EPAM Systems, RD Dep.

Oracle Join Methods

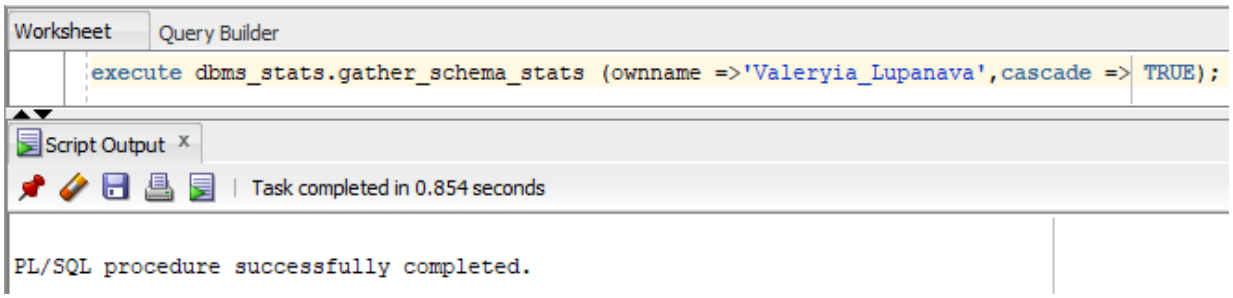
REVISION HISTORY					
Ver.	**Description of Change**	**Author**	**Date**	**Approved**	
				Name	**Effective Date**
1.0	Initial status	Valeryia_Lupanava	04-NOV-2017		

Contents

1. Join Methods

В работе я использовала таблицы из схемы HR. Я изменила скрипты выгрузки таблиц и добавила их своему пользователю. Потом собрала статистику.

EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS (OWNNAME =>'VALERYIA_LUPANAVA',CASCADE => TRUE);



1.1. Nested Loop Join

Выполнила первый SELECT. В запросе выполняется LEFT JOIN к таблице COUNTRIES только регионов с кодом 1.

```
SELECT *
FROM COUNTRIES CN, REGIONS RG
WHERE RG.REGION_ID = CN.REGION_ID(+)
AND RG.REGION_ID = 1;
```

Результат запроса.

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 9 in 0.008 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	BE	Belgium	1	1	Europe
2	BU	Belarus	1	1	Europe
3	CH	Switzerland	1	1	Europe
4	DE	Germany	1	1	Europe
5	DK	Denmark	1	1	Europe
6	FR	France	1	1	Europe
7	IT	Italy	1	1	Europe
8	NL	Netherlands	1	1	Europe
9	UK	United Kingdom	1	1	Europe

Теперь посмотрим план выполнения.

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG
WHERE RG.REGION_ID = CN.REGION_ID(+)
AND RG.REGION_ID = 1;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 17 in 0.046 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 3506115523

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     9 |   261 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS OUTER          |           |     9 |   261 |     4   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| REGIONS   |     1 |    14 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | REG_ID_PK |     1 |       |     0   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL          | COUNTRIES |     9 |   135 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   3 - access("RG"."REGION_ID"=1)
   4 - filter("CN"."REGION_ID"(+)=1)
```

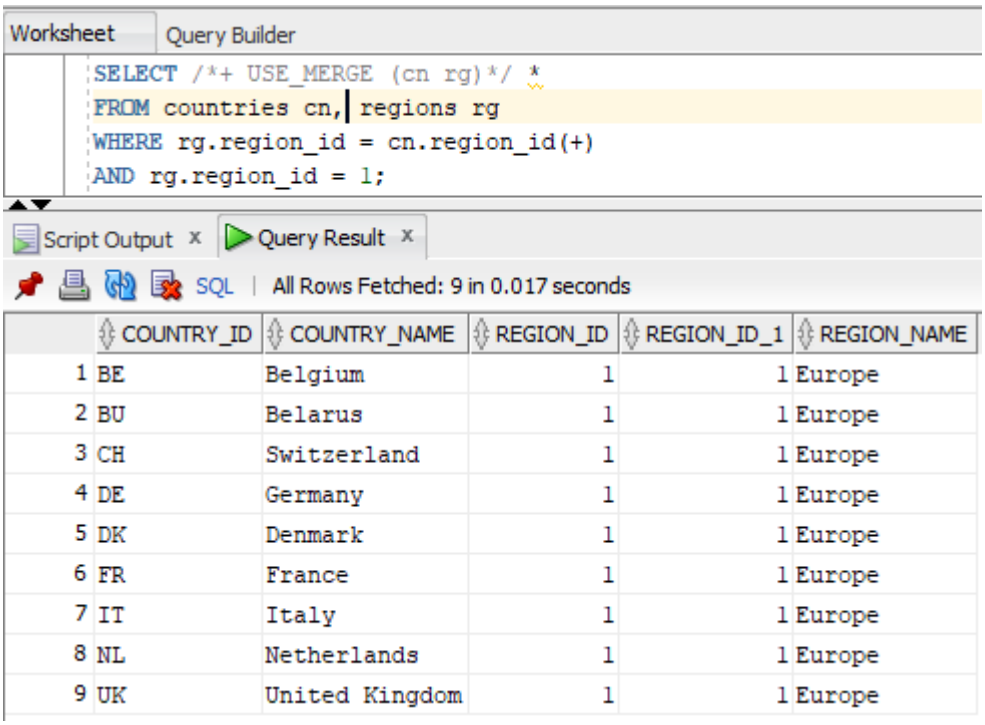
Оптимизатор предложил выполнить NESTED LOOP, поскольку наш набор данных ограничен условием REGION_ID = 1, а также на поля, используемые для соединения, базой данных был создан индекс, поскольку эти поля являются PK.

1.2. Task 3: Sort-Merge Join

В следующем SELECT я применила хинт на USE_MERGE. Запрос такой же, как в первом задании.

```
SELECT /*+ USE_MERGE (CN RG)*/ *
FROM COUNTRIES CN, REGIONS RG
WHERE RG.REGION_ID = CN.REGION_ID(+)
AND RG.REGION_ID = 1;
```

Результат запроса.



Worksheet | Query Builder

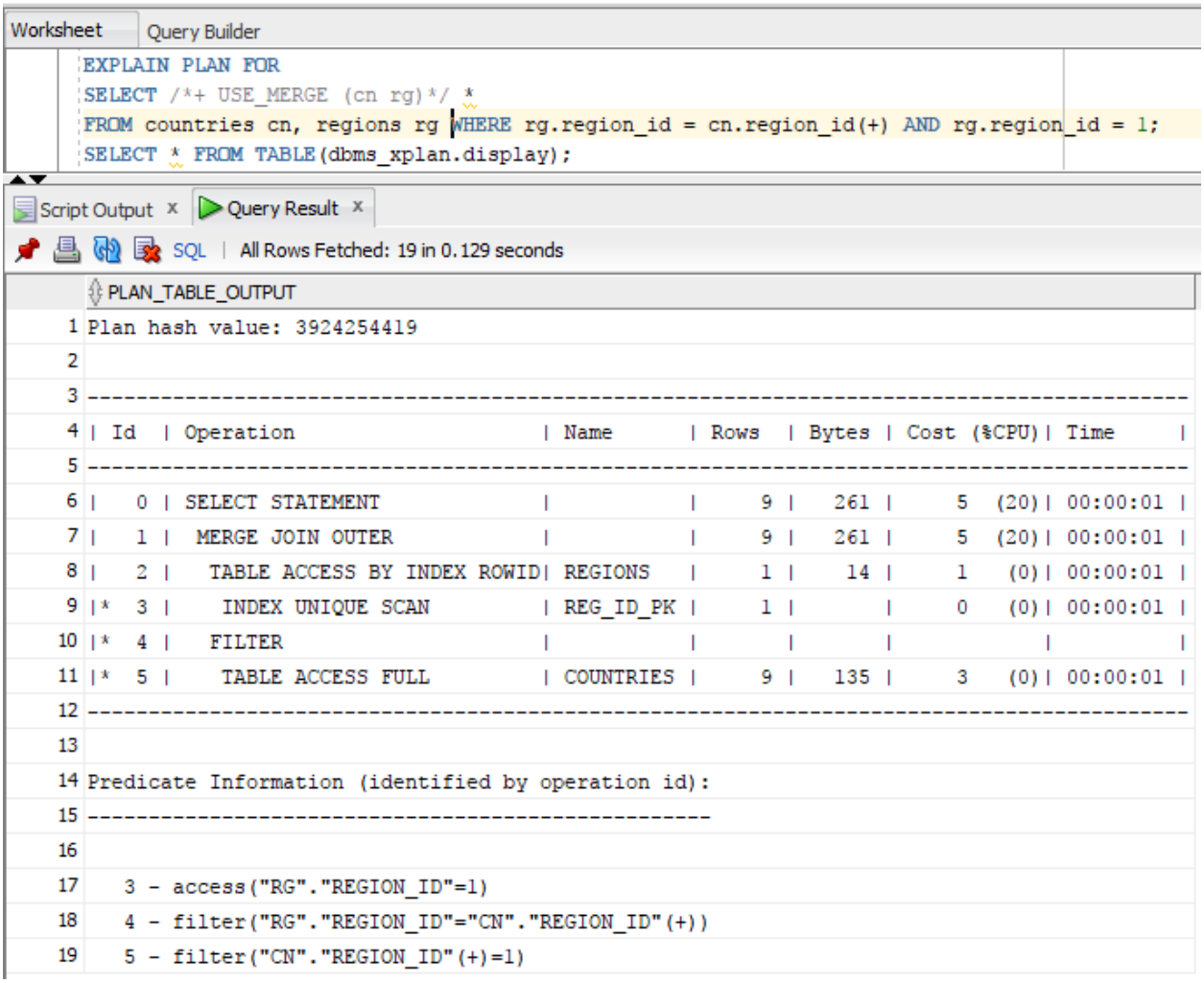
```
SELECT /*+ USE_MERGE (cn rg)*/ *
FROM countries cn, regions rg
WHERE rg.region_id = cn.region_id(+)
AND rg.region_id = 1;
```

Script Output | Query Result

SQL | All Rows Fetched: 9 in 0.017 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	BE	Belgium	1	1	Europe
2	BU	Belarus	1	1	Europe
3	CH	Switzerland	1	1	Europe
4	DE	Germany	1	1	Europe
5	DK	Denmark	1	1	Europe
6	FR	France	1	1	Europe
7	IT	Italy	1	1	Europe
8	NL	Netherlands	1	1	Europe
9	UK	United Kingdom	1	1	Europe

Я получила все те же 9 строк. Теперь посмотрим план выполнения.



Worksheet | Query Builder

```
EXPLAIN PLAN FOR
SELECT /*+ USE_MERGE (cn rg)*/ *
FROM countries cn, regions rg WHERE rg.region_id = cn.region_id(+) AND rg.region_id = 1;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Query Result | SQL | All Rows Fetched: 19 in 0.129 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 3924254419

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     9 |   261 |     5  (20)| 00:00:01 |
|   1 |  MERGE JOIN OUTER             |            |     9 |   261 |     5  (20)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| REGIONS    |     1 |    14 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | REG_ID_PK  |     1 |       |     0   (0)| 00:00:01 |
|*  4 |   FILTER                      |            |       |       |            |          |
|*  5 |    TABLE ACCESS FULL          | COUNTRIES  |     9 |   135 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   3 - access("RG"."REGION_ID"=1)
   4 - filter("RG"."REGION_ID"="CN"."REGION_ID"(+))
   5 - filter("CN"."REGION_ID"(+)=1)
```

Как видно, оптимизатор действительно применил MERGE, поскольку в хинте он был явно указан. Можно увидеть, что в данном случае оптимизатор сначала отсортировал и отфильтровал наименьшую таблицу по условию REGION_ID = 1, затем отсортировал наибольшую таблицу, потом выполнил LEFT JOIN. Поскольку здесь должна неявно присутствовать операция сортировки, то COST у MERGE должен быть больше. В данном случае соединение проходит по проиндексированным полям, поэтому COST незначительно выше.

1.3. Task 4: Hash Join

В следующем SELECT я применила хинт на USE_HASH. Запрос такой же, как в первом задании.

```
SELECT /*+ USE_HASH (CN RG)*/ *
FROM COUNTRIES CN, REGIONS RG
WHERE RG.REGION_ID = CN.REGION_ID(+) AND RG.REGION_ID = 1;
```

Результат запроса.

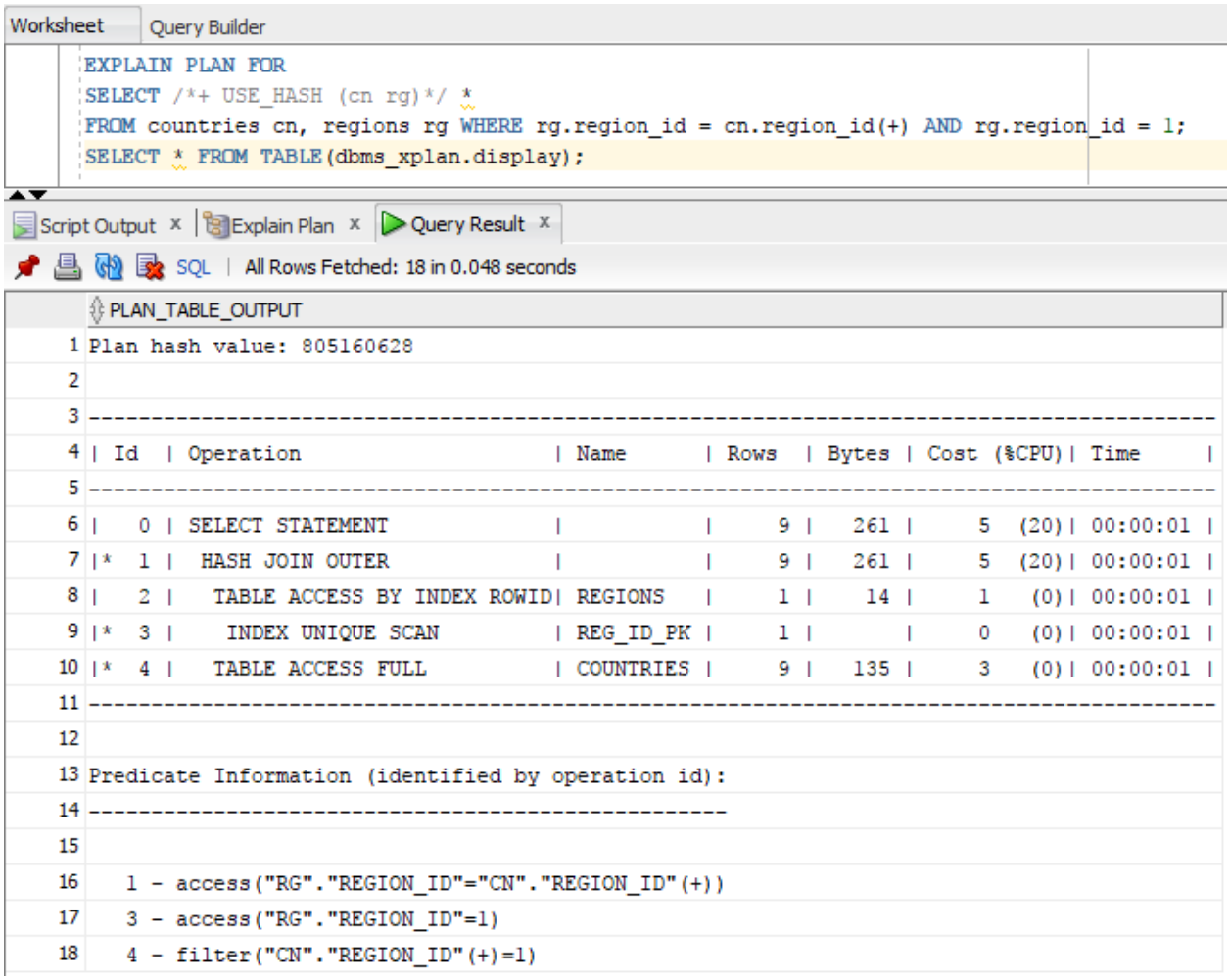
```
SELECT /*+ USE_HASH (cn rg)*/ *
FROM countries cn, regions rg WHERE rg.region_id = cn.region_id(+) AND rg.region_id = 1;
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 9 in 0.01 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	BE	Belgium	1	1	Europe
2	BU	Belarus	1	1	Europe
3	CH	Switzerland	1	1	Europe
4	DE	Germany	1	1	Europe
5	DK	Denmark	1	1	Europe
6	FR	France	1	1	Europe
7	IT	Italy	1	1	Europe
8	NL	Netherlands	1	1	Europe
9	UK	United Kingdom	1	1	Europe

Получили девять строк, все правильно. Проанализируем теперь план выполнения.



```
Worksheet  Query Builder

EXPLAIN PLAN FOR
SELECT /*+ USE_HASH (cn rg)*/ *
FROM countries cn, regions rg WHERE rg.region_id = cn.region_id(+) AND rg.region_id = 1;
SELECT * FROM TABLE(dbms_xplan.display);

Script Output x | Explain Plan x | Query Result x
SQL | All Rows Fetched: 18 in 0.048 seconds

PLAN_TABLE_OUTPUT
1  Plan hash value: 805160628
2
3  ----------------------------------------------------------------------------------------
4  | Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
5  ----------------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT              |           |     9 |   261 |     5  (20)| 00:00:01 |
7  |*  1 |  HASH JOIN OUTER              |           |     9 |   261 |     5  (20)| 00:00:01 |
8  |   2 |   TABLE ACCESS BY INDEX ROWID| REGIONS   |     1 |    14 |     1   (0)| 00:00:01 |
9  |*  3 |    INDEX UNIQUE SCAN          | REG_ID_PK |     1 |       |     0   (0)| 00:00:01 |
10 |*  4 |   TABLE ACCESS FULL           | COUNTRIES |     9 |   135 |     3   (0)| 00:00:01 |
11 ----------------------------------------------------------------------------------------
12
13 Predicate Information (identified by operation id):
14 ---------------------------------------------------
15
16    1 - access("RG"."REGION_ID"="CN"."REGION_ID"(+))
17    3 - access("RG"."REGION_ID"=1)
18    4 - filter("CN"."REGION_ID"(+)=1)
```

Можно увидеть, что оптимизатор выполнил HASH JOIN. То есть сначала он выбрал наименьшую таблицу, захэшировал значение ключа и поместил во в буферную таблицу. Затем он последовательно хэшировал значение ключа второй таблицы, не создавая буферную, сравнивал по условию хэш-ключи двух таблиц и фильтровал в зависимости от того, соблюдается равенство или нет. Также в таблице плана выполнения можно увидеть, что HASH JOIN загрузил CPU на 20%, поскольку только одна из таблиц была сохранена в памяти. Стоимость вышла такая же, как у MERGE.

Поскольку у нас две небольшие таблицы, к тому же на поля соединения навешены индексы, то самым оптимальным будет NESTED LOOP, что и предложил в первый раз оптимизатор БД.

1.4. Task 5: Cartesian Join

- *Сначала я применила LEADING hint и правильный порядок.*

EXPLAIN PLAN FOR

SELECT /*+ LEADING(CN, RG)*/ *

FROM COUNTRIES CN, REGIONS RG;

SELECT * FROM TABLE(dbms_xplan.display);

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(CN, RG)*/ *
FROM COUNTRIES CN, REGIONS RG;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 11 in 0.05 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 4200168870

-------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |   104 |  3016 |    40   (0)| 00:00:01 |
|   1 |  MERGE JOIN CARTESIAN|          |   104 |  3016 |    40   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
|   3 |   BUFFER SORT       |           |     4 |    56 |    37   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| REGIONS   |     4 |    56 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

- *Потом я применила LEADING hint и обратный порядок.*

EXPLAIN PLAN FOR

SELECT /*+ LEADING(RG, CN)*/ *

FROM COUNTRIES CN, REGIONS RG;

SELECT * FROM TABLE(dbms_xplan.display);

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(RG, CN)*/ *
FROM COUNTRIES CN, REGIONS RG;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 11 in 0.051 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 578368426

-------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   104 |  3016 |    10   (0)| 00:00:01 |
|   1 |  MERGE JOIN CARTESIAN|           |   104 |  3016 |    10   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL  | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
|   3 |   BUFFER SORT        |           |    26 |   390 |     7   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL | COUNTRIES |    26 |   390 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

- *Потом я применила ORDERED hint и правильный порядок.*

EXPLAIN PLAN FOR

SELECT /*+ ORDERED(CN, RG)*/ *

FROM COUNTRIES CN, REGIONS RG;

SELECT * FROM TABLE(dbms_xplan.display);

```
EXPLAIN PLAN FOR
SELECT /*+ ORDERED(CN, RG)*/ *
FROM COUNTRIES CN, REGIONS RG;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 11 in 0.1 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 578368426

-------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |   104 |  3016 |    10   (0)| 00:00:01 |
|   1 |  MERGE JOIN CARTESIAN|           |   104 |  3016 |    10   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL  | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
|   3 |   BUFFER SORT        |           |    26 |   390 |     7   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL | COUNTRIES |    26 |   390 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

- *Последним шагом я применила ORDERED hint и обратный порядок.*

EXPLAIN PLAN FOR

SELECT /*+ ORDERED(RG, CN)*/ *

FROM COUNTRIES CN, REGIONS RG;

SELECT * FROM TABLE(dbms_xplan.display);

```
EXPLAIN PLAN FOR
SELECT /*+ ORDERED(RG, CN)*/ *
FROM COUNTRIES CN, REGIONS RG;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 11 in 0.1 seconds

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 578368426
2
3  ------------------------------------------------------------------------------
4  | Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT     |           |   104 |  3016 |    10   (0)| 00:00:01 |
7  |   1 |  MERGE JOIN CARTESIAN|           |   104 |  3016 |    10   (0)| 00:00:01 |
8  |   2 |   TABLE ACCESS FULL  | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
9  |   3 |   BUFFER SORT        |           |    26 |   390 |     7   (0)| 00:00:01 |
10 |   4 |    TABLE ACCESS FULL | COUNTRIES |    26 |   390 |     2   (0)| 00:00:01 |
11 ------------------------------------------------------------------------------
```

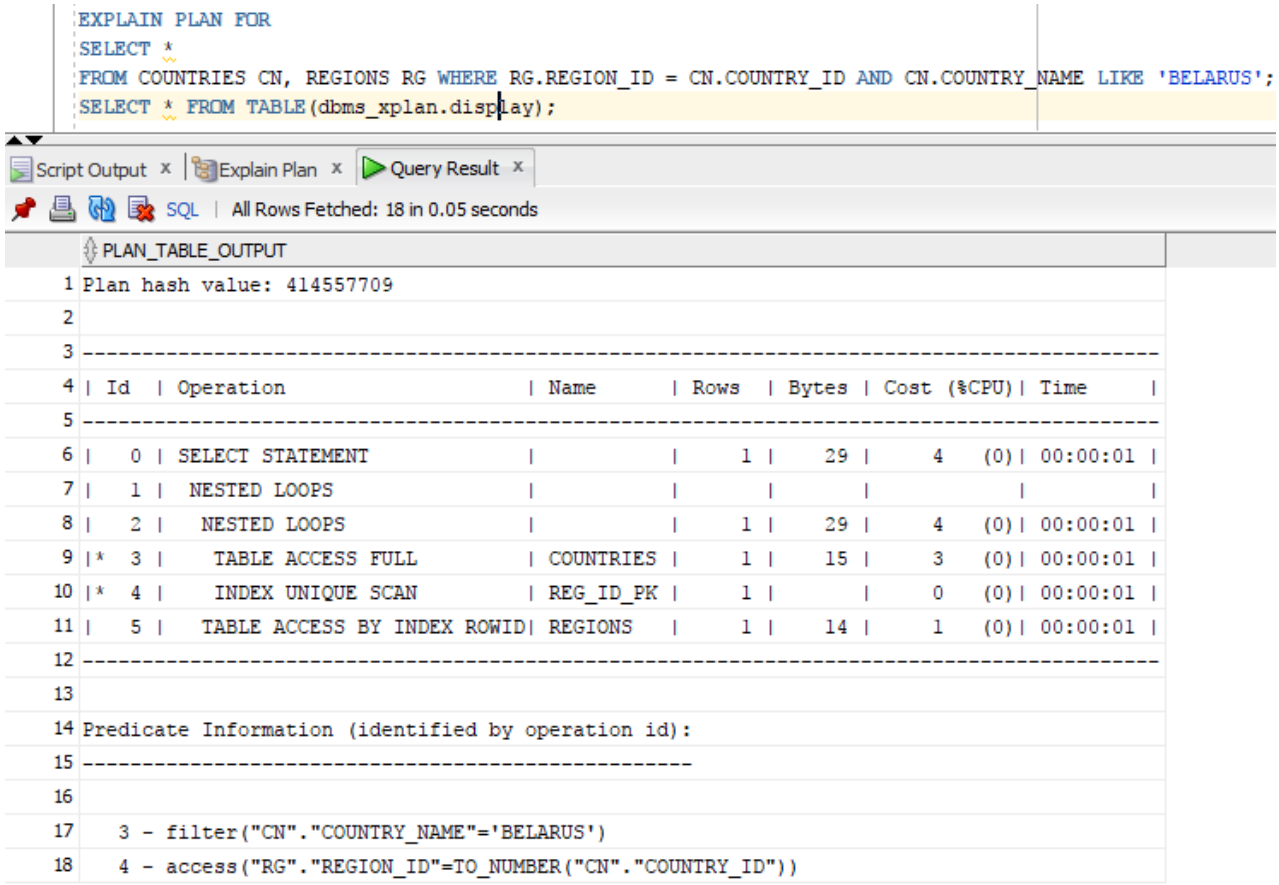
Вывод: можно убедиться, что используя хинт LEADING, мы может управлять порядком JOIN. Какой порядок в LEADING будет указан, такой оптимизатор и примет в CROSS JOIN. При использовании хинта ORDERED мы ни каким образом не можем повлиять на порядок JOIN. Оптимизатор выберет порядок, при котором на первом месте будет наименьшая таблица, что и видно из CROSS JOIN с ORDERED. CROSS JOIN выбирает наименьшую таблицу, т.к. он использует метод MERGE JOIN, а этот метод в свою очередь выполняет операции сортировки и первым делом берет наименьший сет с данными (было описано во втором задании).

*Еще одно важное замечание, в LEADING с порядком **большая -> маленькая таблица** можно наблюдать самый большой COST. Это опять же показывает, как сортировка утяжеляет запрос и как важен порядок таблиц при обработке.*

- *С условием.*

  Первый вариант.

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID(+)
AND CN.COUNTRY_NAME LIKE 'BELARUS';
SELECT * FROM TABLE(dbms_xplan.display);
```



```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID AND CN.COUNTRY_NAME LIKE 'BELARUS';
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 18 in 0.05 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 414557709

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     1 |    29 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |           |       |       |            |          |
|   2 |   NESTED LOOPS               |           |     1 |    29 |     4   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL         | COUNTRIES |     1 |    15 |     3   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN         | REG_ID_PK |     1 |       |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| REGIONS   |     1 |    14 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   3 - filter("CN"."COUNTRY_NAME"='BELARUS')
   4 - access("RG"."REGION_ID"=TO_NUMBER("CN"."COUNTRY_ID"))
```

Второй вариант.

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID(+);
SELECT * FROM TABLE(dbms_xplan.display);
```

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID(+);
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 18 in 0.053 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 162365680

-----------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     4 |   116 |     6  (17)| 00:00:01 |
|   1 |  MERGE JOIN OUTER             |           |     4 |   116 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| REGIONS   |     4 |    56 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN            | REG_ID_PK |     4 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                   |           |    26 |   390 |     4  (25)| 00:00:01 |
|   5 |    TABLE ACCESS FULL          | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

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

   4 - access("RG"."REGION_ID"=TO_NUMBER("CN"."COUNTRY_ID"(+)))
       filter("RG"."REGION_ID"=TO_NUMBER("CN"."COUNTRY_ID"(+)))
```

- *Результат без условия.*

EXPLAIN PLAN FOR

SELECT *

FROM COUNTRIES CN, REGIONS RG;

SELECT * FROM TABLE(dbms_xplan.display);

```
EXPLAIN PLAN FOR
SELECT *
FROM countries cn, regions rg;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 11 in 0.055 seconds

```
PLAN_TABLE_OUTPUT
1  Plan hash value: 578368426
2
3  ------------------------------------------------------------------------------
4  | Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
5  ------------------------------------------------------------------------------
6  |   0 | SELECT STATEMENT     |           |   104 |  3016 |    10   (0)| 00:00:01 |
7  |   1 |  MERGE JOIN CARTESIAN|           |   104 |  3016 |    10   (0)| 00:00:01 |
8  |   2 |   TABLE ACCESS FULL  | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
9  |   3 |   BUFFER SORT        |           |    26 |   390 |     7   (0)| 00:00:01 |
10 |   4 |    TABLE ACCESS FULL | COUNTRIES |    26 |   390 |     2   (0)| 00:00:01 |
11 ------------------------------------------------------------------------------
```

Вывод: как видно из результатов, любое условие превращает CARTESIAN JOIN в другой метод JOIN, поскольку при добавлении WHERE чистого декартового произведения уже не будет.

1.5. Task 6: Left/Right Outer Joins

Implement Left/Right outer joins with:

- *ANSI синтаксис. LEFT JOIN.*

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN LEFT JOIN REGIONS RG ON RG.REGION_ID = CN.COUNTRY_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

```
SELECT *
FROM countries cn LEFT JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 26 in 0.015 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	UK	United Kingdom	1	1	Europe
2	NL	Netherlands	1	1	Europe
3	IT	Italy	1	1	Europe
4	FR	France	1	1	Europe
5	DK	Denmark	1	1	Europe
6	DE	Germany	1	1	Europe
7	CH	Switzerland	1	1	Europe
8	BU	Belarus	1	1	Europe
9	BE	Belgium	1	1	Europe
10	US	United States of America	2	2	Americas
11	MX	Mexico	2	2	Americas
12	CA	Canada	2	2	Americas
13	BR	Brazil	2	2	Americas
14	AR	Argentina	2	2	Americas
15	SG	Singapore	3	3	Asia
16	ML	Malaysia	3	3	Asia
17	JP	Japan	3	3	Asia
18	IN	India	3	3	Asia
19	CN	China	3	3	Asia
20	AU	Australia	3	3	Asia
21	ZW	Zimbabwe	4	4	Middle East and Africa

```
EXPLAIN PLAN FOR
SELECT *
FROM countries cn LEFT JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 15 in 0.047 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 1985576147

-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |    26 |   754 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN OUTER   |           |    26 |   754 |     7  (15)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("RG"."REGION_ID"(+)="CN"."REGION_ID")
```

- *Oracle синтаксис. LEFT JOIN.*

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID(+);
SELECT * FROM TABLE(dbms_xplan.display);
```

```
SELECT *
FROM countries cn, regions rg WHERE rg.REGION_ID = cn.REGION_ID(+);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 26 in 0.01 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	FR	France	1	1	Europe
2	UK	United Kingdom	1	1	Europe
3	DK	Denmark	1	1	Europe
4	BE	Belgium	1	1	Europe
5	DE	Germany	1	1	Europe
6	BU	Belarus	1	1	Europe
7	CH	Switzerland	1	1	Europe
8	IT	Italy	1	1	Europe
9	NL	Netherlands	1	1	Europe
10	AR	Argentina	2	2	Americas
11	CA	Canada	2	2	Americas
12	BR	Brazil	2	2	Americas

```
SELECT *
FROM countries cn, regions rg WHERE rg.REGION_ID = cn.REGION_ID(+);
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 18 in 0.064 seconds

	PLAN_TABLE_OUTPUT
1	Plan hash value: 162365680
2	
3	--
4	| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
5	--
6	| 0 | SELECT STATEMENT | | 26 | 754 | 6 (17)| 00:00:01 |
7	| 1 | MERGE JOIN OUTER | | 26 | 754 | 6 (17)| 00:00:01 |
8	| 2 | TABLE ACCESS BY INDEX ROWID| REGIONS | 4 | 56 | 2 (0)| 00:00:01 |
9	| 3 | INDEX FULL SCAN | REG_ID_PK | 4 | | 1 (0)| 00:00:01 |
10	|* 4 | SORT JOIN | | 26 | 390 | 4 (25)| 00:00:01 |
11	| 5 | TABLE ACCESS FULL | COUNTRIES | 26 | 390 | 3 (0)| 00:00:01 |
12	--
13	
14	Predicate Information (identified by operation id):
15	---
16	
17	4 - access("RG"."REGION_ID"="CN"."REGION_ID"(+))
18	filter("RG"."REGION_ID"="CN"."REGION_ID"(+))

- *ANSI синтаксис. RIGHT JOIN.*

EXPLAIN PLAN FOR

SELECT *

FROM COUNTRIES CN RIGHT JOIN REGIONS RG ON RG.REGION_ID = CN.COUNTRY_ID;

SELECT * FROM TABLE(dbms_xplan.display);

```
SELECT *
FROM countries cn RIGHT JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 26 in 0.01 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
6	BU	Belarus	1	1	Europe
7	CH	Switzerland	1	1	Europe
8	IT	Italy	1	1	Europe
9	NL	Netherlands	1	1	Europe
10	AR	Argentina	2	2	Americas
11	CA	Canada	2	2	Americas
12	BR	Brazil	2	2	Americas
13	US	United States of America	2	2	Americas
14	MX	Mexico	2	2	Americas
15	CN	China	3	3	Asia
16	JP	Japan	3	3	Asia
17	SG	Singapore	3	3	Asia
18	ML	Malaysia	3	3	Asia
19	IN	India	3	3	Asia
20	AU	Australia	3	3	Asia
21	ZM	Zambia	4	4	Middle East and Africa
22	NG	Nigeria	4	4	Middle East and Africa
23	KW	Kuwait	4	4	Middle East and Africa
24	ZW	Zimbabwe	4	4	Middle East and Africa
25	EG	Egypt	4	4	Middle East and Africa
26	IL	Israel	4	4	Middle East and Africa

```
SELECT *
FROM countries cn RIGHT JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 18 in 0.055 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 162365680

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |    26 |   754 |     6  (17)| 00:00:01 |
|   1 |  MERGE JOIN OUTER             |           |    26 |   754 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| REGIONS   |     4 |    56 |     2   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN            | REG_ID_PK |     4 |       |     1   (0)| 00:00:01 |
|*  4 |   SORT JOIN                   |           |    26 |   390 |     4  (25)| 00:00:01 |
|   5 |    TABLE ACCESS FULL          | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - access("RG"."REGION_ID"="CN"."REGION_ID"(+))
       filter("RG"."REGION_ID"="CN"."REGION_ID"(+))
```

- *Oracle синтаксис. RIGHT JOIN.*

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

```
SELECT *
FROM countries cn, regions rg WHERE rg.REGION_ID(+) = cn.REGION_ID;
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 26 in 0.018 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	UK	United Kingdom	1	1	Europe
2	NL	Netherlands	1	1	Europe
3	IT	Italy	1	1	Europe
4	FR	France	1	1	Europe
5	DK	Denmark	1	1	Europe
6	DE	Germany	1	1	Europe
7	CH	Switzerland	1	1	Europe
8	BU	Belarus	1	1	Europe
9	BE	Belgium	1	1	Europe
10	US	United States of America	2	2	Americas
11	MX	Mexico	2	2	Americas
12	CA	Canada	2	2	Americas
13	BR	Brazil	2	2	Americas
14	AR	Argentina	2	2	Americas
15	SG	Singapore	3	3	Asia
16	ML	Malaysia	3	3	Asia

```
FROM countries cn, regions rg WHERE rg.REGION_ID(+) = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 15 in 0.056 seconds

	PLAN_TABLE_OUTPUT
1	Plan hash value: 1985576147
2	
3	--
4	| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
5	--
6	| 0 | SELECT STATEMENT | | 26 | 754 | 7 (15)| 00:00:01 |
7	|* 1 | HASH JOIN OUTER | | 26 | 754 | 7 (15)| 00:00:01 |
8	| 2 | TABLE ACCESS FULL| COUNTRIES | 26 | 390 | 3 (0)| 00:00:01 |
9	| 3 | TABLE ACCESS FULL| REGIONS | 4 | 56 | 3 (0)| 00:00:01 |
10	--
11	
12	Predicate Information (identified by operation id):
13	---
14	
15	1 - access("RG"."REGION_ID"(+)="CN"."REGION_ID")

Вывод: в обоих случаях: LEFT ORACLE SYNTAX и LEFT ANSI SYNTAX OUTPUT одинаковые. Однако, LEFT ORACLE SYNTAX имеет меньшую стоимость и более быстрое время выполнения нежели LEFT ANSI SYNTAX. А вот для RIGHT результат обратный, не смотря на то, что ведущей выступает меньшая таблица.

1.6. Task 7: Full Outer Join

- *ANSI синтаксис. FULL JOIN.*

EXPLAIN PLAN FOR

SELECT *

FROM COUNTRIES FULL OUTER JOIN REGIONS RG ON RG.REGION_ID = CN.COUNTRY_ID;

SELECT * FROM TABLE(dbms_xplan.display);

```
SELECT *
FROM countries cn FULL OUTER JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query Result

SQL | All Rows Fetched: 26 in 0.016 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	AR	Argentina	2	2	Americas
2	AU	Australia	3	3	Asia
3	BE	Belgium	1	1	Europe
4	BR	Brazil	2	2	Americas
5	BU	Belarus	1	1	Europe
6	CA	Canada	2	2	Americas
7	CH	Switzerland	1	1	Europe
8	CN	China	3	3	Asia
9	DE	Germany	1	1	Europe
10	DK	Denmark	1	1	Europe
11	EG	Egypt	4	4	Middle East and Africa
12	FR	France	1	1	Europe
13	IL	Israel	4	4	Middle East and Africa
14	IN	India	3	3	Asia
15	IT	Italy	1	1	Europe
16	JP	Japan	3	3	Asia
17	KW	Kuwait	4	4	Middle East and Africa
18	ML	Malaysia	3	3	Asia
19	MX	Mexico	2	2	Americas
20	NG	Nigeria	4	4	Middle East and Africa
21	NL	Netherlands	1	1	Europe

```
SELECT *
FROM countries cn FULL OUTER JOIN regions rg ON rg.REGION_ID = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output | Explain Plan | Query...

SQL | All Rows Fetched: 16 in 0.061 seconds

```
PLAN_TABLE_OUTPUT
Plan hash value: 53640290

-------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |    26 |  1716 |     7  (15)| 00:00:01 |
|   1 |  VIEW                 | VW_FOJ_0  |    26 |  1716 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN FULL OUTER|           |    26 |   754 |     7  (15)| 00:00:01 |
|   3 |    TABLE ACCESS FULL  | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL  | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - access("RG"."REGION_ID"="CN"."REGION_ID")
```

- *Oracle синтаксис. FULL JOIN.*

```
EXPLAIN PLAN FOR
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID = CN.COUNTRY_ID(+)
UNION
SELECT *
FROM COUNTRIES CN, REGIONS RG WHERE RG.REGION_ID(+)= CN.COUNTRY_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

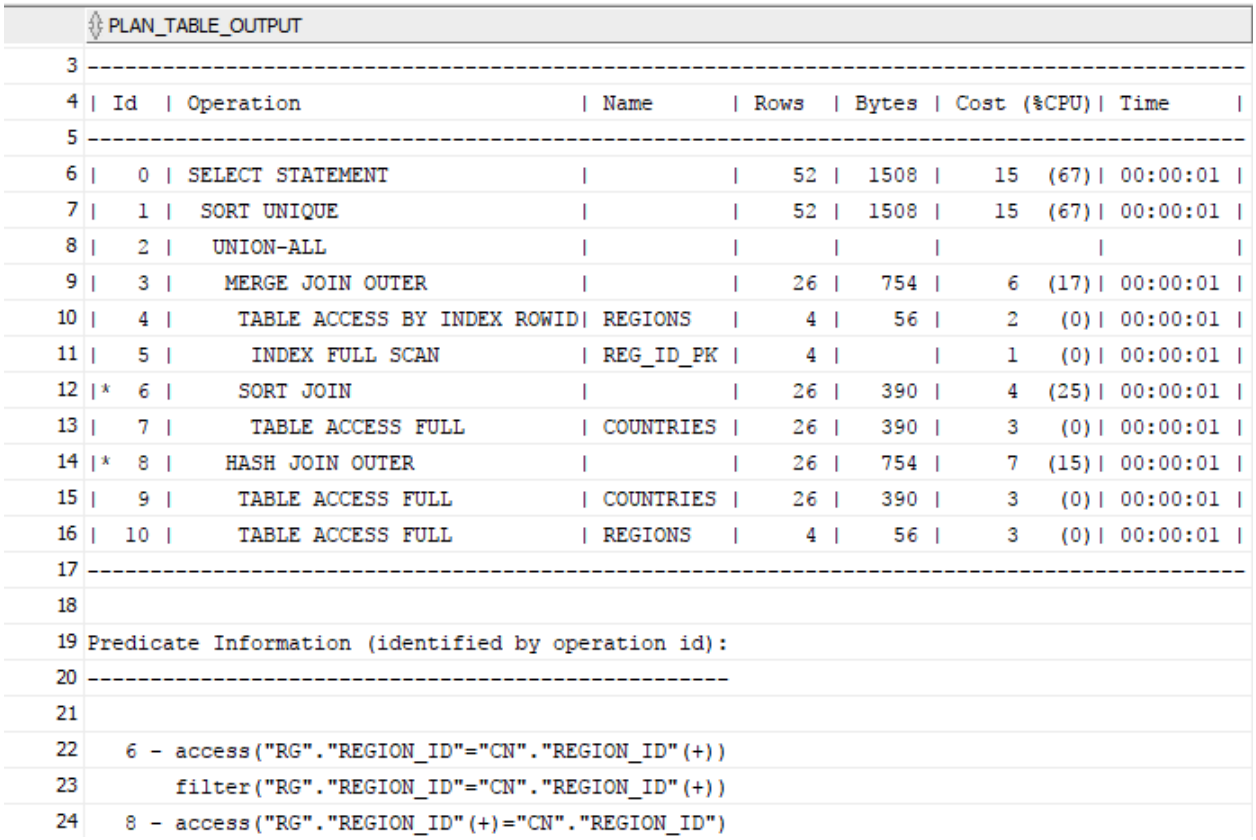
Worksheet | Query Builder

```
SELECT *
FROM countries cn, regions rg WHERE rg.REGION_ID = cn.REGION_ID(+)
UNION
SELECT *
FROM countries cn, regions rg WHERE rg.REGION_ID(+) = cn.REGION_ID;
SELECT * FROM TABLE(dbms_xplan.display);
```

Script Output x | Explain Plan x | Query Result x

SQL | All Rows Fetched: 26 in 0.237 seconds

	COUNTRY_ID	COUNTRY_NAME	REGION_ID	REGION_ID_1	REGION_NAME
1	AR	Argentina	2	2	Americas
2	AU	Australia	3	3	Asia
3	BE	Belgium	1	1	Europe
4	BR	Brazil	2	2	Americas
5	BU	Belarus	1	1	Europe
6	CA	Canada	2	2	Americas
7	CH	Switzerland	1	1	Europe
8	CN	China	3	3	Asia
9	DE	Germany	1	1	Europe
10	DK	Denmark	1	1	Europe
11	EG	Egypt	4	4	Middle East and Africa
12	FR	France	1	1	Europe
13	IL	Israel	4	4	Middle East and Africa
14	IN	India	3	3	Asia
15	IT	Italy	1	1	Europe
16	JP	Japan	3	3	Asia
17	KW	Kuwait	4	4	Middle East and Africa
18	ML	Malaysia	3	3	Asia



```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |           |    52 |  1508 |    15  (67)| 00:00:01 |
|   1 |  SORT UNIQUE                   |           |    52 |  1508 |    15  (67)| 00:00:01 |
|   2 |   UNION-ALL                    |           |       |       |            |          |
|   3 |    MERGE JOIN OUTER            |           |    26 |   754 |     6  (17)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID| REGIONS   |     4 |    56 |     2   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN           | REG_ID_PK |     4 |       |     1   (0)| 00:00:01 |
|*  6 |     SORT JOIN                  |           |    26 |   390 |     4  (25)| 00:00:01 |
|   7 |      TABLE ACCESS FULL         | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
|*  8 |    HASH JOIN OUTER             |           |    26 |   754 |     7  (15)| 00:00:01 |
|   9 |     TABLE ACCESS FULL          | COUNTRIES |    26 |   390 |     3   (0)| 00:00:01 |
|  10 |     TABLE ACCESS FULL          | REGIONS   |     4 |    56 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   6 - access("RG"."REGION_ID"="CN"."REGION_ID"(+))
       filter("RG"."REGION_ID"="CN"."REGION_ID"(+))
   8 - access("RG"."REGION_ID"(+)="CN"."REGION_ID")
```

Вывод: как видно из плана выполнения, ANSI значительно меньше по стоимости нежели ORACLE синтаксис. Такой результат объясняется тем, что в синтаксисе ORACLE необходимо применить UNION, поэтому в итоге получается два запроса. В синтаксисе ANSI оптимизатор просто хэширует меньшую таблицу и потом реализует запрос. В случае ORACLE синтаксиса оптимизатор проводит два метода JOIN: MERGE и HASH.

1.7. Task 8: Results

- *Примеры комбинаций (схема HR):*

Table “A”	Table “B”	Script description	Join type description
JOBS Маленькая таблица с индексом по job_id	EMPLOYEES Большая таблица	SELECT * FROM employees emp, jobs jb WHERE emp.job_ID = jb.job_ID;	HASH JOIN
JOBS Маленькая таблица с индексом по job_id	EMPLOYEES Большая таблица	SELECT * FROM jobs jb, employees emp WHERE jb.job_ID = emp.job_ID(+);	HASH JOIN
EMPLOYEES Большая таблица	DEPARTMENTS Небольшая таблица с индексом по department_id	SELECT * FROM employees emp, DEPARTMENTS dep WHERE emp.DEPARTMENT_ID = dep.DEPARTMENT_ID AND emp.employee_id IN (100,105);	NESTED LOOP JOIN VS. UNIQUE SCAN
EMPLOYEES Большая таблица	DEPARTMENTS Небольшая таблица с индексом по department_id	SELECT dep.MANAGER_ID FROM employees emp, DEPARTMENTS dep WHERE emp.DEPARTMENT_ID(+) = dep.DEPARTMENT_ID AND dep.DEPARTMENT_NAME LIKE 'BI%';	NESTED LOOP JOIN VS. RANGE SCAN

EMPLOYEES Большая таблица	DEPARTMENTS Небольшая таблица с индексом по department_id	SELECT dep.MANAGER_ID FROM employees emp, DEPARTMENTS dep WHERE emp.DEPARTMENT_ID = dep.DEPARTMENT_ID(+);	HASH JOIN
EMPLOYEES Большая таблица с составным индексом по department_id и manager_id	DEPARTMENTS Небольшая таблица с индексом по department_id	SELECT count(*) FROM employees emp LEFT JOIN DEPARTMENTS dep ON emp.DEPARTMENT_ID = dep.DEPARTMENT_ID WHERE emp.MANAGER_ID = 145;	INDEX RANGE SCAN
LOCATIONS Небольшая таблица	COUNTRIES Небольшая таблица с индексом по country_id	SELECT DISTINCT loc.CITY FROM LOCATIONS loc, countries con WHERE loc.COUNTRY_ID = con.COUNTRY_ID AND con.COUNTRY_NAME LIKE 'Belarus';	HASH VS. NESTED LOOP JOIN
LOCATIONS Небольшая таблица с составным индексом по country_id и city	COUNTRIES Небольшая таблица с индексом по country_id	SELECT DISTINCT loc.CITY FROM LOCATIONS loc FULL OUTER JOIN countries con ON loc.COUNTRY_ID = con.COUNTRY_ID;	HASH JOIN VS. FAST FULL SCAN
LOCATIONS Небольшая таблица с индексом по location_id	DEPARTMENTS Небольшая таблица	SELECT distinct loc.STATE_PROVINCE FROM LOCATIONS loc, DEPARTMENTS dep WHERE	NESTED LOOP JOIN VS. RANGE SCAN

		dep.LOCATION_ID(+) = loc.LOCATION_ID;	
LOCATIONS Небольшая таблица с индексом по location_id	DEPARTMENTS Небольшая таблица	SELECT dep.DEPARTMENT_ID, loc.STREET_ADDRESS FROM LOCATIONS loc, DEPARTMENTS dep WHERE dep.LOCATION_ID(+) = loc.LOCATION_ID;	HASH JOIN VS. FAST FULL SCAN
LOCATIONS Небольшая таблица с составным индексом по country_id, city, street_address	DEPARTMENTS Небольшая таблица	SELECT * FROM LOCATIONS loc INNER JOIN DEPARTMENTS dep ON dep.LOCATION_ID = loc.LOCATION_ID WHERE loc.STREET_ADDRESS LIKE 'Kuprevicha%';	NESTED LOOP JOIN VS. SKIP SCAN VS. RANGE SCAN