_SITE(p1) **AAA** */ p1.first_name, p2.first_name, p2.last_name FROM person p1, person@psoug_user p2 WHERE p1.person_id = p2.person_id AND p1.first_name <> p2.first_name; SELECT sql_text, remote FROM v$sql WHERE sql_text LIKE '%AAA%'; SELECT /*+ DRIVING_SITE(p2) **BBB** */ p1.first_name, p2.first_name, p2.last_name FROM person p1, person@psoug_user p2 WHERE p1.person_id = p2.person_id AND p1.first_name <> p2.first_name; SELECT sql_text, remote FROM v$sql WHERE sql_text LIKE '%BBB%';
DYNAMIC_SAMPLING	The DYNAMIC_SAMPLING hint instructs the optimizer how to control dynamic sampling to improve server performance by determining more accurate predicate selectivity and statistics for tables and indexes. You can set the value of DYNAMIC_SAMPLING to a value from 0 to 10. The higher the level, the more effort the compiler puts into dynamic sampling and the more broadly it is applied. Sampling defaults to cursor level unless you specify tablespec. The integer value is 0 to 10, indicating the degree of sampling.

	Force dynamic sampling of tables where statistics do not exist such as Global Temporary Tables. If the table is aliased the alias name, not the table name must be used `/*+ DYNAMIC_SAMPLING([@queryblock] [<tablespec>] <integer>) */`

	`conn uwclass/uwclass`
	
	`CREATE TABLE ds AS`
	`SELECT * FROM all_objects`
	`WHERE SUBSTR(object_name,1,1) BETWEEN 'A' AND 'W';`
	
	`CREATE INDEX ds_objtype`
	`ON ds(object_type);`
	
	`SELECT object_type, COUNT(*)`
	`FROM ds`
	`GROUP BY object_type;`
	
	`set autotrace trace exp`
	
	`SELECT object_name`
	`FROM ds`
	`WHERE object_type = 'JAVA CLASS';`
	
	`SELECT /*+ DYNAMIC_SAMPLING(ds 0) */ object_name`
	`FROM ds`
	`WHERE object_type = 'JAVA CLASS';`
	
	`SELECT /*+ DYNAMIC_SAMPLING(ds 4) */ object_name`
	`FROM ds`
	`WHERE object_type = 'JAVA CLASS';`
	
	`SELECT /*+ DYNAMIC_SAMPLING(ds 9) */ object_name`
	`FROM ds`
	`WHERE object_type = 'JAVA CLASS';`

MODEL_MIN_ANALYSIS	Instructs the optimizer to omit some compile-time optimizations of spreadsheet rules□primarily detailed dependency graph analysis. Other spreadsheet optimizations, such as creating filters to selectively populate spreadsheet access structures and limited rule pruning, are still used by the optimizer. This hint reduces compilation time because spreadsheet analysis can be lengthy if the number of spreadsheet rules is more than several hundreds. `/*+ MODEL_MIN_ANALYSIS */`
	TBD
MONITOR	Forces real-time SQL monitoring for the query, even if the statement is not long running. This hint is valid only when the parameter CONTROL_MANAGEMENT_PACK_ACCESS is set to DIAGNOSTIC+TUNING. /*+ MONITOR */

	`SELECT value`
	`FROM v$parameter`
	`WHERE name = 'control_management_pack_access';`
	
	`SELECT /*+ MONITOR */ COUNT(*)`
	`FROM user_tables;`

NO_MONITOR	Disables real-time SQL monitoring for the query, even if the query is long running. /*+ NO_MONITOR */

	`-- this SQL statement is made intentionally long running`
	`SELECT /*+ NO_MONITOR */ COUNT(*)`
	`FROM dba_segments s, dba_extents e`
	`WHERE s.owner = e.owner;`

OPT_PARAM	Lets you set an initialization parameter for the duration of the current query only. This hint is valid only for the following parameters: OPTIMIZER_DYNAMIC_SAMPLING, OPTIMIZER_INDEX_CACHING, OPTIMIZER_INDEX_COST_ADJ, OPTIMIZER_SECURE_VIEW_MERGING, and STAR_TRANSFORMATION_ENABLED.For example, the following hint sets the parameter STAR_TRANSFORMATION_ENABLED to TRUE for the statement to which it is added. `/*+ OPT_PARAM(parameter_name, parameter_value) */`
	`SELECT name, value`

	FROM v$parameter WHERE name LIKE 'optimizer_index%'; SELECT /*+ OPT_PARAM('optimizer_index_cost_adj' '42') */ * FROM servers;
PUSH_PRED	Instructs the optimizer to push a join predicate into the view. /*+ PUSH_PRED(<@queryblock> | <[@queryblock> <tablespec>]) */

	conn hr/hr
	
	set autotrace trace exp
	
	SELECT *
	FROM employees e, (
	  SELECT manager_id
	  FROM employees) v
	WHERE e.manager_id = v.manager_id(+)
	AND e.employee_id = 100;
	
	SELECT /*+ NO_MERGE(v) PUSH_PRED(v) */ *
	FROM employees e, (
	  SELECT manager_id
	  FROM employees) v
	WHERE e.manager_id = v.manager_id(+)
	AND e.employee_id = 100;

NO_PUSH_PRED	Instructs the optimizer not to push a join predicate into the view. /*+ NO_PUSH_PRED(<@queryblock> | <[@queryblock> <tablespec>]) */

	conn hr/hr
	
	set autotrace traceonly exp
	
	SELECT *
	FROM employees e, (
	  SELECT manager_id
	  FROM employees) v
	WHERE e.manager_id = v.manager_id(+)
	AND e.employee_id = 100;
	
	SELECT /*+ NO_MERGE(v) NO_PUSH_PRED(v) */ *
	FROM employees e, (
	  SELECT manager_id
	  FROM employees) v
	WHERE e.manager_id = v.manager_id(+)
	AND e.employee_id = 100;

PUSH_SUBQ	Instructs the optimizer to evaluate nonmerged subqueries at the earliest possible step in the execution plan. Generally, subqueries that are not merged are executed as the last step in the execution plan. If the subquery is relatively inexpensive and reduces the number of rows significantly, then evaluating the subquery earlier can improve performance. This hint has no effect if the subquery is applied to a remote table or one that is joined using a merge join. /*+ PUSH_SUBQ(<@queryblock>) */
	TBD
NO_PUSH_SUBQ	Instructs the optimizer to evaluate nonmerged subqueries as the last step in the execution plan. Doing so can improve performance if the subquery is relatively expensive or does not reduce the number of rows significantly. /*+ NO_PUSH_SUBQ(<@queryblock>) */
	TBD
PX_JOIN_FILTER	Forces the optimizer to use parallel join bitmap filtering. /*+ PX_JOIN_FILTER(<tablespec>) */
	TBD
NO_PX_JOIN_FILTER	Prevents the optimizer from using parallel join bitmap filtering. /*+ NO_PX_JOIN_FILTER(<tablespec>) */

	TBD
QB_NAME	Use the QB_NAME hint to define a name for a query block. This name can then be used in a hint in the outer query or even in a hint in an inline view to affect query execution on the tables appearing in the named query block. If two or more query blocks have the same name, or if the same query block is hinted twice with different names, then the optimizer ignores all the names and the hints referencing that query block. Query blocks that are not named using this hint have unique system-generated names. These names can be displayed in the plan table and can also be used in hints within the query block, or in query block hints. /*+ QB_NAME(<query_block_name>) */
	conn hr/hr set autotrace traceonly exp SELECT employee_id, last_name FROM employees e WHERE last_name = 'Smith'; SELECT /*+ QB_NAME(qb) FULL(@qb e) */ employee_id, last_name FROM employees e WHERE last_name = 'Smith';
RESULT_CACHE	Instructs the database to cache the results of the current query or query fragment in memory and then to use the cached results in future executions of the query or query fragment. The hint is recognized in the top-level query, the subquery_factoring_clause, or FROM clause inline view. The cached results reside in the result cache memory portion of the shared pool. /*+ RESULT_CACHE */
	TBD
NO_RESULT_CACHE	The optimizer caches query results in the result cache if the RESULT_CACHE_MODE initialization parameter is set to FORCE. In this case, the NO_RESULT_CACHE hint disables such caching for the current query. /*+ NO_RESULT_CACHE */
	TBD
Access Method Hints	
Each following hint described in this section suggests an access method for a table.	
FULL	Explicitly chooses a full table scan for the specified table /*+ FULL(<tablespec>) */
	conn uwclass/uwclass set autotrace traceonly explain SELECT latitude FROM servers WHERE srvr_id = 1; SELECT /*+ FULL(servers) */ latitude FROM servers WHERE srvr_id = 1;
INDEX	Explicitly chooses an index scan for the specified table. You can use the INDEX hint for domain, B*-tree, and bitmap indexes. However, Oracle recommends using INDEX_COMBINE rather than INDEX for bitmap indexes because it is a more versatile hint /*+ INDEX([@queryblock] <tablespec> <index_name>) */
	conn oe/oe CREATE INDEX ix_customers_gender ON customers(gender); set autotrace traceonly explain SELECT * FROM customers WHERE gender = 'M'; SELECT /*+ INDEX(customers ix_customers_gender) */ * FROM customers

	WHERE gender = 'M'; SELECT /*+ INDEX_ASC(customers ix_customers_gender) */ * FROM customers WHERE gender = 'M'; SELECT /*+ INDEX_DESC(customers ix_customers_gender) */ * FROM customers WHERE gender = 'M';
INDEX_ASC	Explicitly chooses an index scan for the specified table. If the statement uses an index range scan, Oracle scans the index entries in ascending order of their indexed values /*+ INDEX_ASC([@queryblock] <tablespec> <index_name>) */
	See INDEX Demo Above
INDEX_DESC	Explicitly chooses an index scan for the specified table. If the statement uses an index range scan, Oracle scans the index entries in descending order of their indexed values. /*+ INDEX_DESC([@queryblock] <tablespec> <indexspec>) */
	See INDEX Demo Above
NO_INDEX	Explicitly disallows a set of indexes for the specified table. The NO_INDEX hint applies to function-based, B*-tree, bitmap, cluster, or domain indexes. /*+ NO_INDEX([@queryblock] <tablespec> <indexspec>) */
	conn uwclass/uwclass set autotrace traceonly explain SELECT latitude FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id; SELECT /*+ NO_INDEX(i pk_serv_inst) */ latitude FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id;
INDEX_FFS	Causes a fast full index scan rather than a full table scan. Appears to be identical to INDEX_FFS_ASC. /*+ INDEX_FFS([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
INDEX_FFS_ASC	Causes a fast full index scan rather than a full table scan /*+ INDEX_FFS_ASC([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
INDEX_FFS_DESC	Causes a fast full index scan in descending order rather than a full table scan /*+ INDEX_FFS_DESC([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
NO_INDEX_FFS	Instructs the optimizer to exclude a fast full index scan of the specified indexes. /*+ NO_INDEX_FFS([@queryblock] <tablespec> <indexspec>) */
	conn uwclass/uwclass set autotrace traceonly exp SELECT latitude FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id; SELECT /*+ NO_INDEX_FFS(i pk_serv_inst) NO_INDEX_FFS(i ix_serv_inst) */ latitude FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id;
INDEX_RS	Instructs the optimizer to perform an index range scan for the specified table. /*+ INDEX_RS([@queryblock] <tablespec> <indexspec>) */
	TBD
INDEX_RS_ASC	Instructs the optimizer to perform an index range scan for the specified table. /*+ INDEX_RS([@queryblock] <tablespec> <indexspec>) */

	TBD
INDEX_RS_DESC	Instructs the optimizer to perform an index range scan for the specified table. /*+ INDEX_RS([@queryblock] <tablespec> <indexspec>) */
	TBD
NO_INDEX_RS	Instructs the optimizer to exclude an index range scan of the specified indexes. /*+ NO_INDEX_RS([@queryblock] <tablespec> <indexspec>) */
	conn hr/hr col column_name format a30 SELECT column_position, column_name FROM user_ind_columns WHERE index_name = 'EMP_NAME_IX'; set autotrace traceonly explain SELECT first_name FROM employees e WHERE last_name BETWEEN 'A' AND 'B'; SELECT /*+ NO_INDEX_RS(e emp_name_ix) */ last_name FROM employees e WHERE first_name BETWEEN 'A' AND 'B';
INDEX_SS	Instructs the optimizer to perform an index skip scan for the specified table. If the statement uses an index range scan, then Oracle scans the index entries in ascending order of their indexed values. In a partitioned index, the results are in ascending order within each partition. /*+ INDEX_SS([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
INDEX_SS_ASC	Instructs the optimizer to perform an index skip scan for the specified table. If the statement uses an index range scan, then Oracle Database scans the index entries in ascending order of their indexed values. In a partitioned index, the results are in ascending order within each partition. Each parameter serves the same purpose as in "INDEX Hint". The default behavior for a range scan is to scan index entries in ascending order of their indexed values, or in descending order for a descending index. This hint does not change the default order of the index, and therefore does not specify anything more than the INDEX_SS hint. However, you can use the INDEX_SS_ASC hint to specify ascending range scans explicitly should the default behavior change. /*+ INDEX_SS_ASC([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
INDEX_SS_DESC	Instructs the optimizer to perform an index skip scan for the specified table. If the statement uses an index range scan and the index is ascending, then Oracle scans the index entries in descending order of their indexed values. In a partitioned index, the results are in descending order within each partition. For a descending index, this hint effectively cancels out the descending order, resulting in a scan of the index entries in ascending order. /*+ INDEX_SS_DESC([@queryblock] <tablespec> <indexspec>) */
	See INDEX SCAN Demos Below
NO_INDEX_SS	Instructs the optimizer to exclude a skip scan of the specified indexes. /*+ NO_INDEX_SS([@queryblock] <tablespec> <indexspec>) */
	TBD
INDEX_COMBINE	Explicitly chooses a bitmap access path for the table. If no indexes are given as arguments for the INDEX_COMBINE hint, the optimizer uses whatever Boolean combination of bitmap indexes has the best cost estimate for the table. If certain indexes are given as arguments, the optimizer tries to use some Boolean combination of those particular bitmap indexes. /*+ INDEX_DESC([@queryblock] <tablespec> <indexspec>) */
	conn hr/hr set autotrace traceonly explain SELECT * FROM employees e WHERE (manager_id = 108) OR (department_id = 110);

```
SELECT /*+ INDEX_COMBINE(e emp_manager_ix emp_department_ix)
*/ *
FROM employees e
WHERE (manager_id = 108) OR (department_id = 110);
```

INDEX_JOIN

Explicitly instructs the optimizer to use an index join as an access path. For the hint to have a positive effect, a sufficiently small number of indexes must exist that contain all the columns required to resolve the query.

```
/*+ INDEX_JOIN([@queryblock] <tablespec> <indexspec>) */
```

```
conn oe/oe

set autotrace traceonly explain

SELECT department_id
FROM employees e
WHERE manager_id < 110
AND department_id < 50;

-------------------------------------------------------------------
---------
| Id | Operation                     | Name              |
Cost(%CPU)|
-------------------------------------------------------------------
---------
|  0 | SELECT STATEMENT              |                   |
2  (0) |
|* 1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES         |
2  (0) |
|* 2 |   INDEX RANGE SCAN            | EMP_DEPARTMENT_IX |
1  (0) |
-------------------------------------------------------------------
---------

SELECT /*+ INDEX_JOIN(e emp_manager_ix emp_department_ix) */
department_id
FROM employees e
  WHERE manager_id < 110
AND department_id < 50;

-------------------------------------------------------------------
-
| Id | Operation           | Name              | Cost(%CPU)
|
-------------------------------------------------------------------
-
|  0 | SELECT STATEMENT    |                   |    3 (34)
|
|* 1 |  VIEW               | index$_join$_001  |    3 (34)
|
|* 2 |   HASH JOIN         |                   |
|
|* 3 |    INDEX RANGE SCAN | EMP_DEPARTMENT_IX |    2 (50)
|
|* 4 |    INDEX RANGE SCAN | EMP_MANAGER_IX    |    2 (50)
|
-------------------------------------------------------------------
-
```

Index Scan Demos

```
conn hr/hr

col column_name format a30

SELECT column_position, column_name
FROM user_ind_columns
WHERE index_name = 'EMP_NAME_IX';

set autotrace traceonly explain

SELECT last_name
FROM employees e;

SELECT /*+ INDEX_FFS(e emp_name_ix) */ last_name
```

```
FROM employees e;

SELECT /*+ INDEX_SS(e emp_name_ix) */ last_name
FROM employees e;

SELECT /*+ INDEX_SS_ASC(e emp_name_ix) */ last_name
FROM employees e;

SELECT /*+ INDEX_DESC(e emp_name_ix) */ last_name
FROM employees e;
```

Cluster Only Access Method Hints

Each following hint can only be used with clusters.

CLUSTER

explicitly chooses a cluster scan to access the specified table. Only applies to clusters.

```
/*+  CLUSTER([@queryblock] <tablespec>) */
```

```
conn uwclass/uwclass

CREATE CLUSTER sc_srvr_id (
srvr_id NUMBER(10))
SIZE 1024;

CREATE INDEX idx_sc_srvr_id ON CLUSTER sc_srvr_id;

CREATE TABLE cservers
CLUSTER sc_srvr_id (srvr_id) AS
SELECT * FROM servers;

CREATE TABLE cserv_inst
CLUSTER sc_srvr_id (srvr_id) AS
SELECT * FROM serv_inst;

set autotrace traceonly exp

SELECT srvr_id
FROM cservers
WHERE srvr_id = 503
GROUP BY srvr_id;

SELECT /*+ CLUSTER(cservers) */ srvr_id
FROM cservers
WHERE srvr_id = 503
GROUP BY srvr_id;
```

HASH

Explicitly chooses a hash scan to access the specified table. Only applies to clusters.

```
/*+ HASH(<tablespec>) */
```

```
conn uwclass/uwclass

CREATE CLUSTER sthc_si (srvr_id NUMBER(10))
SIZE 1024 SINGLE TABLE HASHKEYS 11
TABLESPACE uwdata;

CREATE TABLE si_hash
CLUSTER sthc_si (srvr_id) AS
SELECT *
FROM serv_inst;

set autotrace traceonly explain

SELECT srvr_id
FROM si_hash
WHERE srvr_id = 503
GROUP BY srvr_id;

SELECT /*+ HASH(si_hash) */ srvr_id
FROM si_hash
WHERE srvr_id = 503
GROUP BY srvr_id;
```

Join Order

The hints in this section suggest join orders:

LEADING	Instructs the optimizer to use the specified set of tables as the prefix in the execution plan. /*+ LEADING([@queryblock] <table_name> <table_name>) */
	conn hr/hr set autotrace traceonly explain SELECT * FROM employees e, departments d, job_history j WHERE e.department_id = d.department_id AND e.hire_date = j.start_date; SELECT /*+ LEADING(e j) */ * FROM employees e, departments d, job_history j WHERE e.department_id = d.department_id AND e.hire_date = j.start_date;
ORDERED	Causes Oracle to only join tables in the order in which they appear in the FROM clause. /*+ ORDERED */
	conn oe/oe set autotrace traceonly explain SELECT o.order_id, c.customer_id, l.unit_price * l.quantity FROM customers c, order_items l, orders o WHERE c.cust_last_name = 'Mastroianni' AND o.customer_id = c.customer_id AND o.order_id = l.order_id; SELECT /*+ ORDERED */ o.order_id, c.customer_id, l.unit_price * l.quantity FROM customers c, order_items l, orders o WHERE c.cust_last_name = 'Mastroianni' AND o.customer_id = c.customer_id AND o.order_id = l.order_id;

Join Operation

Each hint described in this section suggests a join operation for a table.

USE_HASH	Causes Oracle to join each specified table with another row source with a hash join. /*+ USE_HASH([@queryblock] <tablespec> <tablespec>) */
	conn uwclass/uwclass set autotrace traceonly explain SELECT DISTINCT s.srvr_id FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id; SELECT /*+ USE_HASH (s i) */ DISTINCT s.srvr_id FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id; SELECT /*+ USE_MERGE (s i) */ DISTINCT s.srvr_id FROM servers s, serv_inst i WHERE s.srvr_id = i.srvr_id;
NO_USE_HASH	Instructs the optimizer to exclude hash joins when joining each specified table to another row source using the specified table as the inner table. /*+ NO_USE_HASH([@queryblock] <tablespec> <tablespec>) */
	TBD
USE_MERGE	Causes Oracle to join each specified table with another row source with a sort-merge join. /*+ USE_MERGE([@queryblock] <tablespec> <tablespec>) */
	See USE_HASH Demo Above

NO_USE_MERGE	Instructs the optimizer to exclude sort-merge joins when joining each specified table to another row source using the specified table as the inner table. `/*+ NO_USE_MERGE([@queryblock] <tablespec> <tablespec>) */`

	`conn hr/hr`
	
	`set autotrace traceonly explain`
	
	`SELECT *`
	`FROM employees e, departments d`
	`WHERE e.department_id = d.department_id;`
	
	`SELECT /*+ NO_USE_MERGE(e d) */ *`
	`FROM employees e, departments d`
	`WHERE e.department_id = d.department_id;`

USE_NL	Causes Oracle to join each specified table to another row source with a nested loops join using the specified table as the inner table. `/*+ USE_NL([@queryblock] <tablespec> <tablespec>) */`

	`conn uwclass/uwclass`
	
	`set autotrace traceonly explain`
	
	`SELECT DISTINCT s.srvr_id`
	`FROM servers s, serv_inst i`
	`WHERE s.srvr_id+0 = i.srvr_id+0;`
	
	`SELECT /*+ USE_NL (i s) */ DISTINCT s.srvr_id`
	`FROM servers s, serv_inst i`
	`WHERE s.srvr_id+0 = i.srvr_id+0;`

USE_NL_WITH_INDEX	Instructs the optimizer to join the specified table to another row source with a nested loops join using the specified table as the inner table. `/*+ USE_NL_WITH_INDEX([@queryblock] <tablespec> <index_name>) */`

	`conn oe/oe`
	
	`set autotrace traceonly explain`
	
	`SELECT *`
	`FROM orders h, order_items l`
	`WHERE l.order_id = h.order_id`
	`AND l.order_id > 3500;`
	
	`SELECT /*+ USE_NL_WITH_INDEX(l item_order_ix) */ *`
	`FROM orders h, order_items l`
	`WHERE l.order_id = h.order_id`
	`AND l.order_id > 3500;`

NO_USE_NL	Instructs the optimizer to exclude nested loops joins when joining each specified table to another row source using the specified table as the inner table. `/*+ NO_USE_NL([@queryblock] <tablespec> <tablespec>) */`

	`conn oe/oe`
	
	`set autotrace traceonly explain`
	
	`SELECT *`
	`FROM orders h, order_items l`
	`WHERE l.order_id = h.order_id`
	`AND l.order_id > 3500;`
	
	`SELECT /*+ NO_USE_NL(l h) */ *`
	`FROM orders h, order_items l`
	`WHERE l.order_id = h.order_id`
	`AND l.order_id > 3500;`

NATIVE_FULL_OUTER_JOIN	Instructs the optimizer to use native full outer join, which is a native execution method based on a hash join. `/*+ NATIVE_FULL_OUTER_JOIN( */`
	TBD

NO_NATIVE_FULL_OUTER_JOIN	Instructs the optimizer to exclude the native execution method when joining each specified table. Instead, the full outer join is executed as a union of left outer join and anti-join. `/*+ NO_NATIVE_FULL_OUTER_JOIN */`
	TBD
Parallel Execution	
The hints described in this section determine how statements are parallelized or not parallelized when using parallel execution.	
PARALLEL	Specify the desired number of concurrent servers that can be used for a parallel operation. The hint applies to the INSERT, UPDATE, and DELETE portions of a statement as well as to the table scan portion. If any parallel restrictions are violated, the hint is ignored. `/*+ PARALLEL([@queryblock] <tablespec> <degree | DEFAULT>) */`

	`conn hr/hr`
	
	`set autotrace traceonly exp`
	
	`SELECT last_name`
	`FROM employees hr_emp;`
	
	`SELECT /*+ FULL(hr_emp) PARALLEL(hr_emp, 2) */ last_name`
	`FROM employees hr_emp;`
	
	`-- overrides table definition and uses init parmameter`
	`SELECT /*+ FULL(hr_emp) PARALLEL(hr_emp, DEFAULT) */`
	`last_name`
	`FROM employees hr_emp;`

NO_PARALLEL	Overrides a PARALLEL specification in the table clause. In general, hints take precedence over table clauses. `/*+ NO_PARALLEL([@queryblock] <tablespec>) */`

	`conn hr/hr`
	
	`CREATE TABLE employees_demo`
	`PARALLEL (DEGREE 4) AS`
	`SELECT * FROM employees;`
	
	`SELECT table_name, degree`
	`FROM user_tables;`
	
	`set autotrace traceonly exp`
	
	`SELECT last_name`
	`FROM employees_demo hr_emp;`
	
	`SELECT /*+ NO_PARALLEL(hr_emp) */ last_name`
	`FROM employees_demo hr_emp;`

PARALLEL_INDEX	Specify the desired number of concurrent servers that can be used to parallelize index range scans for partitioned indexes. `/*+ PARALLEL_INDEX([@queryblock] <tablespec> <index_name> <degree | DEFAULT>) */`
	TBD
NO_PARALLEL_INDEX	Override a PARALLEL attribute setting on an index. In this way you can avoid a parallel index scan operation. `/*+ NO_PARALLEL_INDEX([@queryblock] <tablespec> <index_name>) */`
	TBD
PQ_DISTRIBUTE	Improve parallel join operation performance. Do this by specifying how rows of joined tables should be distributed among producer and consumer query servers. Using this hint overrides decisions the optimizer would normally make. Outer_distribution is the distribution for the outer table. Inner_distribution is the distribution for the inner table. `/*+ PQ_DISTRIBUTE([@queryblock] <tablespec> <outer_distribution> <inner_distribution>) */`

	TBD
Query Transformation	
FACT	In the context of the star transformation. It instructs the optimizer that the table specified in table specification should be considered as a fact table. `/*+ FACT([@queryblock] <tablespec>) */`
	TBD
NO_FACT	Used in the context of the star transformation. It instruct the optimizer that the queried table should not be considered as a fact table. `/*+ NO_FACT([@queryblock] <tablespec>) */`
	TBD
NO_EXPAND	Prevents the cost-based optimizer from considering OR-expansion for queries having OR conditions or INLISTS in the WHERE clause. Normally, the optimizer would consider using OR expansion and use this method if it decides the cost is lower than not using it. `/*+ NO_EXPAND(<@queryblock>);`

	`conn oe/oe`
	
	`set autotrace traceonly explain`
	
	`SELECT *`
	`FROM employees e, departments d`
	`WHERE e.manager_id = 108`
	`OR d.department_id = 110;`
	
	`SELECT /*+ NO_EXPAND */ *`
	`FROM employees e, departments d`
	`WHERE e.manager_id = 108`
	`OR d.department_id = 110;`

MERGE	The MERGE hint lets you merge views in a query. If a view's query block contains a GROUP BY clause or DISTINCT operator in the SELECT list, then the optimizer can merge the view into the accessing statement only if complex view merging is enabled. Complex merging can also be used to merge an IN subquery into the accessing statement if the subquery is uncorrelated. `/*+ MERGE(<@queryblock> [tablespec]); */`

	`conn hr/hr`
	
	`set autotrace traceonly explain`
	
	`SELECT e1.last_name, e1.salary, v.avg_salary`
	`FROM employees e1, (`
	`  SELECT department_id, AVG(salary) avg_salary`
	`  FROM employees e2`
	`  GROUP BY department_id) v`
	`WHERE e1.department_id = v.department_id`
	`AND e1.salary > v.avg_salary`
	
	`SELECT /*+ MERGE(v) */ e1.last_name, e1.salary, v.avg_salary`
	`FROM employees e1, (`
	`  SELECT department_id, AVG(salary) avg_salary`
	`  FROM employees e2`
	`  GROUP BY department_id) v`
	`WHERE e1.department_id = v.department_id`
	`AND e1.salary > v.avg_salary;`

NO_MERGE	Instructs the optimizer not to combine the outer query and any inline view queries into a single query. `/*+ NO_MERGE(<@queryblock> [tablespecification]); */`

	`conn hr/hr`
	
	`set autotrace traceonly explain`
	
	`SELECT /*+NO_MERGE(seattle_dept)*/ e1.last_name,`
	`seattle_dept.department_name`
	`FROM employees e1,`
	`(SELECT location_id, department_id, department_name`
	`FROM departments`

```
WHERE location_id = 1700) seattle_dept
WHERE e1.department_id = seattle_dept.department_id;

SELECT /*+ NO_MERGE(seattle_dept) */ e1.last_name,
seattle_dept.department_name
FROM employees e1, (
  SELECT location_id, department_id, department_name
  FROM departments
  WHERE location_id = 1700) seattle_dept
WHERE e1.department_id = seattle_dept.department_id;
```

NO_QUERY_TRANSFORMATION	Instructs the optimizer to skip all query transformations, including but not limited to OR-expansion, view merging, subquery unnesting, star transformation, and materialized view rewrite. /*+ NO_QUERY_TRANSFORMATION) */

```
conn uwclass/uwclass

set autotrace traceonly explain

SELECT DISTINCT srvr_id
FROM servers
WHERE srvr_id NOT IN (
  SELECT srvr_id
  FROM servers
  MINUS
  SELECT srvr_id
  FROM serv_inst);

SELECT /*+ NO_QUERY_TRANSFORMATION */ DISTINCT srvr_id
FROM servers
WHERE srvr_id NOT IN (
  SELECT srvr_id
  FROM servers
  MINUS
  SELECT srvr_id
  FROM serv_inst);
```

NO_REWRITE	Use on any query block of a request. This hint disables query rewrite for the query block, overriding the setting of the parameter QUERY_REWRITE_ENABLED. /*+ NO_REWRITE(<@queryblock>) */

```
conn sh/sh

set autotrace traceonly explain

SELECT SUM(s.amount_sold) AS dollars
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;

SELECT /*+ NO_REWRITE */ SUM(s.amount_sold) AS dollars
FROM sales s, times t
WHERE s.time_id = t.time_id
GROUP BY t.calendar_month_desc;
```

NO_UNNEST	Turns off unnesting of subqueries /*+ NO_UNNEST(<@queryblock>) */

```
conn uwclass/uwclass

set autotrace traceonly explain

SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT /*+ unnest */ srvr_id FROM serv_inst);

-------------------------------------------------------------
-------
| Id | Operation              | Name         | Rows | Cost
(%CPU) |
-------------------------------------------------------------
-------
|  0 | SELECT STATEMENT       |              |   11 |    5
```

```
(20) |
|* 1 |  HASH JOIN SEMI        |              |   11 |    5
(20) |
|  2 |    INDEX FULL SCAN     | PK_SERVERS   |  141 |
1    (0) |
|  3 |   INDEX FAST FULL SCAN | PK_SERVERS   |  999 |
3    (0) |
---------------------------------------------------------------
-------

Predicate Information (identified by operation id):
---------------------------------------------------

1 - access("SRVR_ID"="SRVR_ID")

SELECT srvr_id
FROM servers
WHERE srvr_id IN (
  SELECT /*+ no_unnest */ srvr_id FROM serv_inst);

---------------------------------------------------------------
---
| Id | Operation          | Name         | Rows | Cost
(%CPU) |
---------------------------------------------------------------
---
|  0 | SELECT STATEMENT   |              |    1 |  128
(0) |
|* 1 |  INDEX FULL SCAN   | PK_SERVERS   |    7 |    1
(0) |
|* 2 |   INDEX FULL SCAN  | PK_SERV_INST |    2 |    2
(0) |
---------------------------------------------------------------
---

Predicate Information (identified by operation id):
---------------------------------------------------

1 - filter( EXISTS (SELECT /*+ NO_UNNEST */ 0 FROM
"SERV_INST"
           "SERV_INST" WHERE "SRVR_ID"=:B1))
2 - access("SRVR_ID"=:B1)
    filter("SRVR_ID"=:B1)
```

REWRITE

Use with or without a view list. If you use REWRITE with a view list and the list contains an eligible materialized view, Oracle uses that view regardless of its cost. Oracle does not consider views outside of the list. If you do not specify a view list, Oracle searches for an eligible materialized view and always uses it regardless of its cost.

```
/*+ REWRITE([@queryblock] <view, view, ...>) */
```

```
conn uwclass/uwclass

CREATE MATERIALIZED VIEW mv_rewrite
TABLESPACE uwdata
REFRESH ON DEMAND
ENABLE QUERY REWRITE
AS SELECT s.srvr_id, i.installstatus, COUNT(*)
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id
GROUP BY s.srvr_id, i.installstatus;

set autotrace traceonly exp

SELECT s.srvr_id, i.installstatus, COUNT(*)
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id
AND s.srvr_id = 502
GROUP BY s.srvr_id, i.installstatus;

SELECT /*+ REWRITE */ s.srvr_id, i.installstatus, COUNT(*)
FROM servers s, serv_inst i
WHERE s.srvr_id = i.srvr_id
AND s.srvr_id = 502
```

	GROUP BY s.srvr_id, i.installstatus;
STAR_TRANSFORMATION	Makes the optimizer use the best plan in which the transformation has been used. Without the hint, the optimizer could make a cost-based decision to use the best plan generated without the transformation, instead of the best plan for the transformed query. Even if the hint is given, there is no guarantee that the transformation will take place. The optimizer will only generate the subqueries if it seems reasonable to do so. If no subqueries are generated, there is no transformed query, and the best plan for the untransformed query will be used regardless of the hint. /*+ STAR_TRANSFORMATION(<@queryblock>) */
	conn sh/sh set autotrace traceonly exp SELECT * FROM sales s, times t, products p, channels c WHERE s.time_id = t.time_id AND s.prod_id = p.prod_id AND s.channel_id = c.channel_id AND p.prod_status = 'obsolete'; SELECT /*+ STAR_TRANSFORMATION */ * FROM sales s, times t, products p, channels c WHERE s.time_id = t.time_id AND s.prod_id = p.prod_id AND s.channel_id = c.channel_id AND p.prod_status = 'obsolete';
NO_STAR_TRANSFORMATION	Instructs the optimizer not to perform star query transformation. /*+ NO_STAR_TRANSFORMATION(<@queryblock>) */
	TBD
UNNEST	Instructs the optimizer to unnest and merge the body of the subquery into the body of the query block that contains it, allowing the optimizer to consider them together when evaluating access paths and joins. /*+ UNNEST(<@queryblock>) */
	See NO_UNNEST Demo Above
USE_CONCAT	Forces combined OR conditions in the WHERE clause of a query to be transformed into a compound query using the UNION ALL set operator. Normally, this transformation occurs only if the cost of the query using the concatenations is cheaper than the cost without them. The USE_CONCAT hint turns off inlist processing and OR-expands all disjunctions, including inlists.
	conn hr/hr set autotrace traceonly explain SELECT * FROM employees e WHERE manager_id = 108 OR department_id = 110; SELECT /*+ USE_CONCAT */ * FROM employees e WHERE manager_id = 108 OR department_id = 110;
XML Hints	
NO_XMLINDEX_REWRITE	Instructs the optimizer to prohibit the rewriting of XPath expressions in SQL statements. /*+ NO_XMLINDEX_REWRITE */
	SELECT /*+ NO_XMLINDEX_REWRITE */ COUNT(*) FROM table WHERE existsNode(OBJECT_VALUE, '/*') = 1;
NO_XML_QUERY_REWRITE	Instructs the optimizer to prohibit the rewriting of XPath expressions in SQL statements. /*+ NO_XML_QUERY_REWRITE */
	SELECT /*+NO_XML_QUERY_REWRITE*/ XMLQUERY('<A/>') FROM DUAL;
Others	

ANTIJOIN	
	TBD
CARDINALITY	Instructs the optimizer to use the provided intgeger as the computed cardinality of table (tablespace) without checking. /*+ CARDINALITY(<tablespec>, <integer>) */
	conn uwclass/uwclass set autotrace traceonly explain SELECT * FROM serv_inst si WHERE srvr_id = 1; SELECT /*+ cardinality(si 999) */ * FROM serv_inst si WHERE srvr_id = 1;
NO_ACCESS	
	TBD
NO_BUFFER	
	TBD
PUSH_JOIN_PRED	Force pushing of a join predicate into the view (found in the 8.1.5 docs)

	SELECT /*+ PUSH_JOIN_PRED(v) */ T1.X, V.Y
	FROM T1 (
	  SELECT T2.X, T3.Y
	  FROM T2, T3
	  WHERE T2.X = T3.X) v
	WHERE t1.x = v.x
	AND t1.y = 1;

NO_PUSH_JOIN_PRED	Prevent pushing of a join predicate into the view
	TBD
NO_QKN_BUFF	
	TBD
NO_SEMIJOIN	
	TBD
OR_EXPAND	
	TBD
SEMIJOIN	
	TBD
SEMIJOIN_DRIVER	
	TBD
Undocumented Optimizer Hints (some of these may not actually exist)	
BITMAP	
	TBD
BUFFER	
	TBD
BYPASS_RECURSIVE_CHECK	
	TBD
BYPASS_UJVC	
	TBD
CACHE_CB	
	TBD
CACHE_TEMP_TABLE	
	TBD
CIV_GB	

	TBD
COLLECTIONS_GET_REFS	
	TBD
CPU_COSTING	
	TBD
CUBE_GB	
	TBD
DEREF_NO_REWRITE	
	TBD
DML_ UPDATE	
	TBD
DOMAIN_INDEX_NO_SORT	
	TBD
DOMAIN_INDEX_SORT	
	TBD
DYNAMIC_SAMPLING_EST_CDN	
	TBD
FORCE_SAMPLE_BLOCK	
	TBD
GBY_CONC_ROLLUP	
	TBD
HWM_BROKERED	
	TBD
IGNORE_ON_CLAUSE	
	TBD
IGNORE_ WHERE _CLAUSE	
	TBD
INDEX_RRS	
	TBD
INLINE	If you want to control the optimiser, then the 'materialize' hint makes it create a temporary table; the 'inline' hint makes it perform 'macro-substitution'. ~ Jonathan Lewis
	TBD
MATERIALIZE	If you want to control the optimiser, then the 'materialize' hint makes it create a temporary table; the 'inline' hint makes it perform 'macro-substitution'. ~ Jonathan Lewis
	TBD
LIKE_EXPAND	
	TBD
LOCAL_INDEXES	
	TBD
MV_MERGE	
	TBD
NESTED_TABLE_GET_REFS	
	TBD
NESTED_TABLE_SET_REFS	
	TBD
NESTED_TABLE_SET_SETID	
	TBD
NO_ELIMINATE	no_eliminate_oby(@mhy_view)
	TBD
NO_EXPAND_GSET_TO_UNION	
	TBD

NO_FILTERING	
	TBD
NO_ORDER_ROLLUPS	
	TBD
NO_PRUNE_GSETS	
	TBD
NO_STATS_GSETS	
	TBD
NOCPU_COSTING	
	TBD
OB_NAME	ob_name(my_view)
	TBD
OVERFLOW_NOMOVE	
	TBD
PIV_GB	
	TBD
PIV_SSF	
	TBD
PQ_MAP	
	TBD
PQ_NOMAP	
	TBD
REMOTE_MAPPED	
	TBD
RESTORE_AS_INTERVALS	
	TBD
SAVE_AS_INTERVALS	
	TBD
SCN_ASCENDING	
	TBD
SELECTIVITY	
	TBD
SKIP_EXT_OPTIMIZER	
	TBD
SQLLDR	
	TBD
SWAP_JOIN_INPUTS	
	TBD
SYS_DL_CURSOR	
	TBD
SYS_PARALLEL_TXN	
	TBD
SYS_RID_ORDER	
	TBD
TIV_GB	
	TBD
TIV_SSF	
	TBD
USE_ANTI	

	TBD
USE_SEMI	
	TBD
USE_TTT_FOR_GSETS	
	TBD
Global Hints	
Global Hints Demo	

```
conn hr/hr

CREATE OR REPLACE VIEW v AS
SELECT e1.first_name, e1.last_name, j.job_id, SUM(e2.salary)
total_sal
FROM employees e1,
    (SELECT *
     FROM employees e3) e2, job_history j
     WHERE e1.employee_id = e2.manager_id
     AND e1.employee_id = j.employee_id
     AND e1.hire_date = j.start_date
     AND e1.salary = (
       SELECT MAX(e2.salary)
       FROM employees e2
       WHERE e2.department_id = e1.department_id)
GROUP BY e1.first_name, e1.last_name, j.job_id
ORDER BY total_sal;

EXPLAIN PLAN FOR
SELECT * FROM v;

SELECT * FROM TABLE(dbms_xplan.display);

EXPLAIN PLAN FOR
SELECT /*+ INDEX(v.e2.e3 emp_job_ix) */ *
FROM v;

SELECT * FROM TABLE(dbms_xplan.display);

EXPLAIN PLAN FOR
SELECT /*+ INDEX(@SEL$2 e2.e3 emp_job_ix) */ *
FROM v;

SELECT * FROM TABLE(dbms_xplan.display);

EXPLAIN PLAN FOR
SELECT /*+ INDEX(@SEL$3 e3 emp_job_ix) */ *
FROM v;

SELECT * FROM TABLE(dbms_xplan.display);
```

Global Hints with NO_MERGE Demo

```
conn hr/hr

CREATE OR REPLACE VIEW v1 AS
SELECT *
FROM employees
WHERE employee_id < 150;

CREATE OR REPLACE VIEW v2 AS
SELECT v1.employee_id employee_id, departments.department_id
department_id
FROM v1, departments
WHERE v1.department_id = departments.department_id;

EXPLAIN PLAN FOR
SELECT *
FROM v2
WHERE department_id = 30;

SELECT * FROM TABLE(dbms_xplan.display);

EXPLAIN PLAN FOR
```

```
SELECT /*+ NO_MERGE(v2) INDEX(v2.v1.employees emp_emp_id_pk)
FULL(v2.departments) */ *
FROM v2
WHERE department_id = 30;

SELECT * FROM TABLE(dbms_xplan.display);
```

Related Topics
Histograms
Outlines
SELECT
Tuning

Home : Code Library : Sponsors : Privacy : Terms of Use : Contact Us