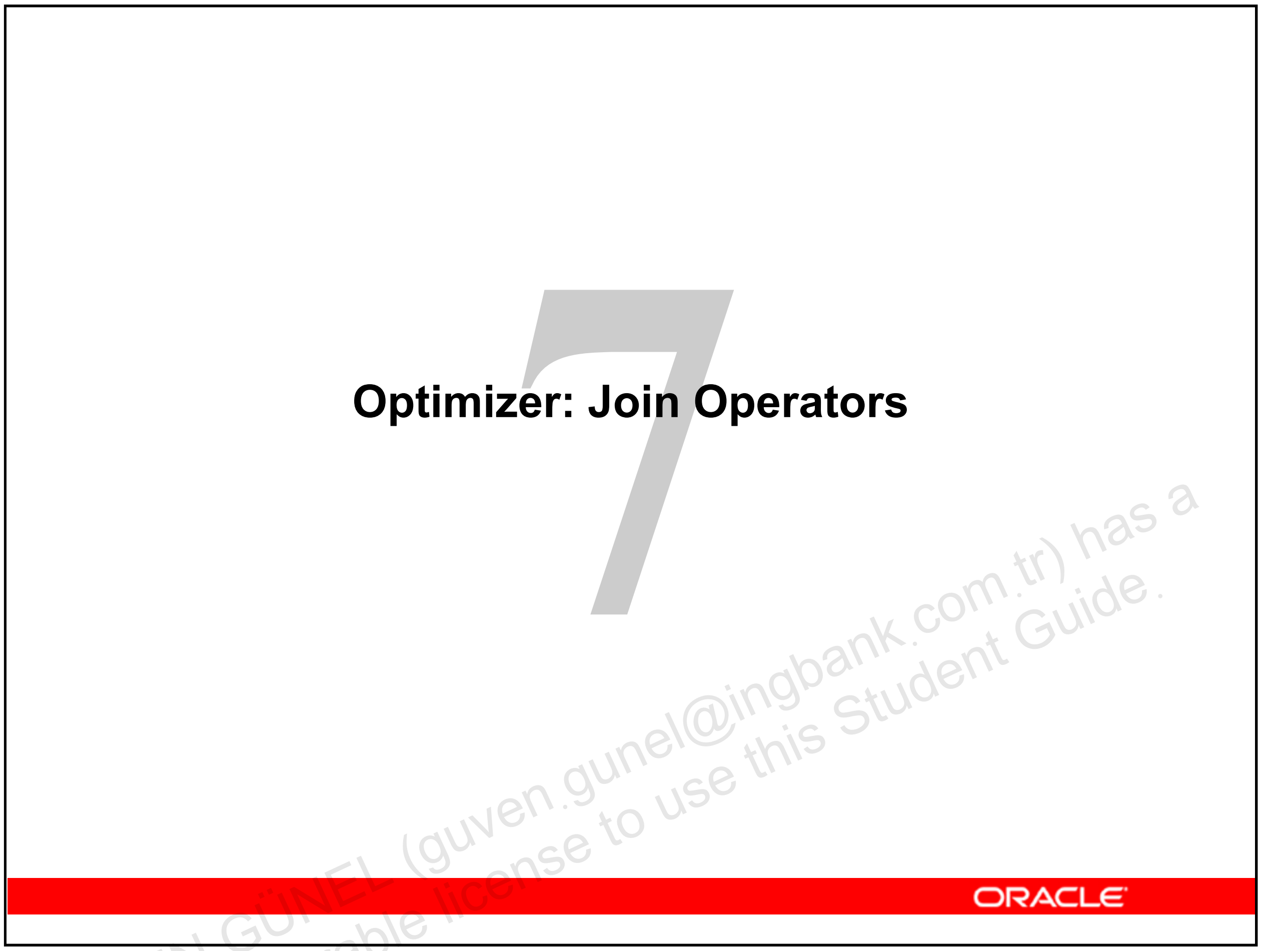
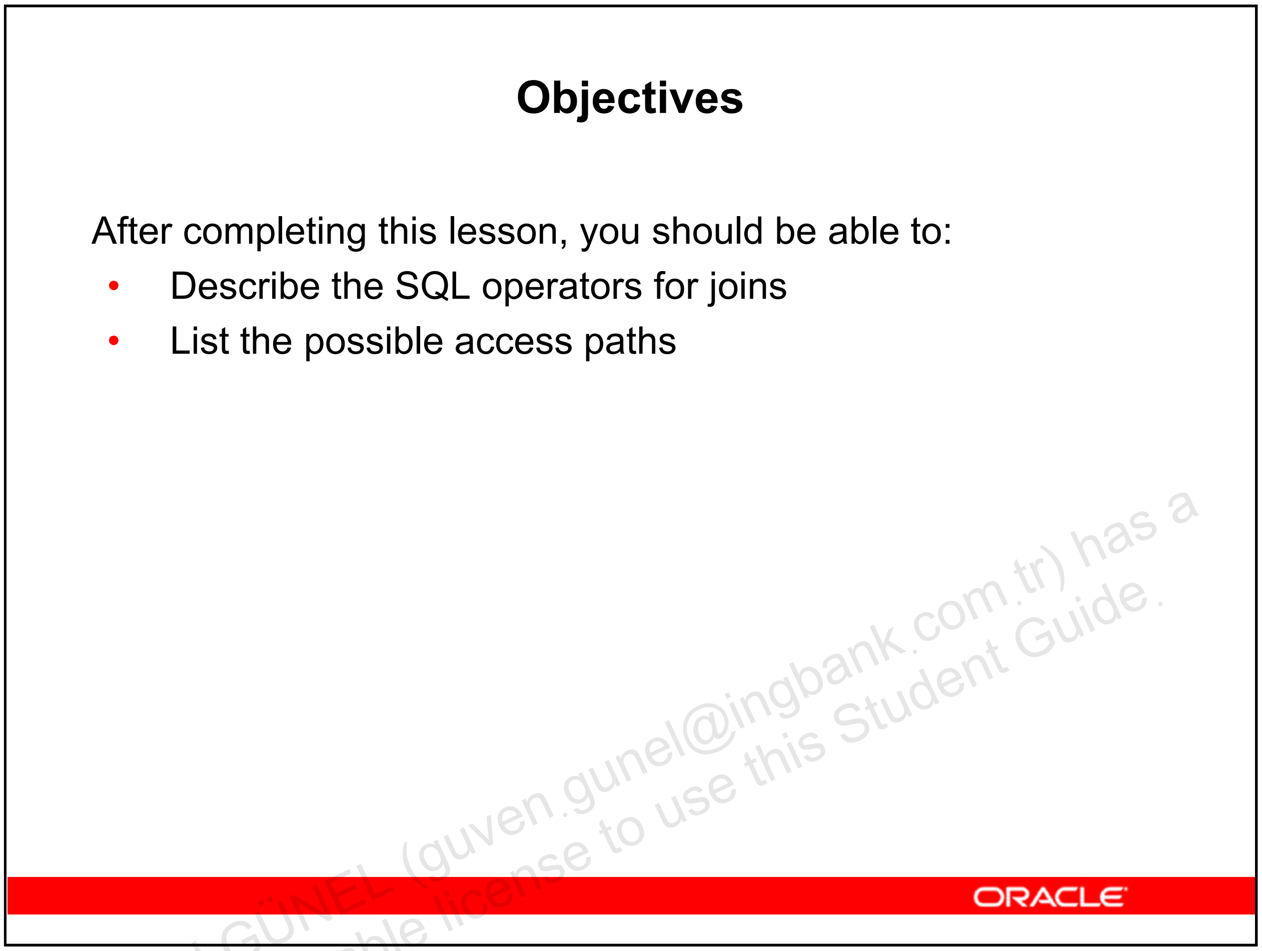
Optimizer: Join Operators

Chapter 7

Optimizer: Join Operators



Optimizer: Join Operators



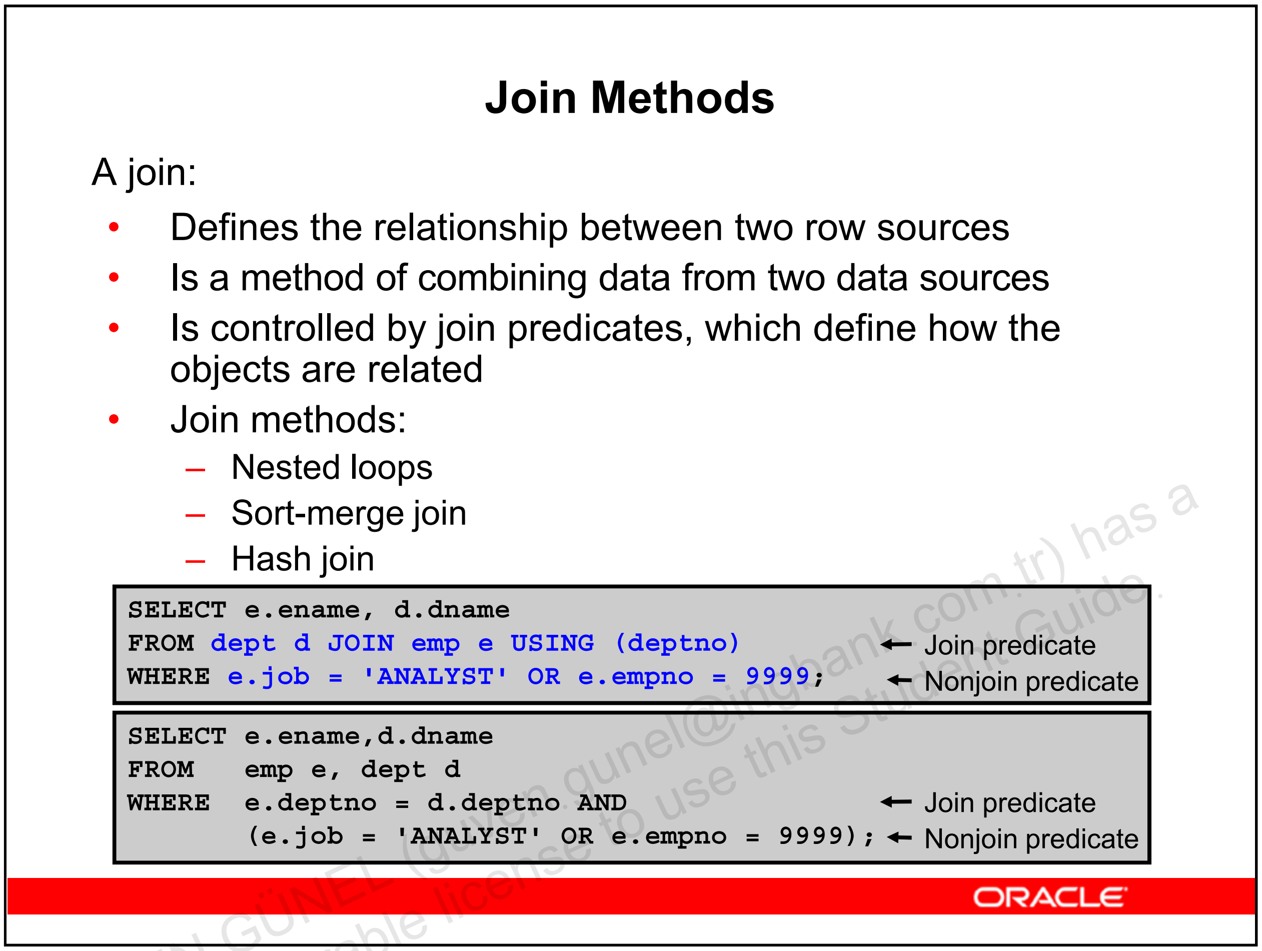
Objectives

After completing this lesson, you should be able to:

- Describe the SQL operators for joins
- List the possible access paths

Objectives

This lesson helps you understand the execution plans related to join operations.



Join Methods

A join:

- Defines the relationship between two row sources
- Is a method of combining data from two data sources
- Is controlled by join predicates, which define how the objects are related
- Join methods:
  - Nested loops
  - Sort-merge join
  - Hash join

```
SELECT e.ename, d.dname
FROM dept d JOIN emp e USING (deptno)           ← Join predicate
WHERE e.job = 'ANALYST' OR e.empno = 9999;      ← Nonjoin predicate
```

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

ORACLE

Join Methods

A row source is a set of data that can be accessed in a query. It can be a table, an index, a nonmergeable view, or even the result set of a join tree consisting of many different objects.

A join predicate is a predicate in the `WHERE` clause that combines the columns of two of the tables in the join.

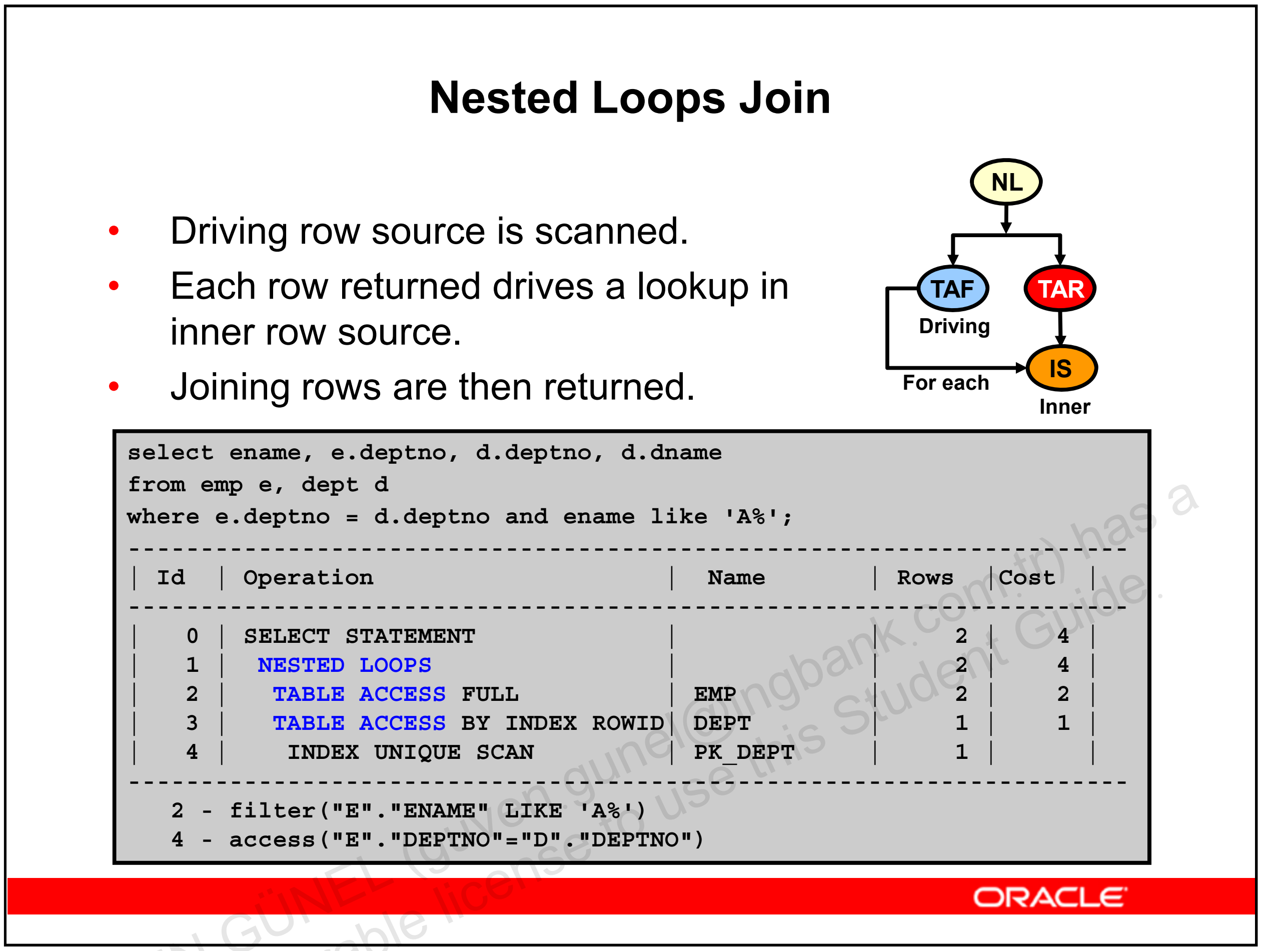
A nonjoin predicate is a predicate in the `WHERE` clause that references only one table.

A join operation combines the output from two row sources (such as tables or views) and returns one resulting row source (data set). The optimizer supports different join methods such as the following:

- **Nested loop join:** Useful when small subsets of data are being joined and if the join condition is an efficient way of accessing the second table
- **Sort-merge join:** Can be used to join rows from two independent sources. Hash joins generally perform better than sort-merge joins. On the other hand, sort-merge joins can perform better than hash joins if one or two row sources are already sorted.
- **Hash join:** Used for joining large data sets. The optimizer uses the smaller of two tables or data sources to build a hash table on the join key in memory. It then scans the larger table, probing the hash table to find the joined rows. This method is best used when the

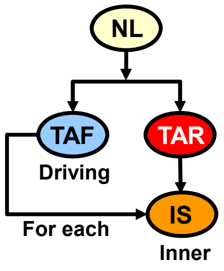
smaller table fits in the available memory. The cost is then limited to a single read pass over the data for the two tables.

Note: The slide shows you the same query using both the American National Standards Institute (ANSI) and non-ANSI join syntax. The ANSI syntax is the first example.



Nested Loops Join

- Driving row source is scanned.
- Each row returned drives a lookup in inner row source.
- Joining rows are then returned.



```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
---------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  |Cost  |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |    2  |   4  |
|   1 |  NESTED LOOPS                |               |    2  |   4  |
|   2 |   TABLE ACCESS FULL          | EMP           |    2  |   2  |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT          |    1  |   1  |
|   4 |    INDEX UNIQUE SCAN         | PK_DEPT       |    1  |      |
---------------------------------------------------------------------
   2 - filter("E"."ENAME" LIKE 'A%')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

ORACLE

Nested Loops Join

In the general form of the nested loops join, one of the two tables is defined as the outer table, or the driving table. The other table is called the inner table, or the right-hand side.

For each row in the outer (driving) table that matches the single table predicates, all rows in the inner table that satisfy the join predicate (matching rows) are retrieved. If an index is available, it can be used to access the inner table by rowid.

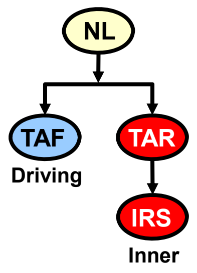
Any nonjoin predicates on the inner table are considered after this initial retrieval, unless a composite index combining both the join and the nonjoin predicate is used.

The code to emulate a nested loop join might look as follows:

```
for r1 in (select rows from EMP that match single table predicate) loop
   for r2 in (select rows from DEPT that match current row from EMP) loop
      output values from current row of EMP and current row of DEPT
    end loop
  end loop
```

The optimizer uses nested loop joins when joining small number of rows, with a good driving condition between the two tables. You drive from the outer loop to the inner loop, so the order of tables in the execution plan is important. Therefore, you should use other join methods when two independent row sources are joined.

Nested Loops Join: Prefetching



```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
---------------------------------------------------------------------
|   0 | SELECT STATEMENT            |       |   2 |  84 |   5
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT  |   1 |  22 |   1
|   2 |   NESTED LOOPS              |       |   2 |  84 |   5
|*  3 |    TABLE ACCESS FULL        | EMP   |   2 |  40 |   3
|*  4 |    INDEX RANGE SCAN         | IDEPT |   1 |     |   0
---------------------------------------------------------------------
   3 - filter("E"."ENAME" LIKE 'A%')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

ORACLE

Nested Loops Join: Prefetching

Oracle 9*i*R2 introduced a mechanism called nested loops prefetching. The idea is to improve I/O utilization, therefore response time, of index access with table lookup by batching rowid lookups into parallel block reads.

This change to the plan output is not considered a different execution plan. It does not affect the join order, join method, access method, or parallelization scheme.

This optimization is only available when the inner access path is index range scan and not if the inner access path is index unique scan.

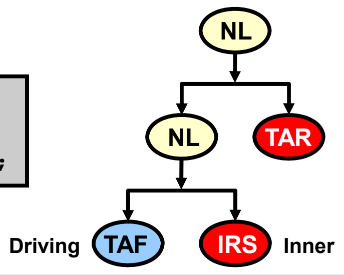
The prefetching mechanism is used by table lookup. When an index access path is chosen and the query cannot be satisfied by the index alone, the data rows indicated by the `ROWID` also must be fetched. This `ROWID` to data row access (table lookup) is improved using data block prefetching, which involves reading an array of blocks which are pointed at by an array of qualifying `ROWID`s.

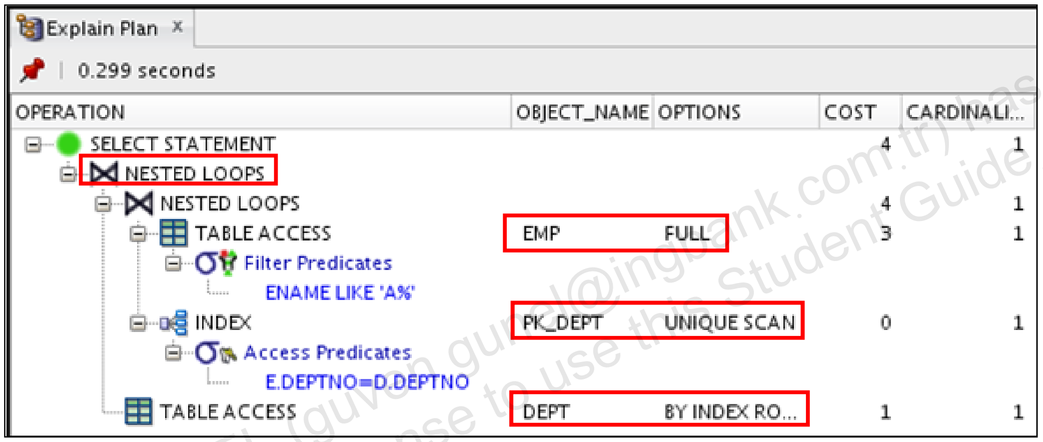
Without data block prefetching, accessing a large number of rows using a poorly clustered B*-tree index could be expensive. Each row accessed by the index would likely be in a separate data block and thus would require a separate I/O operation.

With data block prefetching, the system delays data blocks reads until multiple rows specified by the underlying index are ready to be accessed and then retrieves multiple data blocks simultaneously, rather than reading a single data block at a time.

Nested Loops Join: 11*g* Implementation

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





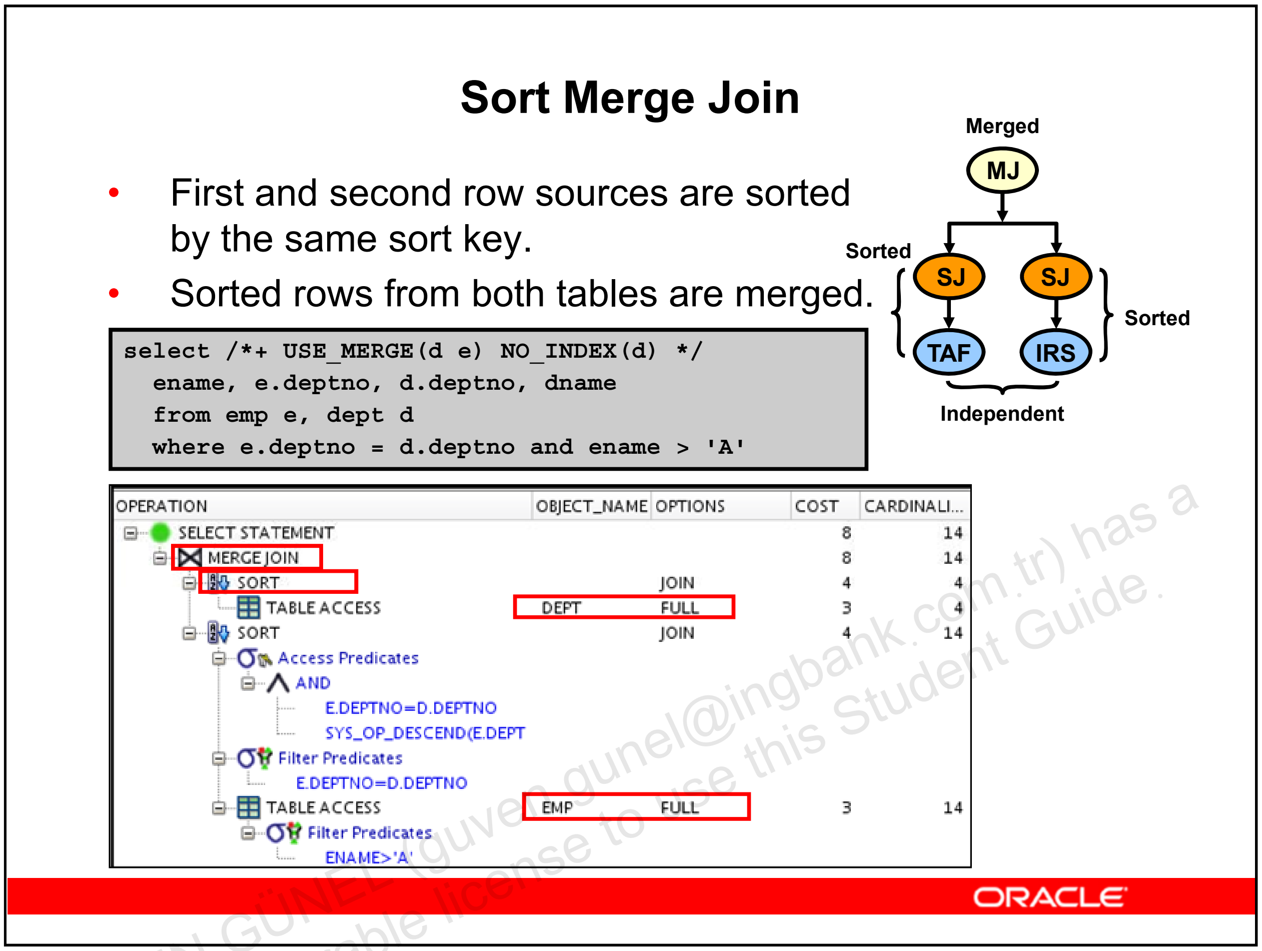
```
0.299 seconds

OPERATION                         OBJECT_NAME  OPTIONS         COST  CARDINALI...
SELECT STATEMENT                                                  4            1
  NESTED LOOPS
    NESTED LOOPS                                                  4            1
      TABLE ACCESS                EMP          FULL               3            1
        Filter Predicates
          ENAME LIKE 'A%'
      INDEX                       PK_DEPT      UNIQUE SCAN        0            1
        Access Predicates
          E.DEPTNO=D.DEPTNO
    TABLE ACCESS                  DEPT         BY INDEX RO...     1            1
```

ORACLE

Nested Loops Join: 11*g* Implementation

Oracle Database 11*g* introduces a new way of performing joins with `NESTED LOOPS` operators. With this `NESTED LOOPS` implementation, the system first performs a `NESTED LOOPS` join between the other table and the index. This produces a set of `ROWID`s that you can use to look up the corresponding rows from the table with the index. Instead of going to the table for each `ROWID` produced by the first `NESTED LOOPS` join, the system batches up the `ROWID`s and performs a second `NESTED LOOPS` join between the `ROWID`s and the table. This `ROWID` batching technique improves performance as the system only reads each block in the inner table once.

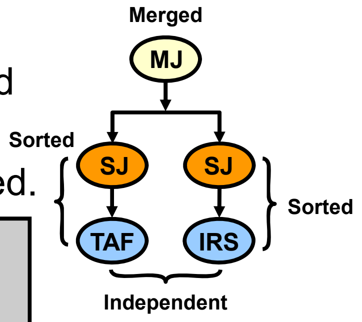
Sort Merge Join

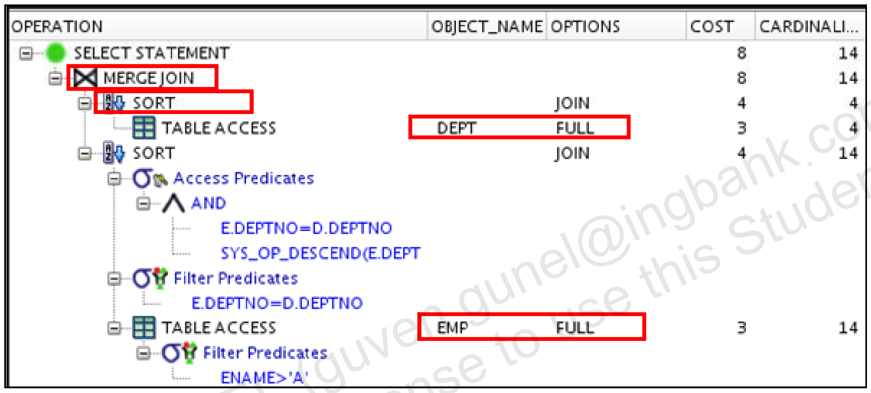


Sort Merge Join

- First and second row sources are sorted by the same sort key.
- Sorted rows from both tables are merged.

```
select /*+ USE_MERGE(d e) NO_INDEX(d) */
  ename, e.deptno, d.deptno, dname
  from emp e, dept d
  where e.deptno = d.deptno and ename > 'A'
```





```
OPERATION                          OBJECT_NAME OPTIONS   COST  CARDINALI...
SELECT STATEMENT                                            8           14
  MERGE JOIN                                                8           14
    SORT                                       JOIN         4            4
      TABLE ACCESS                 DEPT        FULL         3            4
    SORT                                       JOIN         4           14
      Access Predicates
        AND
          E.DEPTNO=D.DEPTNO
          SYS_OP_DESCEND(E.DEPT
      Filter Predicates
        E.DEPTNO=D.DEPTNO
      TABLE ACCESS                 EMP         FULL         3           14
        Filter Predicates
          ENAME>'A'
```

Sort Merge Join

In a sort merge join, there is no concept of a driving table. A sort merge join is executed as follows:

1. Get the first data set, using any access and filter predicates, and sort it on the join columns.
2. Get the second data set, using any access and filter predicates, and sort it on the join columns.
3. For each row in the first data set, find the start point in the second data set and scan until you find a row that does not join.

The merge operation combines the two sorted row sources to retrieve every pair of rows that contain matching values for the columns used in the join predicate.

If one row source has already been sorted in a previous operation (there is an index on the join column, for example), the sort merge operation skips the sort on that row source. When you perform a merge join, you must fetch all rows from the two row sources before to return the first row to the next operation. Sorting could make this join technique expensive, especially if sorting cannot be performed in memory.

The optimizer can select a sort merge join over a hash join for joining large amounts of data if any of the following conditions are true:

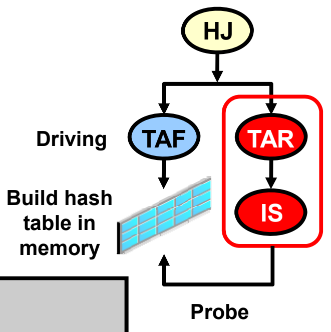
- The join condition between two tables is not an equijoin.
- Sorts already required by previous operations.

Note: Sort merge joins are useful when the join condition between two tables is an inequality condition (but not a nonequality), such as `<`, `<=`, `>`, or `>=`.

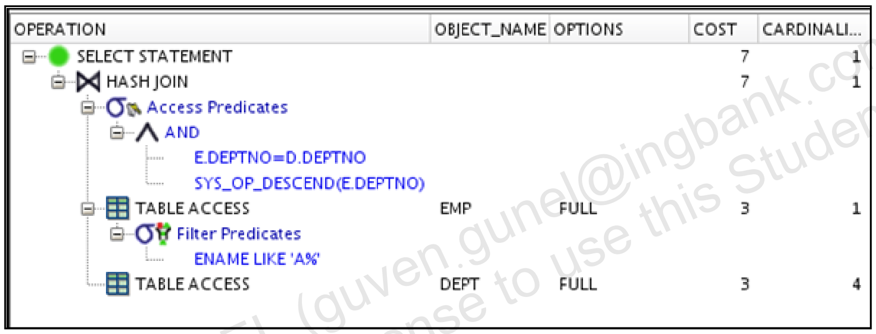
Hash Join

Hash Join

- The smallest row source is used to build a hash table.
- The second row source is hashed and checked against the hash table.



```
select /*+ USE_HASH(e d) */
  ename, e.deptno, d.deptno, dname
  from emp e, dept d
  where e.deptno = d.deptno and ename like 'A%'
```



```
OPERATION                              OBJECT_NAME OPTIONS   COST   CARDINALI...
SELECT STATEMENT                                                7          1
  HASH JOIN                                                     7          1
    Access Predicates
      AND
        E.DEPTNO=D.DEPTNO
        SYS_OP_DESCEND(E.DEPTNO)
    TABLE ACCESS                       EMP         FULL         3          1
      Filter Predicates
        ENAME LIKE 'A%'
    TABLE ACCESS                       DEPT        FULL         3          4
```

ORACLE

Hash Join

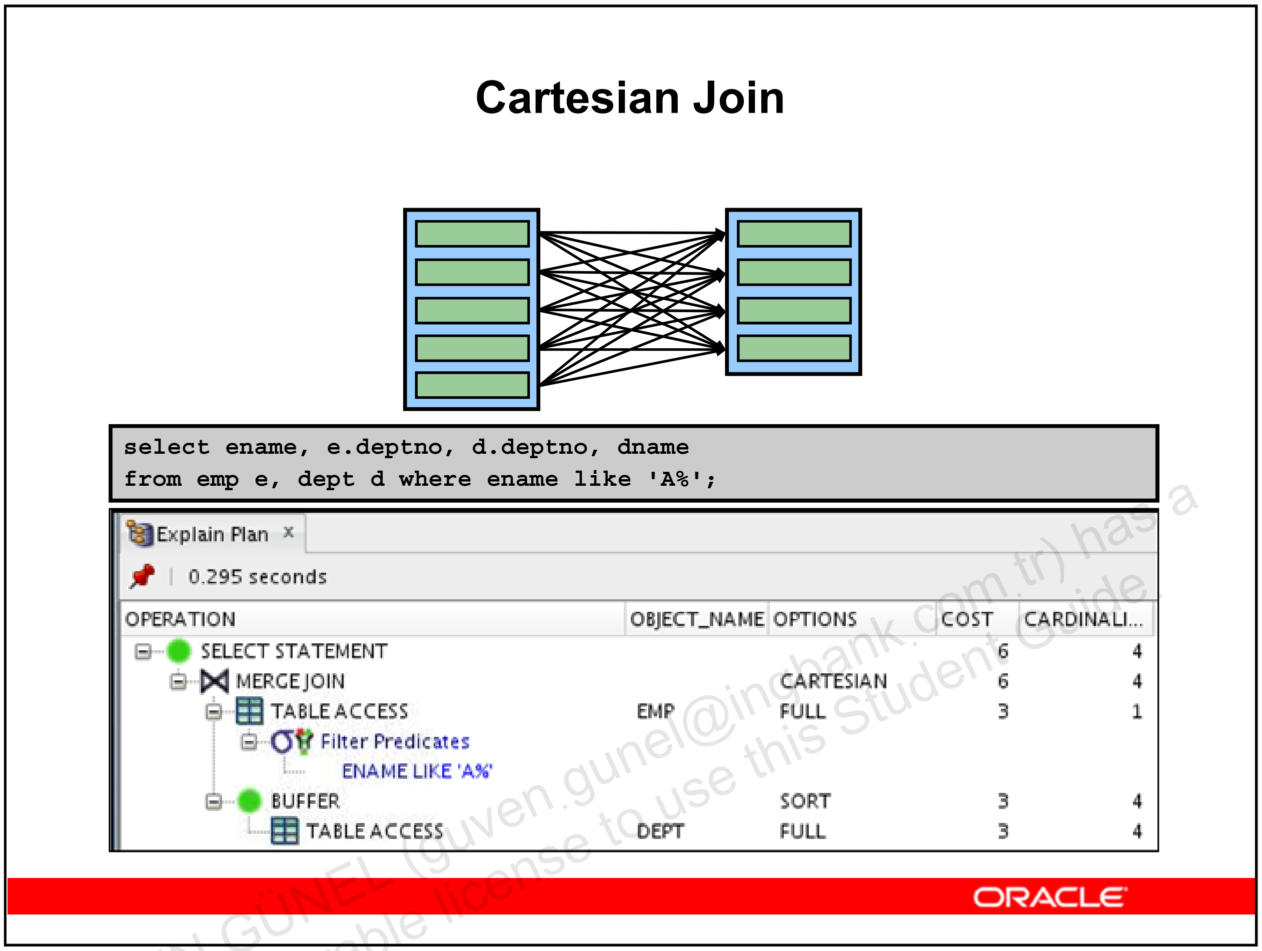
To perform a hash join between two row sources, the system reads the first data set and builds an array of hash buckets in memory. A hash bucket is little more than a location that acts as the starting point for a linked list of rows from the build table. A row belongs to a hash bucket if the bucket number matches the result that the system gets by applying an internal hashing function to the join column or columns of the row.

The system starts to read the second set of rows, using whatever access mechanism is most appropriate for acquiring the rows, and uses the same hash function on the join column or columns to calculate the number of the relevant hash bucket. The system then checks to see if there are any rows in that bucket. This is known as probing the hash table.

If there are no rows in the relevant bucket, the system can immediately discard the row from the probe table.

If there are some rows in the relevant bucket, the system does an exact check on the join column or columns to see if there is a proper match. Any rows that survive the exact check can immediately be reported (or passed on to the next step in the execution plan). So, when you perform a hash join, you must fetch all rows from the smallest row source to return the first row to next operation.

Note: Hash joins are performed only for equijoins, and are most useful when joining large amount of data.



Cartesian Join

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

Explain Plan

0.295 seconds

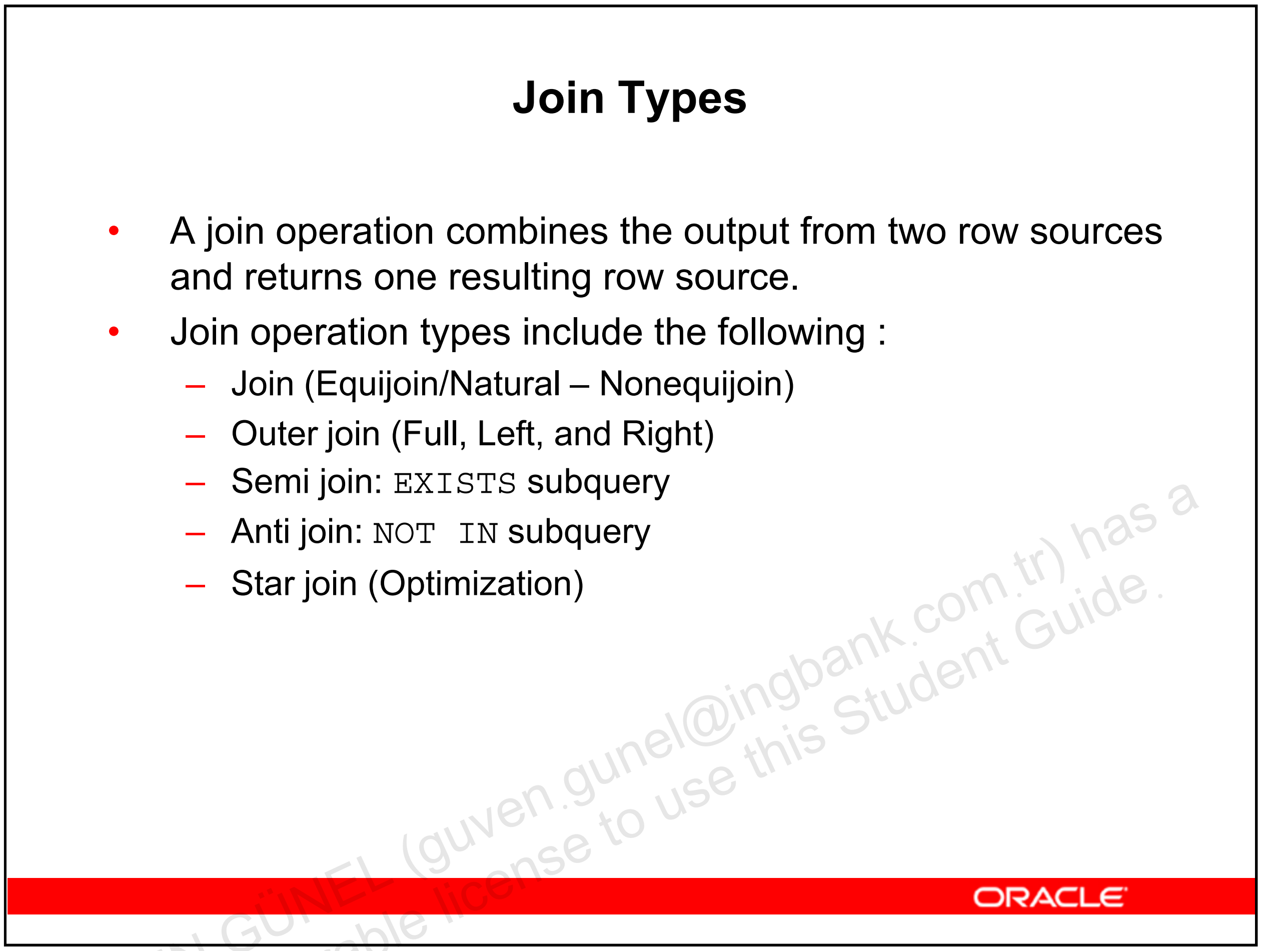
OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			6	4
MERGE JOIN		CARTESIAN	6	4
TABLE ACCESS	EMP	FULL	3	1
Filter Predicates				
ENAME LIKE 'A%'				
BUFFER		SORT	3	4
TABLE ACCESS	DEPT	FULL	3	4

Cartesian Join

A Cartesian join is used when one or more of the tables does not have any join conditions to any other tables in the statement. The optimizer joins every row from one data source with every row from the other data source, creating the Cartesian product of the two sets.

A Cartesian join can be seen as a nested loop with no elimination; the first row source is read and then for every row, all the rows are returned from the other row source.

Note: Cartesian join is generally not desirable. However, it is perfectly acceptable to have one with single-row row source (guaranteed by a unique index, for example) joined to some other table.



Join Types

- A join operation combines the output from two row sources and returns one resulting row source.
- Join operation types include the following :
  - Join (Equijoin/Natural – Nonequijoin)
  - Outer join (Full, Left, and Right)
  - Semi join: `EXISTS` subquery
  - Anti join: `NOT IN` subquery
  - Star join (Optimization)

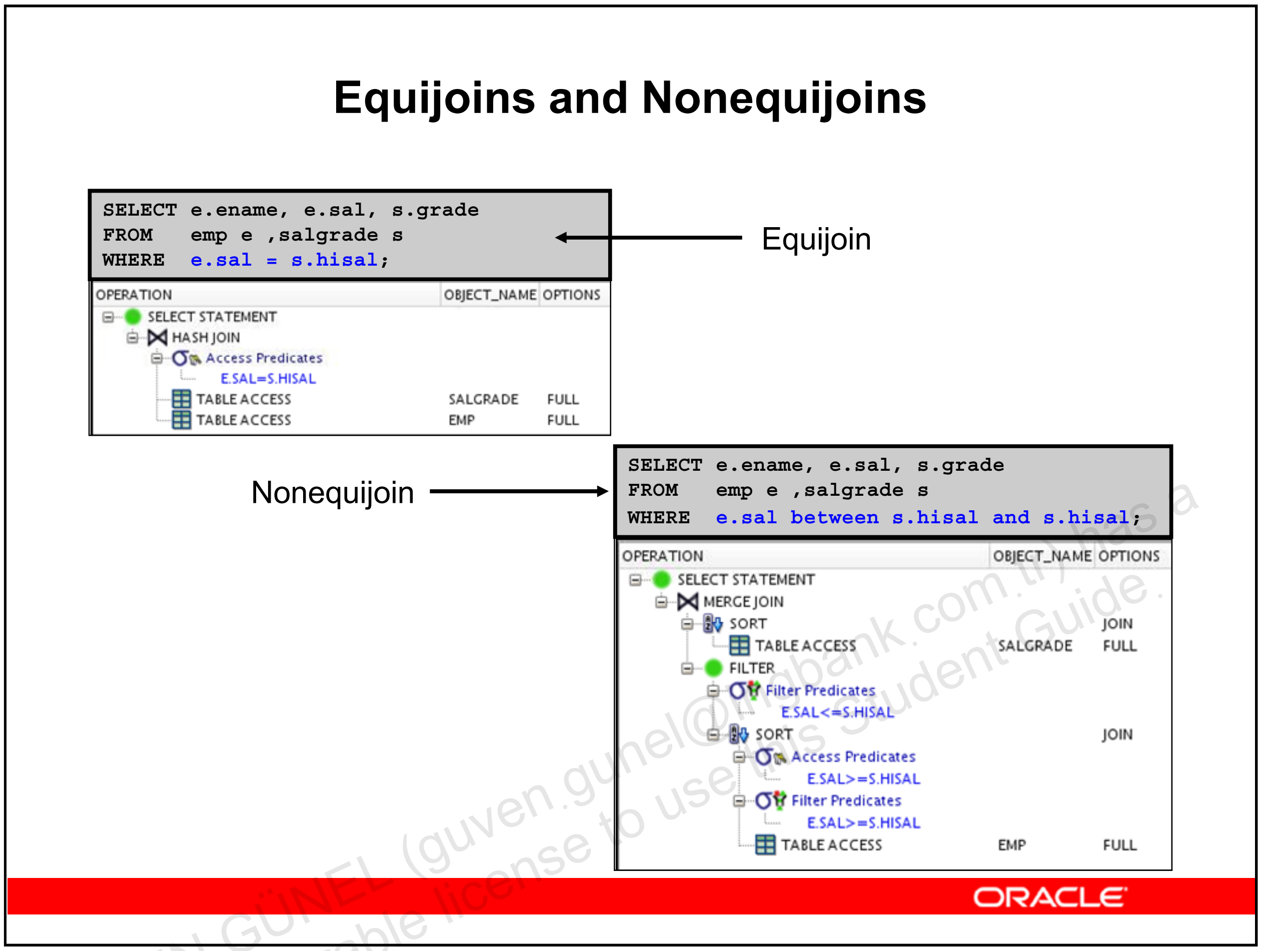
Join Types

Join operation types include the following:

- **Join (equijoin and nonequijoin):** Returns rows that match predicate join
- **Outer join:** Returns rows that match predicate join and row when no match is found
- **Semi join:** Returns rows that match the `EXISTS` subquery. Find one match in the inner table, then stop search.
- **Anti join:** Returns rows with no match in the `NOT IN` subquery. Stop as soon as one match is found.
- **Star join:** This is not a join type, but just a name for an implementation of a performance optimization to better handle the fact and dimension model.

Antijoin and semijoin are considered to be join types, even though the SQL constructs that cause them are subqueries. Antijoin and semijoin are internal optimizations algorithms used to flatten subquery constructs in such a way that they can be resolved in a join-like way.

Equijoins and Nonequijoins



Equijoins and Nonequijoins

Equijoin

```
SELECT e.ename, e.sal, s.grade
FROM   emp e ,salgrade s
WHERE  e.sal = s.hisal;
```

```
OPERATION                              OBJECT_NAME OPTIONS
SELECT STATEMENT
  HASH JOIN
    Access Predicates
      E.SAL=S.HISAL
    TABLE ACCESS                       SALGRADE    FULL
    TABLE ACCESS                       EMP         FULL
```

Nonequijoin

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

```
OPERATION                              OBJECT_NAME OPTIONS
SELECT STATEMENT
  MERGE JOIN
    SORT                                           JOIN
      TABLE ACCESS                     SALGRADE    FULL
    FILTER
      Filter Predicates
        E.SAL<=S.HISAL
      SORT                                         JOIN
        Access Predicates
          E.SAL>=S.HISAL
        Filter Predicates
          E.SAL>=S.HISAL
        TABLE ACCESS                   EMP         FULL
```

ORACLE

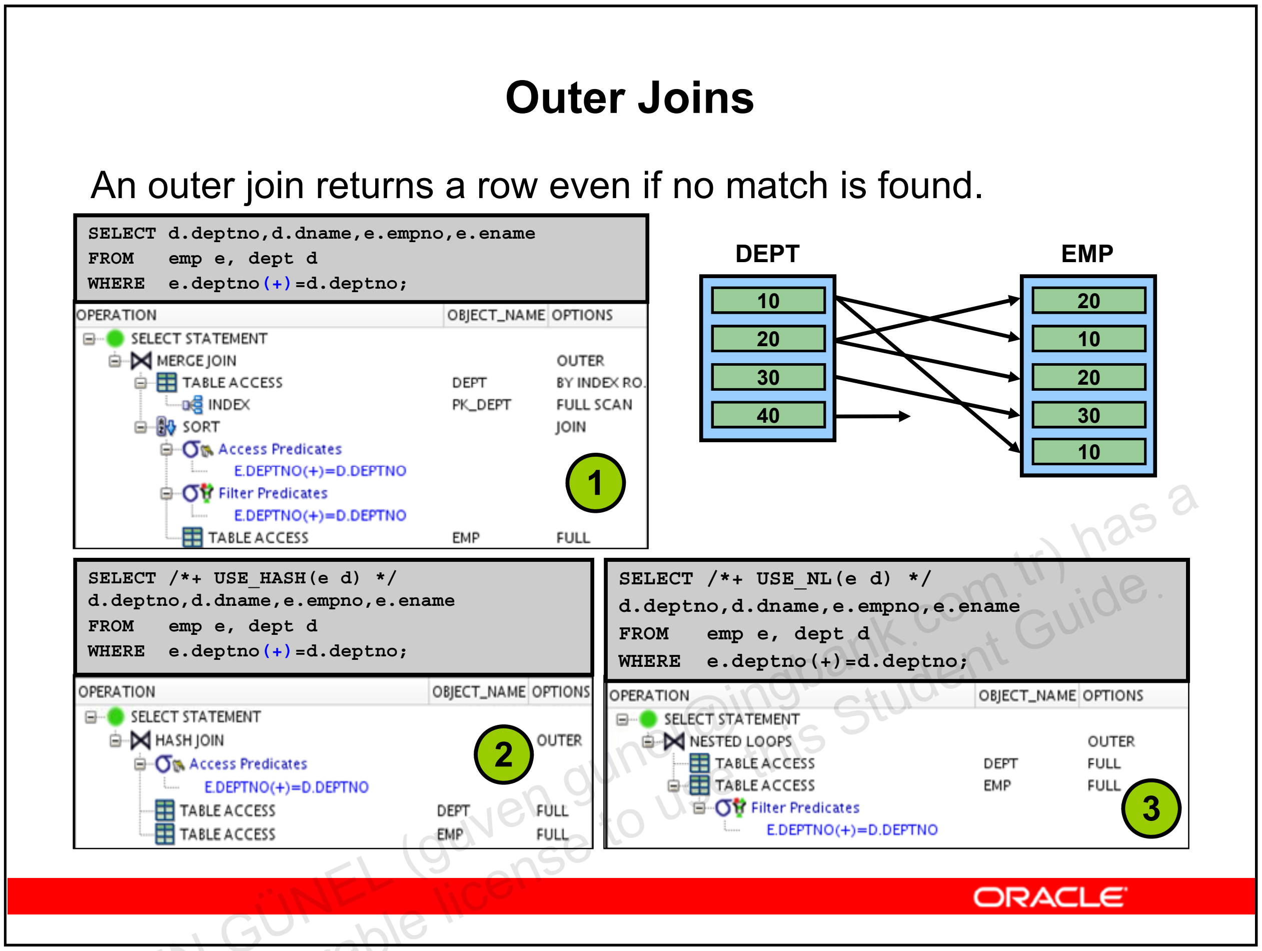
Equijoins and Nonequijoins

The join condition determines whether a join is an equijoin or a nonequijoin. An equijoin is a join with a join condition containing an equality operator. When a join condition relates two tables by an operator other than equality, it is a nonequijoin.

Equijoins are the most commonly used. An example each of an equijoin and a nonequijoin are shown in the slide. Nonequijoins are less frequently used.

To improve SQL efficiency, use equijoins whenever possible. Statements that perform equijoins on untransformed column values are the easiest to tune.

Note: If you have a nonequijoin, a hash join is not possible.



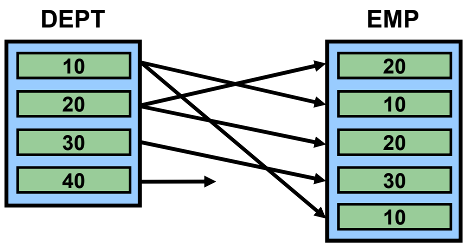
Outer Joins

An outer join returns a row even if no match is found.

```
SELECT d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;
```

```
OPERATION                          OBJECT_NAME  OPTIONS
SELECT STATEMENT
  MERGE JOIN                                    OUTER
    TABLE ACCESS                   DEPT         BY INDEX RO
      INDEX                        PK_DEPT      FULL SCAN
    SORT                                        JOIN
      Access Predicates
        E.DEPTNO(+)=D.DEPTNO
      Filter Predicates
        E.DEPTNO(+)=D.DEPTNO
      TABLE ACCESS                 EMP          FULL
```

1



```
SELECT /*+ USE_HASH(e d) */
d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;
```

```
OPERATION                          OBJECT_NAME  OPTIONS
SELECT STATEMENT
  HASH JOIN                                     OUTER
    Access Predicates
      E.DEPTNO(+)=D.DEPTNO
    TABLE ACCESS                   DEPT         FULL
    TABLE ACCESS                   EMP          FULL
```

2

```
SELECT /*+ USE_NL(e d) */
d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;
```

```
OPERATION                          OBJECT_NAME  OPTIONS
SELECT STATEMENT
  NESTED LOOPS                                  OUTER
    TABLE ACCESS                   DEPT         FULL
    TABLE ACCESS                   EMP          FULL
      Filter Predicates
        E.DEPTNO(+)=D.DEPTNO
```

3

ORACLE

Outer Joins

The simple join is the most commonly used within the system. Other joins open up extra functionality, but have much more specialized uses. The outer join operator is placed on the deficient side of the query. In other words, it is placed against the table that has the missing join information. Consider EMP and DEPT. There may be a department that has no employees. If EMP and DEPT are joined together, this particular department would not appear in the output because there is no row that matches the join condition for that department. By using the outer join, the missing department can be displayed.

1. **Merge Outer joins:** By default, the optimizer uses MERGE OUTER JOIN.
2. **Outer join with nested loops:** The left/driving table is always the table whose rows are being preserved (DEPT in the example). For each row from DEPT, look for all matching rows in EMP. If none is found, output DEPT values with null values for the EMP columns. If rows are found, output DEPT values with these EMP values.
3. **Hash Outer joins:** The left/outer table whose rows are being preserved is used to build the hash table, and the right/inner table is used to probe the hash table. When a match is found, the row is output and the entry in the hash table is marked as matched to a row. After the inner table is exhausted, the hash table is read over once again, and any rows that are not marked as matched are output with null values for the EMP columns. The system hashes the table whose rows are not being preserved, and then reads the

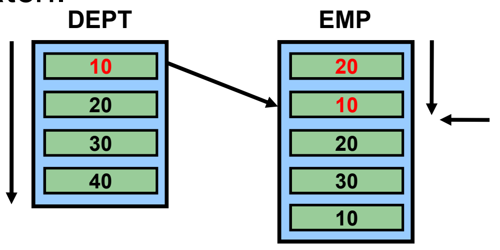
table whose rows are being preserved, probing the hash table to see whether there was a row to join to.

Note: You can also use the ANSI syntax for full, left, and right outer joins (not shown in the slide).

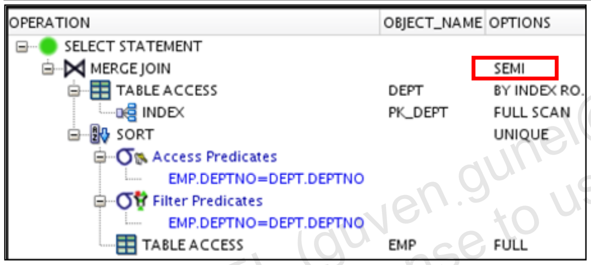
Semijoins

Semijoins

Semijoins look only for the first match.



```
SELECT deptno, dname
FROM   dept
WHERE  EXISTS (SELECT 1 FROM emp WHERE emp.deptno=dept.deptno);
```



```
OPERATION                              OBJECT_NAME  OPTIONS
SELECT STATEMENT
  MERGE JOIN                                        SEMI
    TABLE ACCESS                       DEPT         BY INDEX RO
      INDEX                            PK_DEPT      FULL SCAN
    SORT                                            UNIQUE
      Access Predicates
        EMP.DEPTNO=DEPT.DEPTNO
      Filter Predicates
        EMP.DEPTNO=DEPT.DEPTNO
      TABLE ACCESS                     EMP          FULL
```

Semijoins

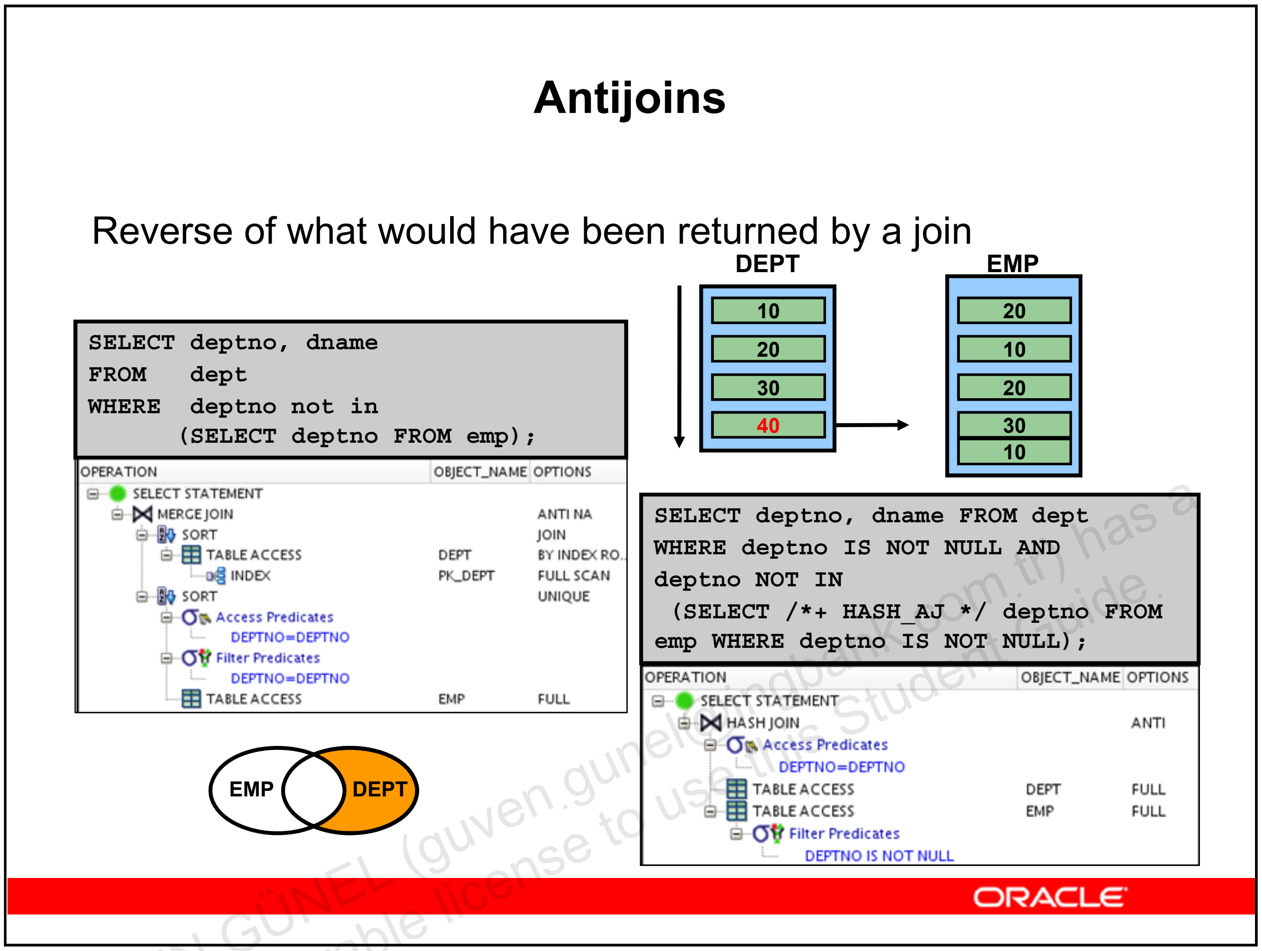
Semijoins return a result when you hit the first joining record. A semijoin is an internal way of transforming an `EXISTS` subquery into a join. However, you cannot see this occur anywhere.

Semijoins return rows that match an `EXISTS` subquery without duplicating rows from the left side of the predicate when multiple rows on the right side satisfy the criteria of the subquery.

In the above diagram, for each `DEPT` record, only the first matching `EMP` record is returned as a join result. This prevents scanning huge numbers of duplicate rows in a table when all you are interested in is if there are any matches.

When the subquery is not unnested, a similar result could be achieved by using a `FILTER` operation and scanning a row source until a match is found, then returning it.

Note: A semijoin can always use a Merge join. The optimizer may choose nested-loop, or hash joins methods to perform semijoins as well.



Antijoins

Reverse of what would have been returned by a join

```
SELECT deptno, dname
FROM   dept
WHERE  deptno not in
      (SELECT deptno FROM emp);
```

```
OPERATION                              OBJECT_NAME  OPTIONS
SELECT STATEMENT
  MERGE JOIN                                        ANTI NA
    SORT                                            JOIN
      TABLE ACCESS                     DEPT         BY INDEX RO..
        INDEX                          PK_DEPT      FULL SCAN
    SORT                                            UNIQUE
      Access Predicates
          DEPTNO=DEPTNO
      Filter Predicates
          DEPTNO=DEPTNO
      TABLE ACCESS                     EMP          FULL
```

```
SELECT deptno, dname FROM dept
WHERE deptno IS NOT NULL AND
deptno NOT IN
 (SELECT /*+ HASH_AJ */ deptno FROM
emp WHERE deptno IS NOT NULL);
```

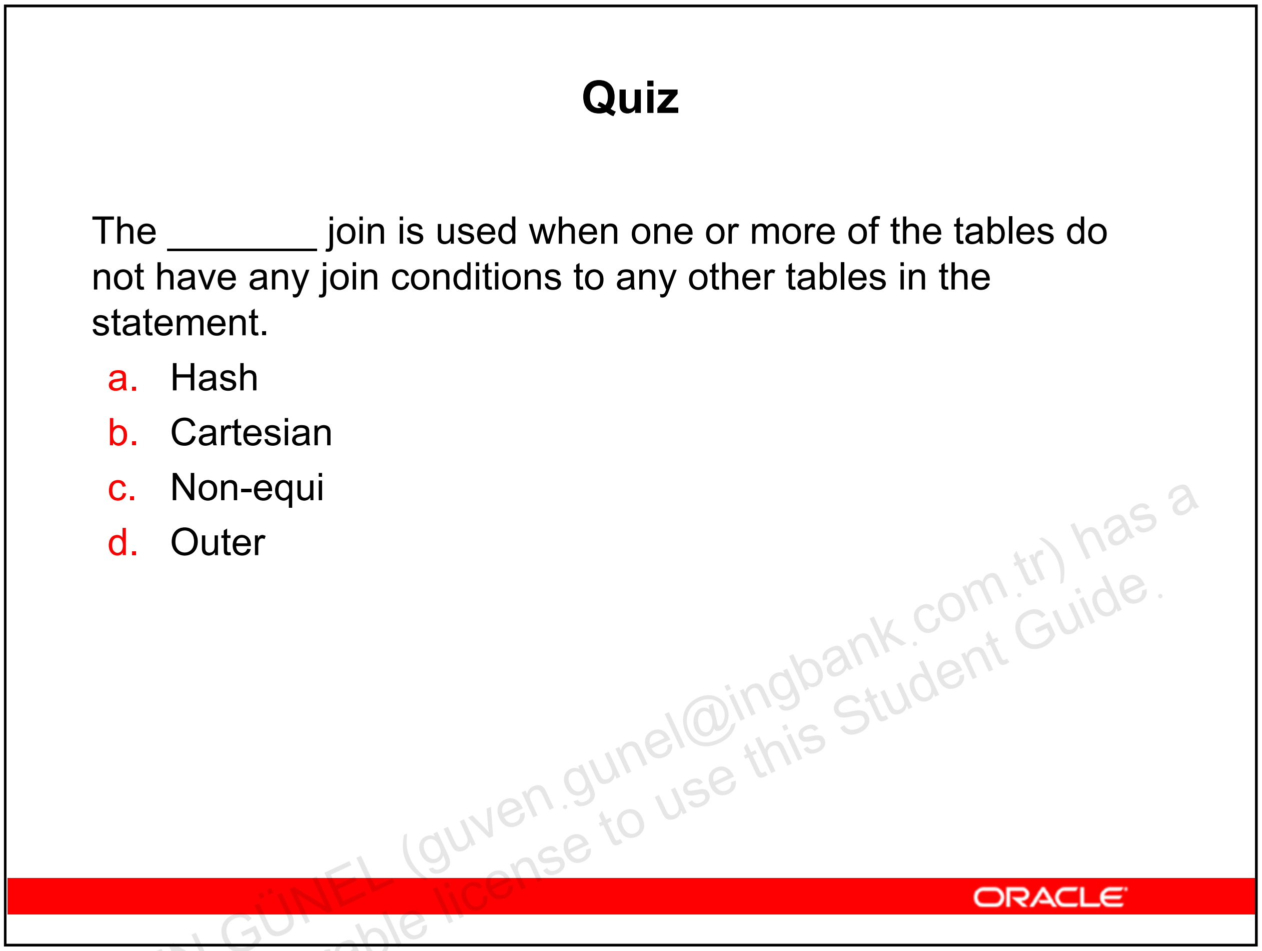
```
OPERATION                              OBJECT_NAME  OPTIONS
SELECT STATEMENT
  HASH JOIN                                         ANTI
    Access Predicates
        DEPTNO=DEPTNO
    TABLE ACCESS                       DEPT         FULL
    TABLE ACCESS                       EMP          FULL
      Filter Predicates
          DEPTNO IS NOT NULL
```

Antijoins

Antijoins return rows that fail to match (NOT IN) the subquery at the right side. For example, an antijoin can select a list of departments which do not have any employee.

The optimizer uses a merge antijoin algorithm for NOT IN subqueries by default. However, if the HASH_AJ or NL_AJ hints are used and various required conditions are met, the NOT IN uncorrelated subquery can be changed. Although antijoins are mostly transparent to the user, it is useful to know that these join types exist and could help explain unexpected performance changes between releases.

Quiz

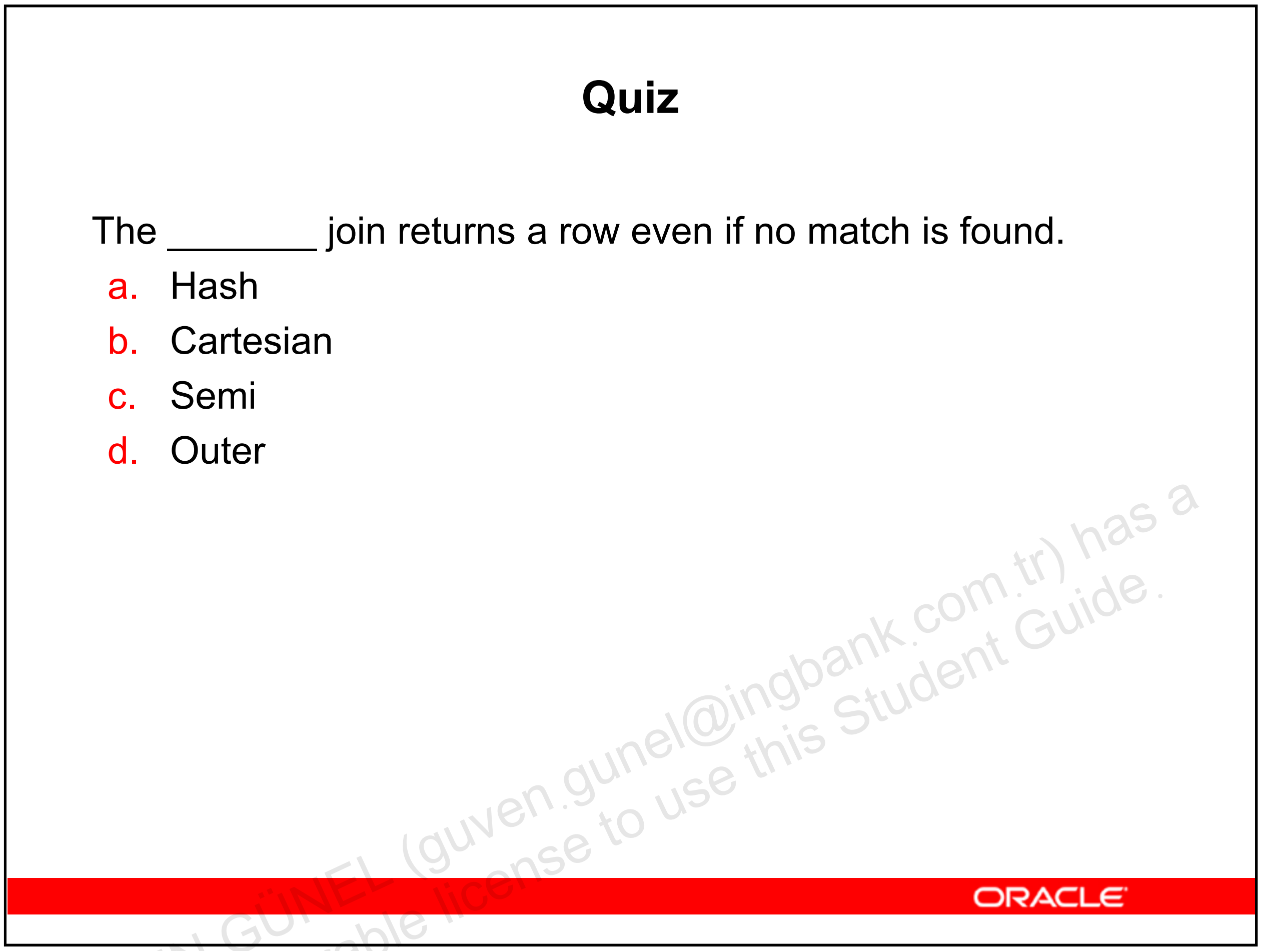


Quiz

The ________ join is used when one or more of the tables do not have any join conditions to any other tables in the statement.

a. Hash
b. Cartesian
c. Non-equi
d. Outer

Answer: b

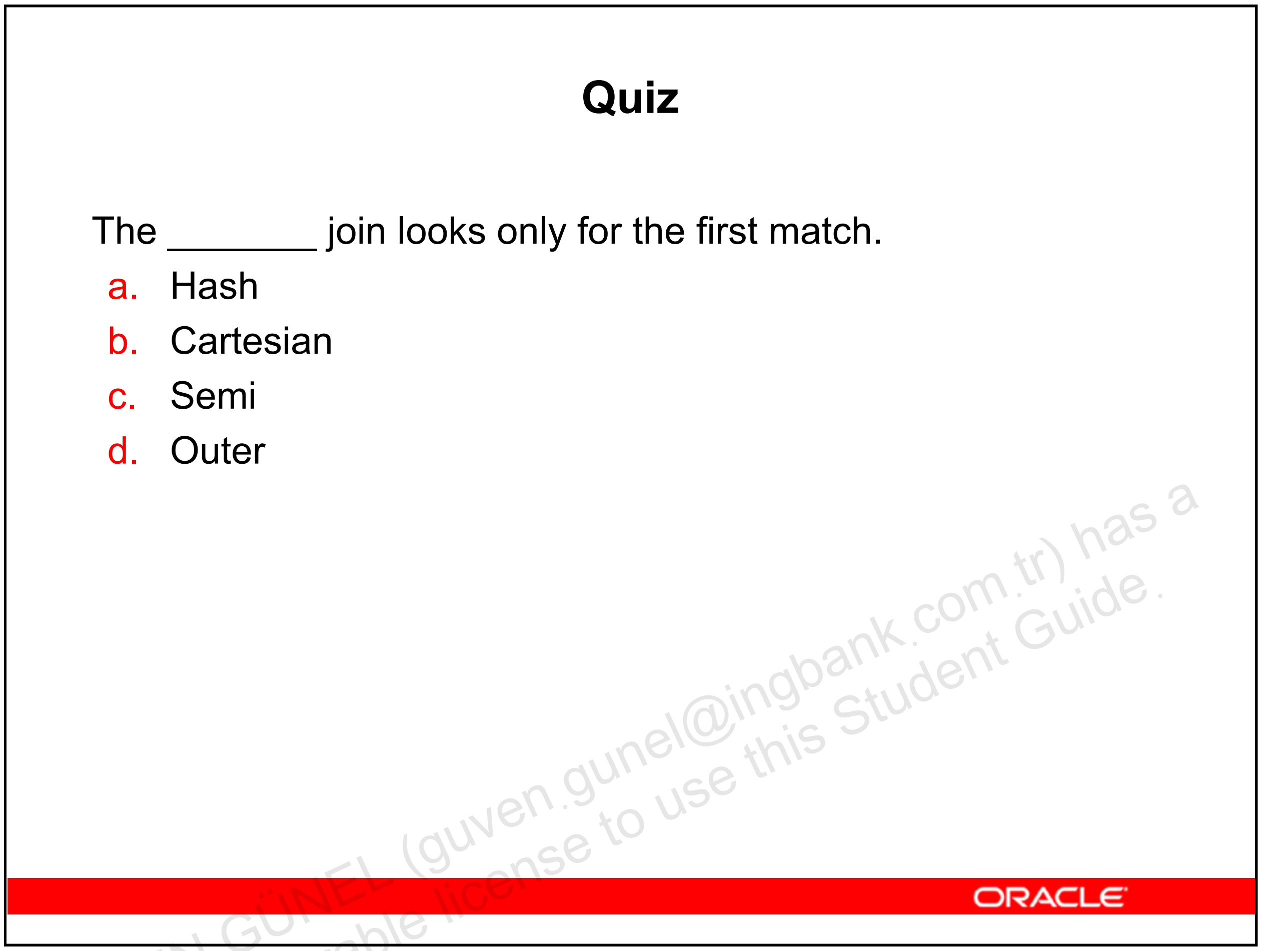


Quiz

The ________ join returns a row even if no match is found.

a. Hash
b. Cartesian
c. Semi
d. Outer

Answer: d

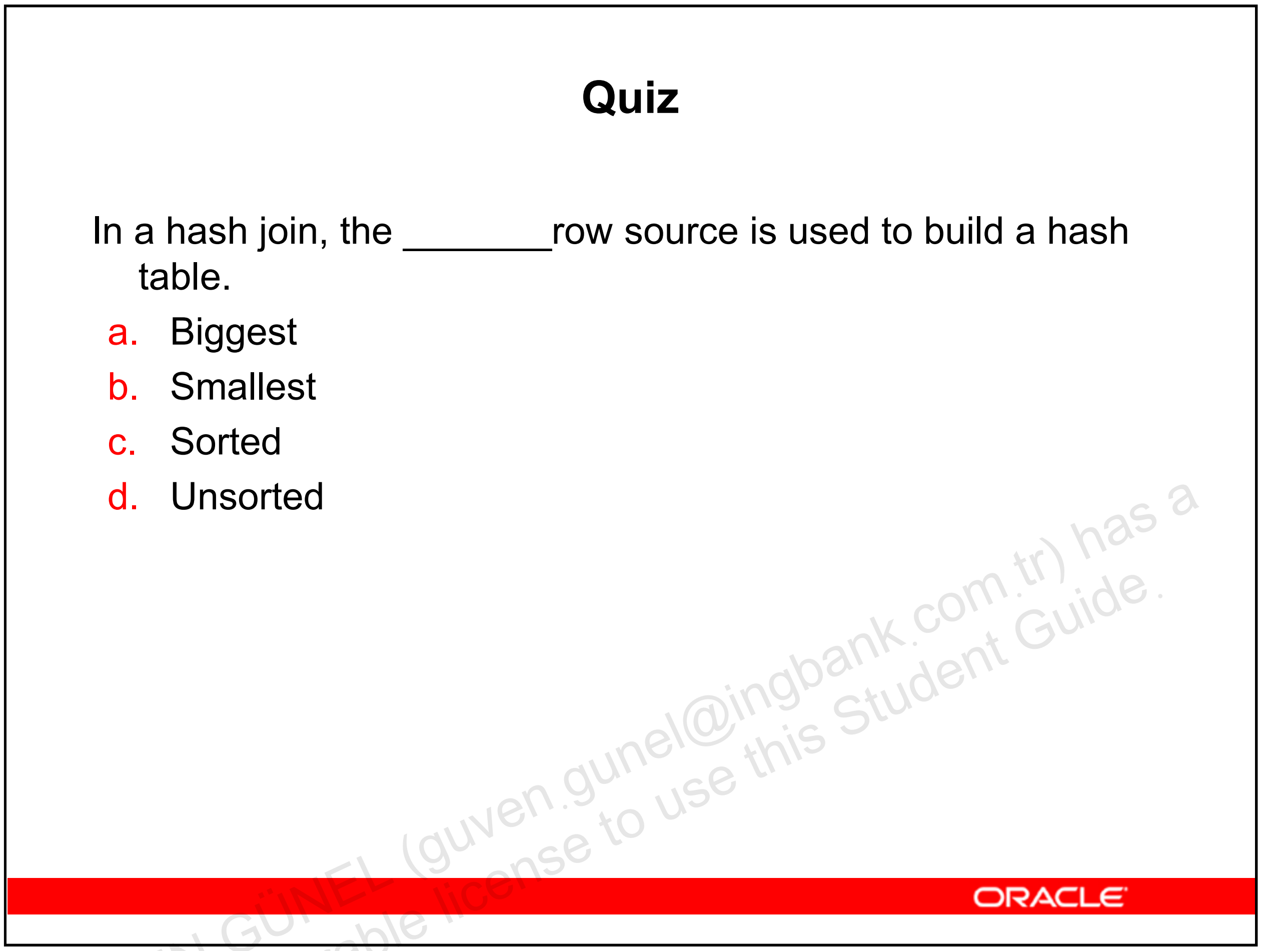


Quiz

The ________ join looks only for the first match.

a. Hash
b. Cartesian
c. Semi
d. Outer

Answer: c

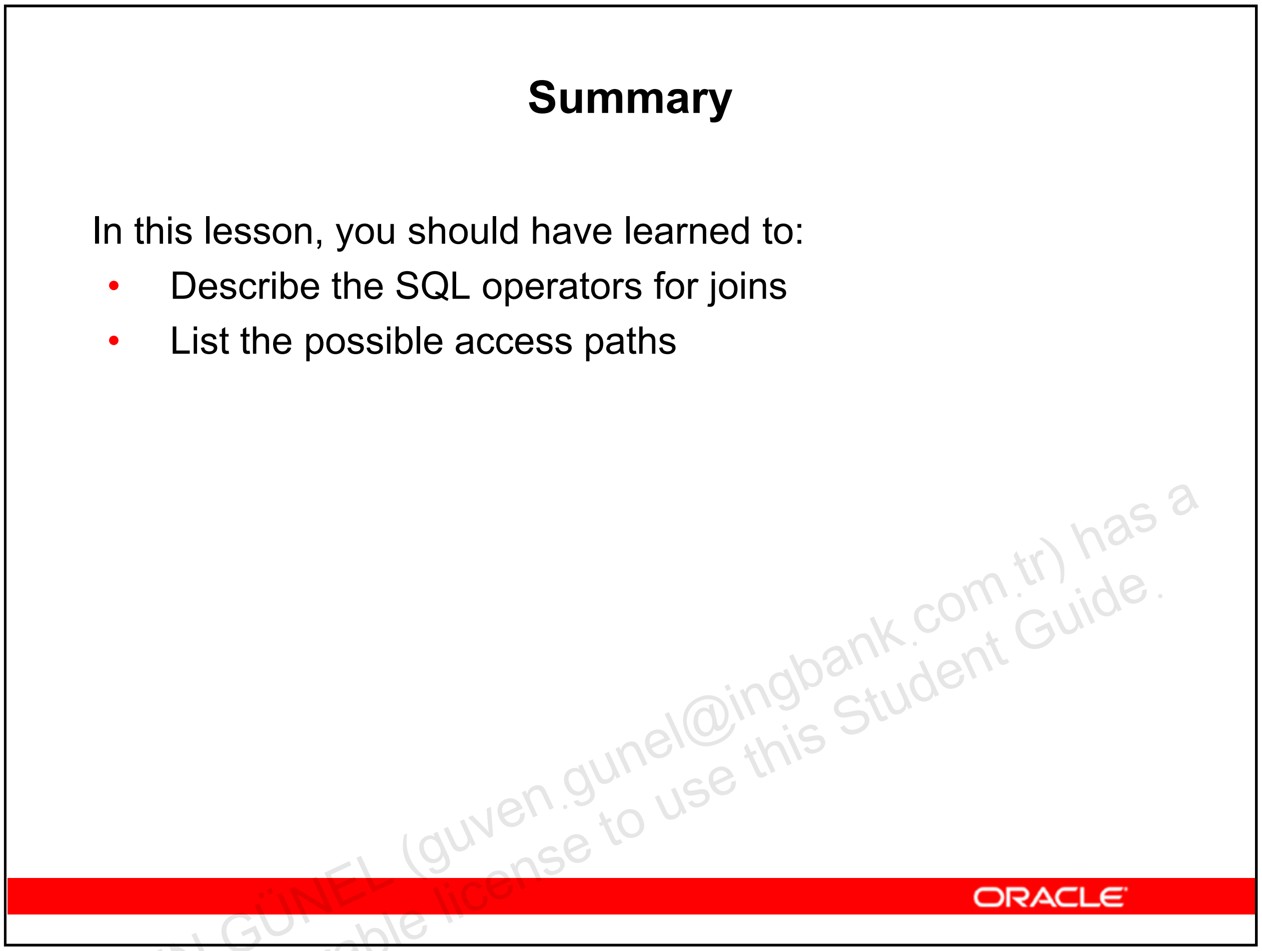


Quiz

In a hash join, the _______row source is used to build a hash table.

a. Biggest
b. Smallest
c. Sorted
d. Unsorted

Answer: b

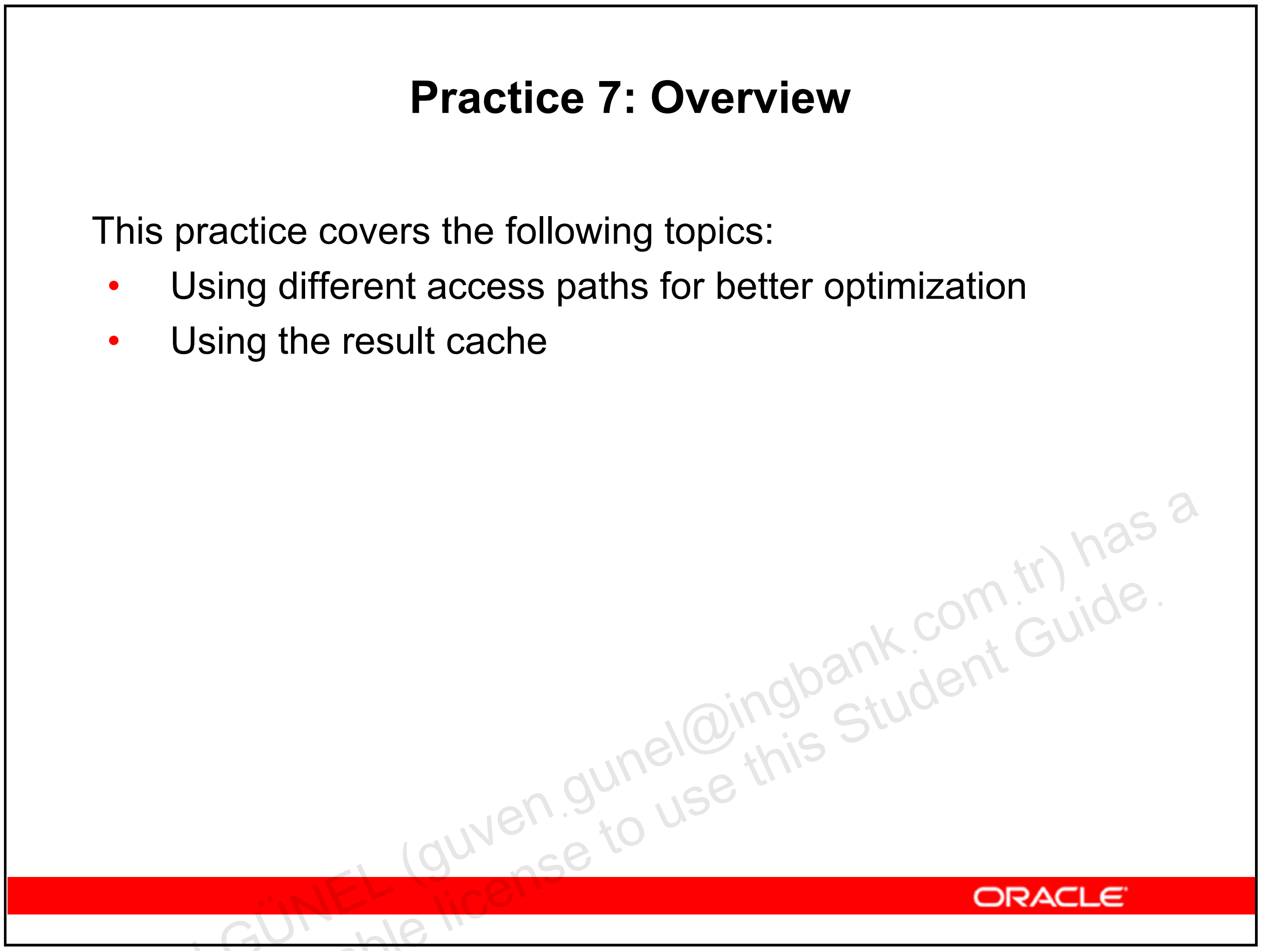


Summary

In this lesson, you should have learned to:

- Describe the SQL operators for joins
- List the possible access paths

Practice 7: Overview



Practice 7: Overview

This practice covers the following topics:

- Using different access paths for better optimization
- Using the result cache