Real-Time SQL Interview Questions for Data Professionals: Data Analyst, Data Scientist, Database Developer & More (Applicable to Any Company)

Question 1:

List all possible matches between teams if each team plays against every other team exactly once. Ensure that reverse duplicates (e.g., "India vs Pak" and "Pak vs India") are not included in the output.

Input Table: `match`

team
India
Pak
Aus
Eng

Expected Output:

team_A	team_B
Aus	Eng
Aus	India
Aus	Pak
Eng	India
Eng	Pak
India	Pak

Step-by-Step Solution:

Step 1: Create the Input Table

```
CREATE TABLE match (
   team VARCHAR2(20) );
```

Step 2: Insert Data into the Table

```
INSERT INTO match (team) VALUES ('India');
INSERT INTO match (team) VALUES ('Pak');
INSERT INTO match (team) VALUES ('Aus');
INSERT INTO match (team) VALUES ('Eng');
COMMIT;
```

Query :

Solution Approach 1: Using CROSS JOIN with Filtering

Step 3: Generate All Possible Matches

```sql
--select * from match;

SELECT
     t1.team AS team_A,
     t2.team AS team_B
FROM
     match t1
CROSS JOIN
     match t2
WHERE
     t1.team < t2.team
     order by t1.team,t2.team  asc;
```

Script Output | Query Result

SQL | All Rows Fetched: 6 in 0.005 seconds

	TEAM_A	TEAM_B
1	Aus	Eng
2	Aus	India
3	Aus	Pak
4	Eng	India
5	Eng	Pak
6	India	Pak

Explanation: How It Works?

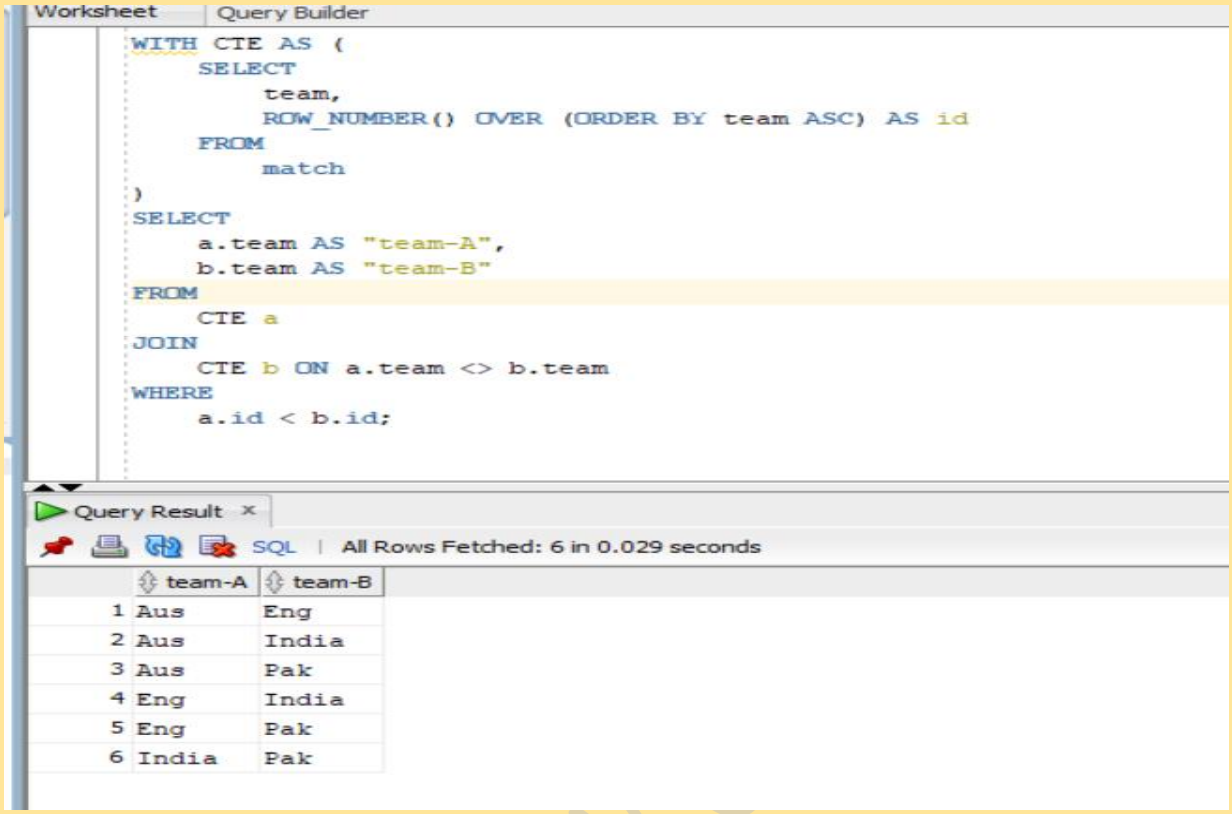
CROSS JOIN: The CROSS JOIN between match t1 and match t2 generates all possible team pairings, including self-pairings (e.g., "India vs India").

WHERE Condition:

- **WHERE t1.team < t2.team filters out:**
- **Self-matches like ("India", "India").**
- **Reverse duplicates by maintaining only lexicographically smaller-to-larger pairs, e.g., "Aus vs Eng" but not "Eng vs Aus".**

1. **ORDER BY Clause: ORDER BY t1.team, t2.team ASC ensures the output is sorted alphabetically by both columns, giving a neat and predictable order.**

2. Alternative Query Solution: Using Recursive CTE with Row_Number() :



How This Query Works:

Step-by-Step Explanation:

Step 1: Using CTE with ROW_NUMBER():

```
WITH CTE AS (
  SELECT  *, ROW_NUMBER() OVER (ORDER BY team ASC) AS id
  FROM match )
```

- The CTE (Common Table Expression) creates a temporary result set with an additional id column.
- ROW_NUMBER() OVER (ORDER BY team ASC) assigns a unique sequential number to each team in alphabetical order.

team	id
Aus	1
Eng	2
India	3
Pak	4

Step 2: Joining the CTE on Different Teams:

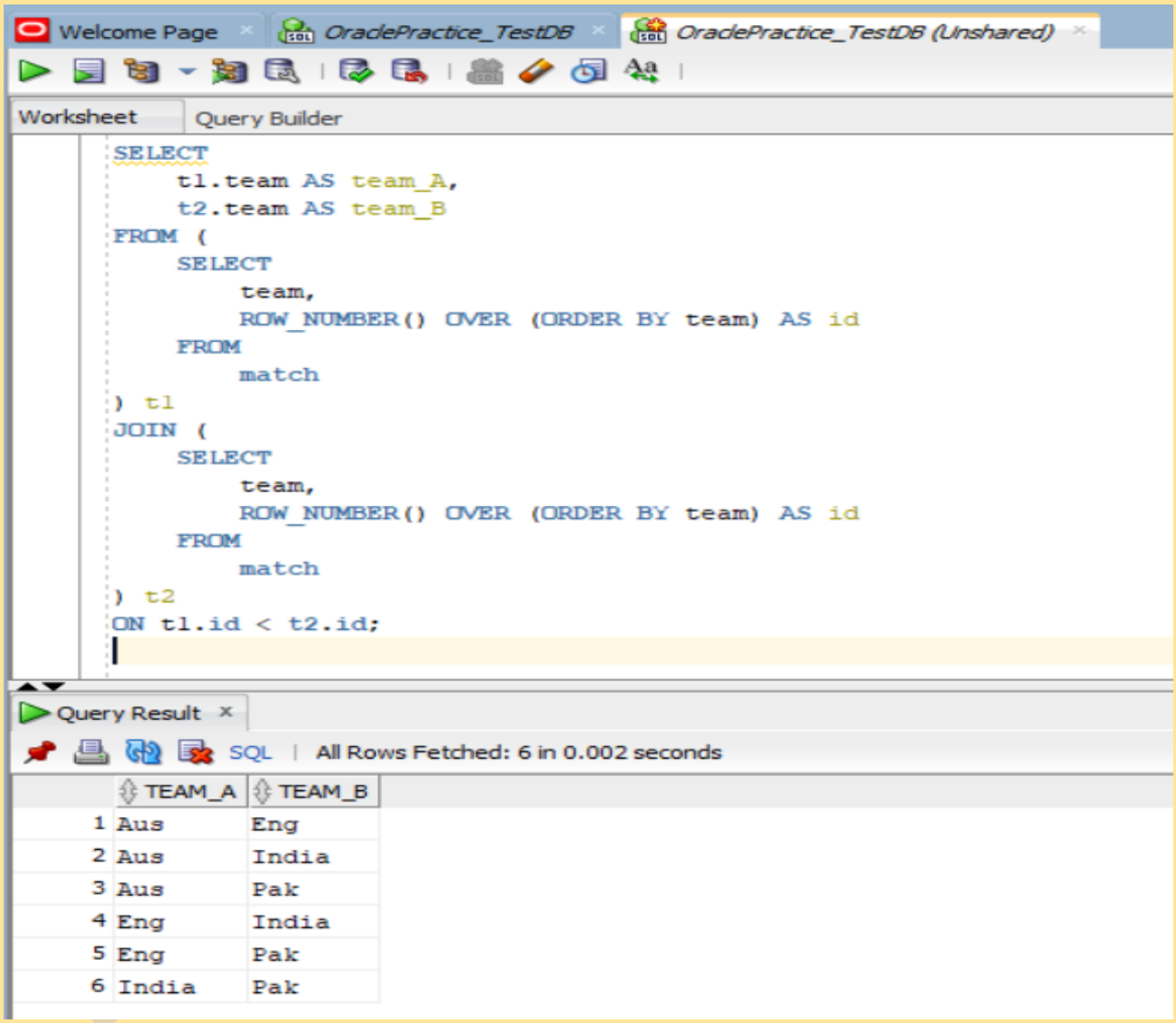
```
SELECT
   a.team AS "team-A",  b.team AS "team-B"
FROM  CTE AS a
JOIN  CTE AS b ON a.team <> b.team
```

- The JOIN on a.team <> b.team ensures all possible combinations of teams, avoiding self-matches like "India vs India."

Step 3: Filtering to Avoid Reverse Duplicates: WHERE a.id < b.id;

- The WHERE a.id < b.id condition ensures only unique matchups are included by enforcing a consistent ordering based on the id column.
- This eliminates reverse duplicates, providing clean results.

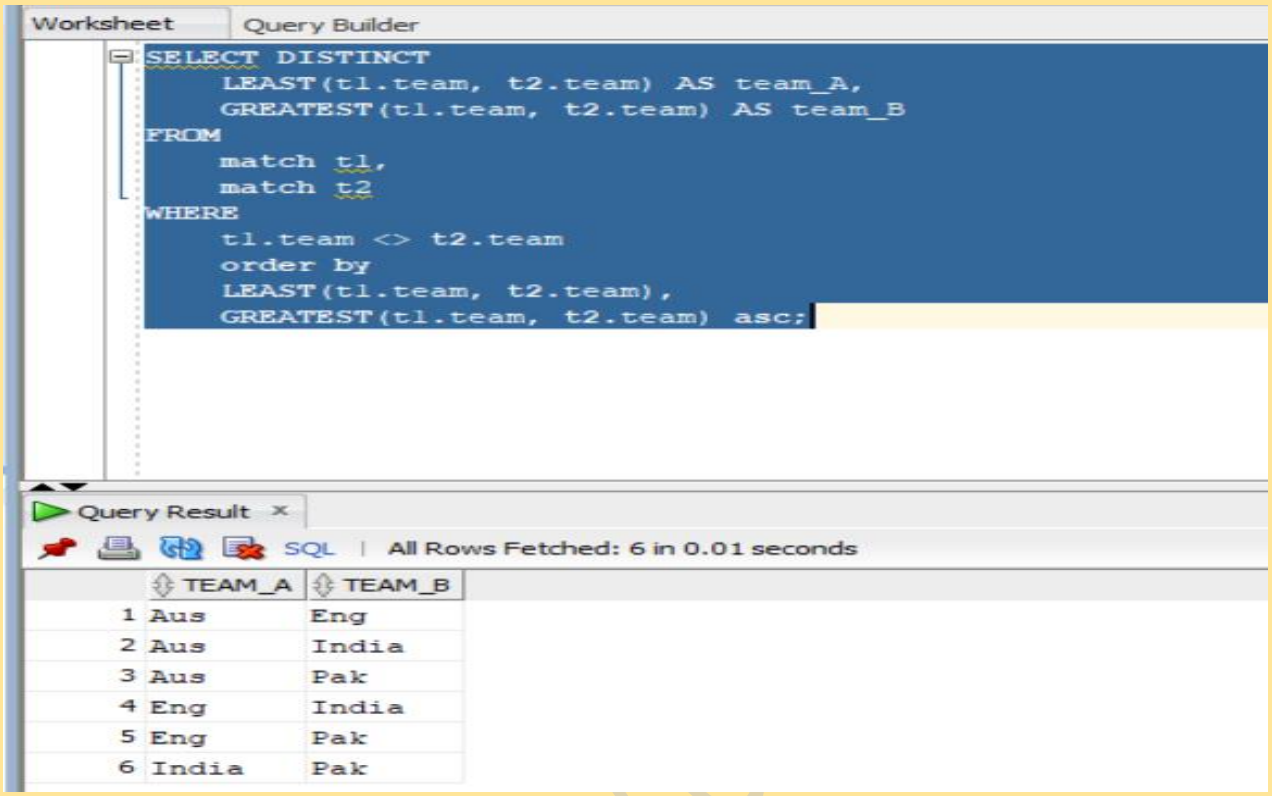
3.Another Alternative solution for the same query: Using Self-Join with ROW_NUMBER() and Comparison:



Explanation:

1. **Two subqueries generate row numbers for each team.**
2. **The JOIN on id < id avoids reverse duplicates and self-matches.**

Alternative solution 4: Using Cartesian Product and DISTINCT:

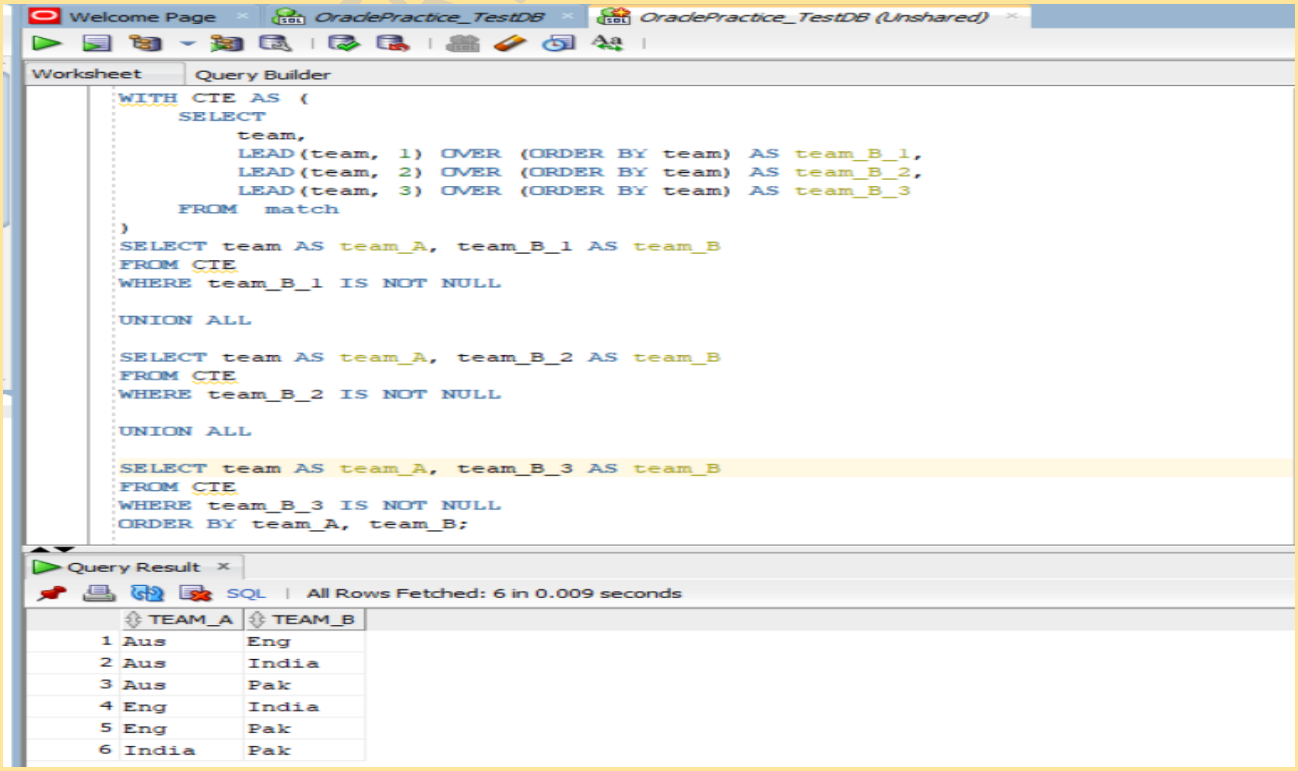


	TEAM_A	TEAM_B
1	Aus	Eng
2	Aus	India
3	Aus	Pak
4	Eng	India
5	Eng	Pak
6	India	Pak

Explanation:

1. **The LEAST() and GREATEST() functions arrange teams in a consistent order.**
2. **DISTINCT ensures uniqueness.**

Solution 5: LEAD() Function Approach:



	TEAM_A	TEAM_B
1	Aus	Eng
2	Aus	India
3	Aus	Pak
4	Eng	India
5	Eng	Pak
6	India	Pak

Explanation:

1. **Generate All Unique Matches:**
   - **The LEAD() function creates matches with the next, second next, and third next teams.**
2. **Combine Results Using UNION ALL:**
   - **Ensures all possible combinations are included in the result set.**
3. **Sorting the Results: ORDER BY team_A, team_B;**

- **ORDER BY team_A**: Sorts the primary column alphabetically.
- **ORDER BY team_B**: Sorts the secondary column alphabetically for each **team_A**.

Question 2. Write a query to get the output shown in the image. The input table contains employee IDs and names, and the expected output should group employees in pairs, displaying them as a concatenated string along with a group number.

INPUT

id integer	name character varying (10)
1	Emp1
2	Emp2
3	Emp3
4	Emp4
5	Emp5
6	Emp6
7	Emp7
8	Emp8

OUTPUT

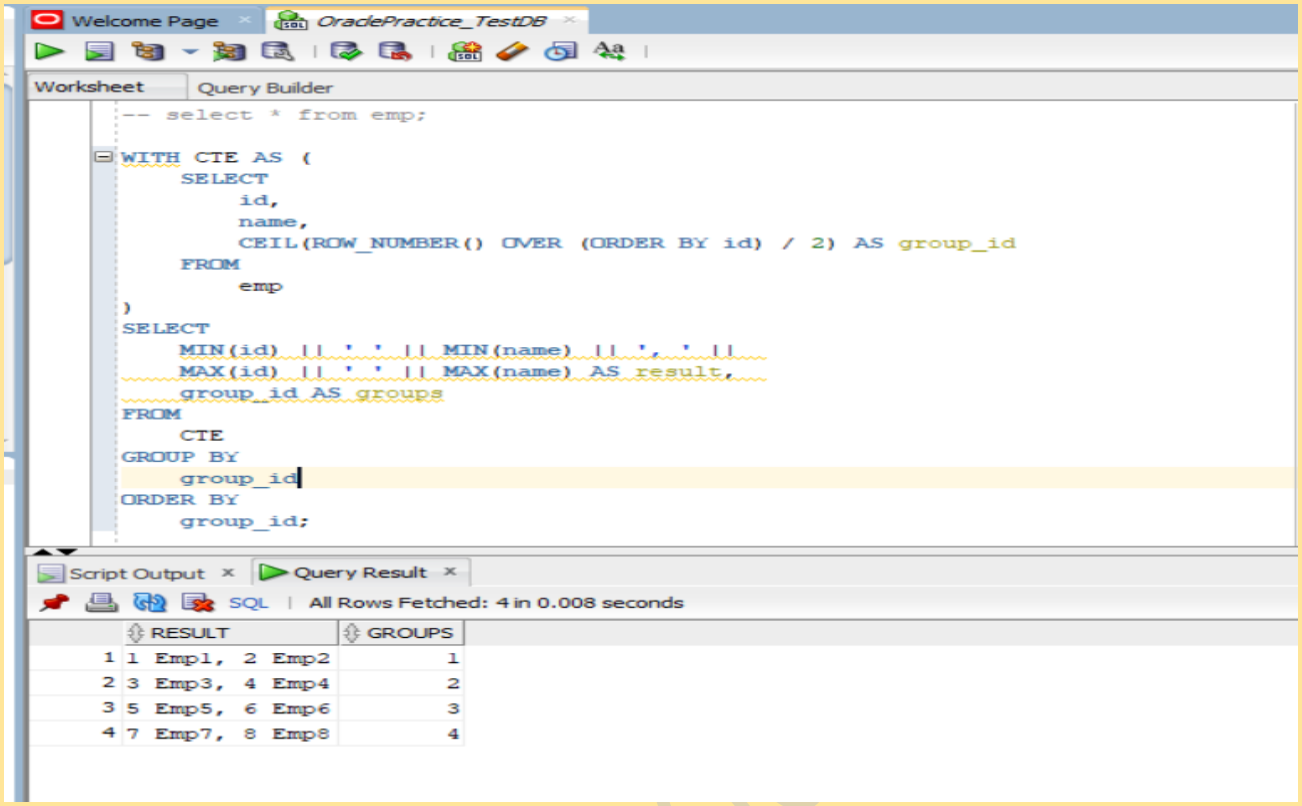
result text	groups integer
1 Emp1, 2 Emp2	1
3 Emp3, 4 Emp4	2
5 Emp5, 6 Emp6	3
7 Emp7, 8 Emp8	4

Step-by-Step Solution:

1. Create the Input Table and Insert Data:

```
CREATE TABLE emp (
   id INT,
   name VARCHAR(10));
INSERT INTO emp (id, name) VALUES
   (1, 'Emp1'),
   (2, 'Emp2'),
   (3, 'Emp3'),
   (4, 'Emp4'),
   (5, 'Emp5'),
   (6, 'Emp6'),
   (7, 'Emp7'),
   (8, 'Emp8');
COMMIT;
```

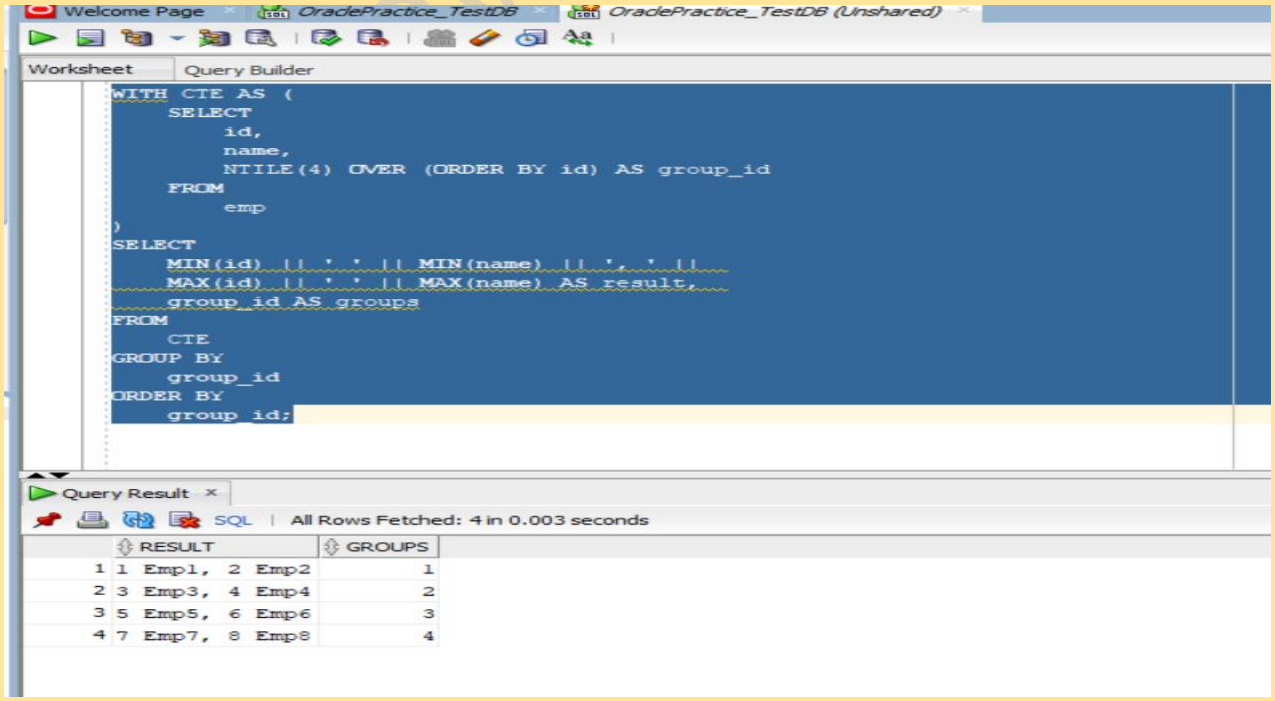
Solution 1: Using ROW_NUMBER() and CEIL() Functions:



✅ Why Use This?

- **Simple and effective: Uses window functions to generate row numbers.**
- **Grouping: Achieved through CEIL() function by dividing the row number by 2.**
- **Good for moderate-sized datasets.**

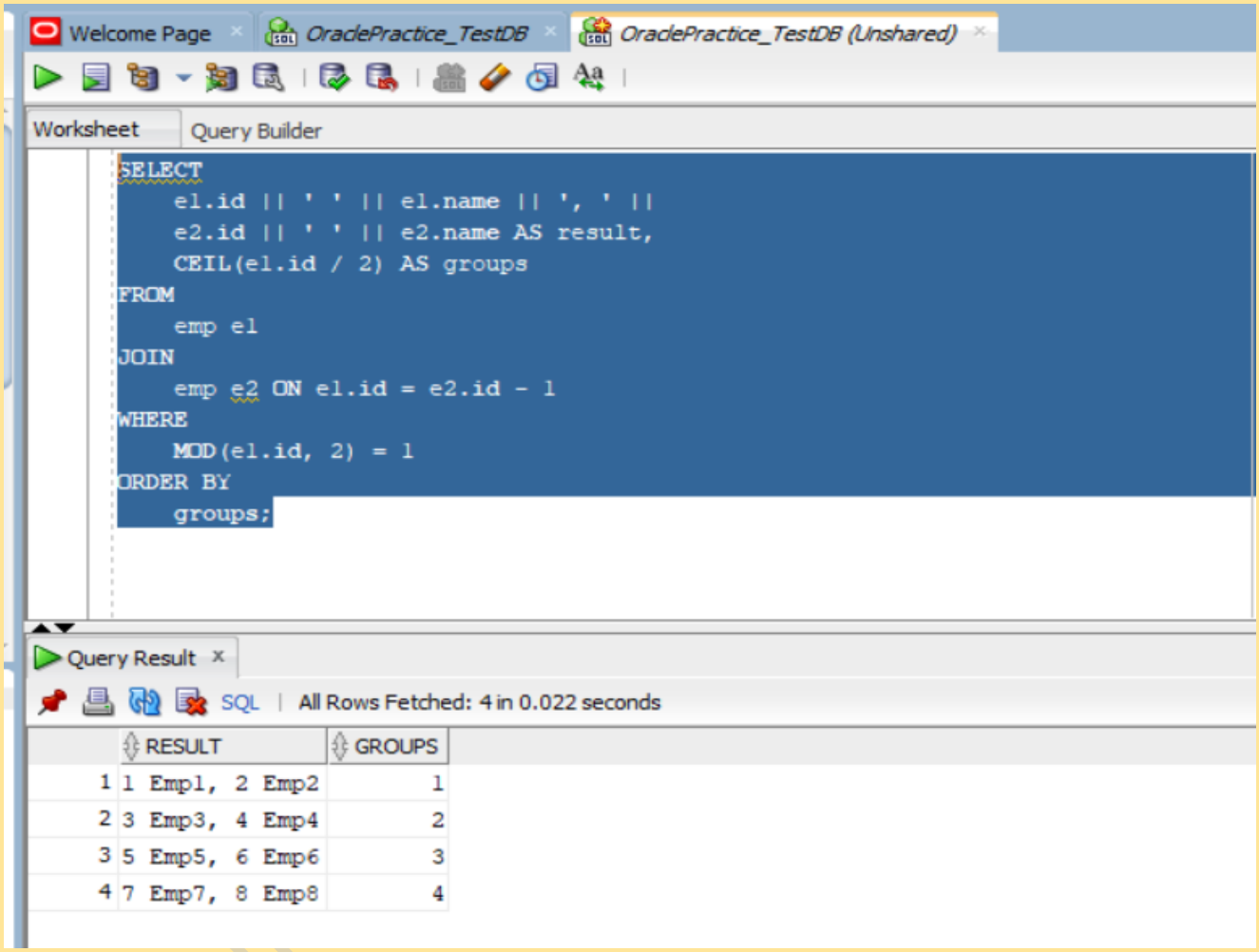
Solution 2: Using NTILE() Function for Grouping:



✅ Why Use This?

- **NTILE() function evenly distributes rows into 4 groups.**
- **Ideal when you know the exact number of groups needed.**
- **Automatically balances rows per group, useful in dynamic datasets.**

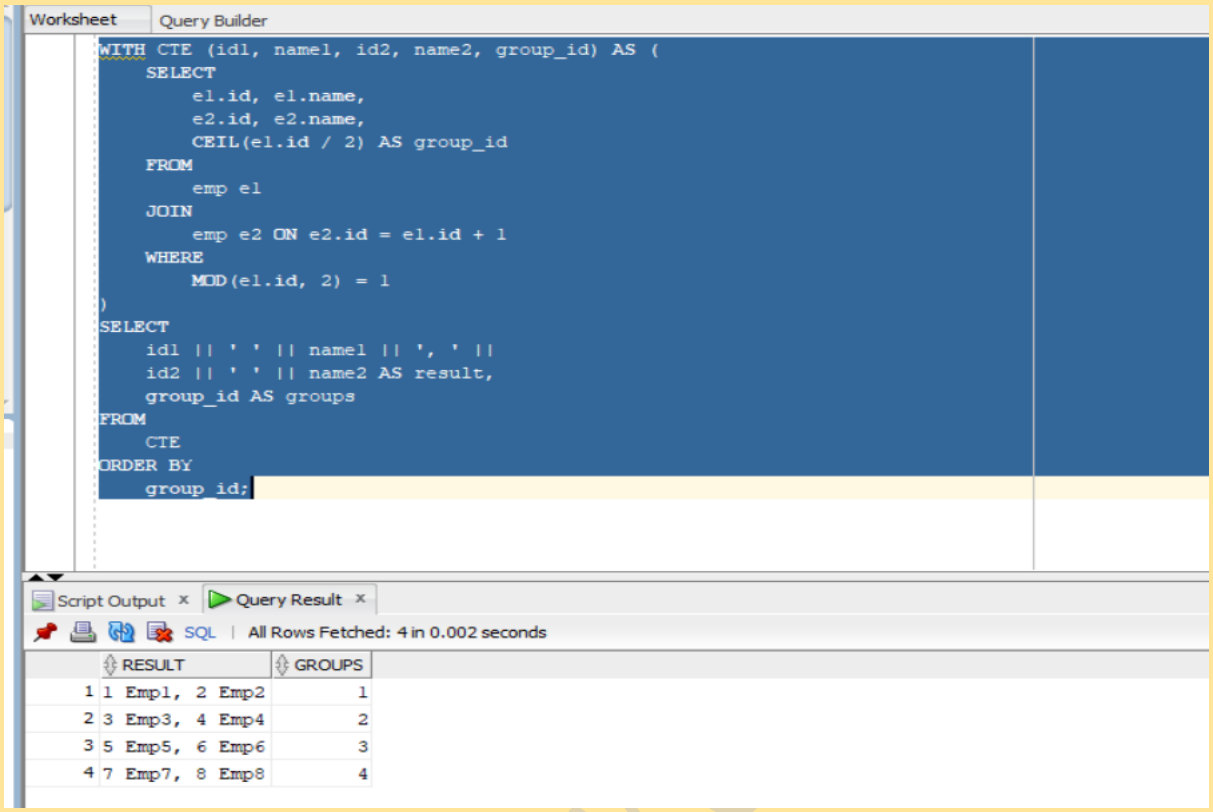
□ Solution 3: Using Self-Join on Sequential IDs:



✅ Why Use This?

- **Self-join to pair sequential IDs.**
- **No need for window functions or CTEs.**
- **Useful when you want to maintain strict ID order.**

Solution 4: Recursive CTE Approach (Oracle-Compatible):



```
WITH CTE (id1, name1, id2, name2, group_id) AS (
    SELECT
        e1.id, e1.name,
        e2.id, e2.name,
        CEIL(e1.id / 2) AS group_id
    FROM
        emp e1
    JOIN
        emp e2 ON e2.id = e1.id + 1
    WHERE
        MOD(e1.id, 2) = 1
)
SELECT
    id1 || ' ' || name1 || ', ' ||
    id2 || ' ' || name2 AS result,
    group_id AS groups
FROM
    CTE
ORDER BY
    group_id;
```

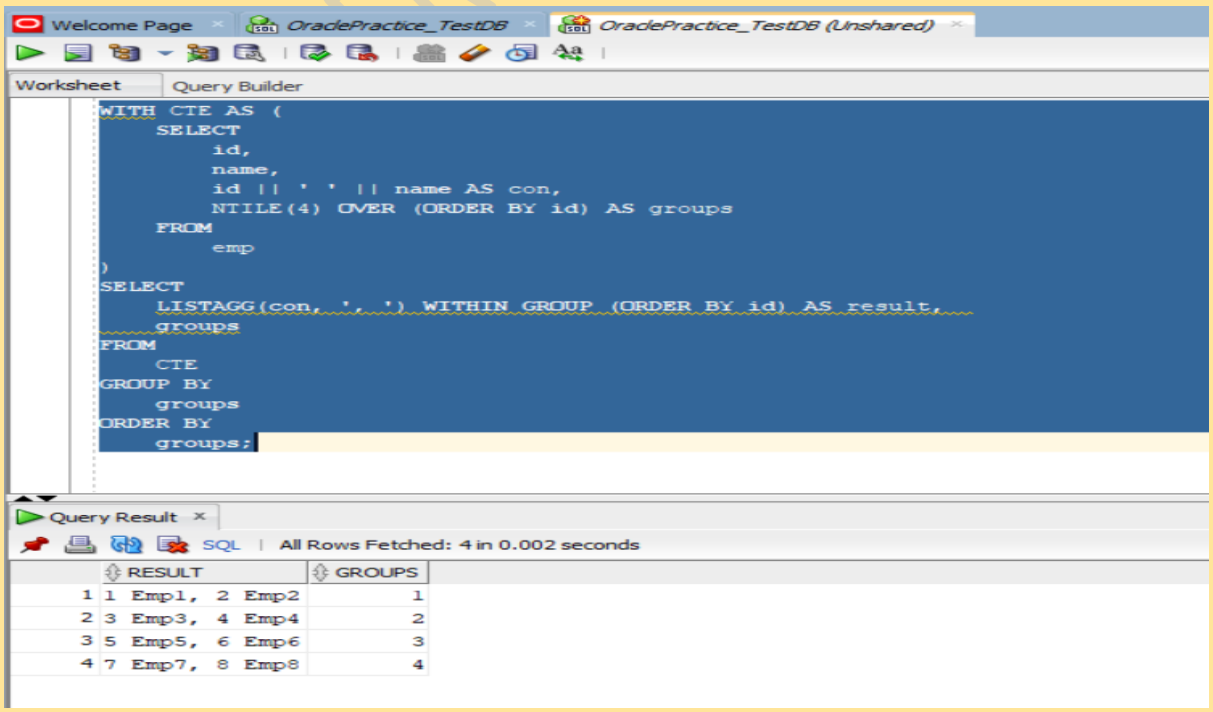
Script Output | Query Result | SQL | All Rows Fetched: 4 in 0.002 seconds

	RESULT	GROUPS
1	1 Emp1, 2 Emp2	1
2	3 Emp3, 4 Emp4	2
3	5 Emp5, 6 Emp6	3
4	7 Emp7, 8 Emp8	4

✓ Why Use This?

- **Removed the WITH RECURSIVE syntax. Oracle only requires WITH CTE AS.**
- **Corrected the column list syntax in the WITH CTE declaration.**

Solution 5: Using LISTAGG and NTILE in Oracle:



```
WITH CTE AS (
    SELECT
        id,
        name,
        id || ' ' || name AS con,
        NTILE(4) OVER (ORDER BY id) AS groups
    FROM
        emp
)
SELECT
    LISTAGG(con, ', ') WITHIN GROUP (ORDER BY id) AS result,
    groups
FROM
    CTE
GROUP BY
    groups
ORDER BY
    groups;
```

Query Result | SQL | All Rows Fetched: 4 in 0.002 seconds

	RESULT	GROUPS
1	1 Emp1, 2 Emp2	1
2	3 Emp3, 4 Emp4	2
3	5 Emp5, 6 Emp6	3
4	7 Emp7, 8 Emp8	4

How This Works:

1. **Concatenate ID and Name: id || ' ' || name AS con**

Creates a concatenated string of ID and name to match the desired format.

2. **Generate Group Numbers with NTILE:**

NTILE(4) OVER (ORDER BY id) AS groups

NTILE(4) divides the rows into 4 equal groups, ensuring 2 employees per group.

3. **Concatenate Strings Using LISTAGG:**

LISTAGG(con, ', ') WITHIN GROUP (ORDER BY id) AS result

- **Similar to STRING_AGG, LISTAGG concatenates the employee pairs with a comma separator.**
- **WITHIN GROUP (ORDER BY id) ensures the correct order of IDs within each group.**

4. **Group and Sort the Results: GROUP BY groups**
   ORDER BY groups;

- **Groups by groups to produce one row per pair.**
- **Orders by groups to maintain the expected sequence.**

Top Advanced SQL Interview Questions

Question 3. Write a SQL Query to generate Cricket match fixtures for Asia Cup.

Scenario:

The Asia Cup is approaching, and all participating cricket teams need to play against each other. The objective is to generate a list of match fixtures where each team plays once against every other team, ensuring:

- **No duplicate matches, e.g., "India vs Pakistan" and "Pakistan vs India" should not both appear.**
- **No self-matches, e.g., a team should not play against itself ("India vs India" should not appear).**

Source Table: TEAMS

The input data is stored in a **table named TEAMS**, which contains a list of **participating countries**:

COUNTRY
India
Srilanka
Bangladesh
Pakistan

Expected Output:

The output should provide the possible match fixtures between all the teams participating in Asia cup.

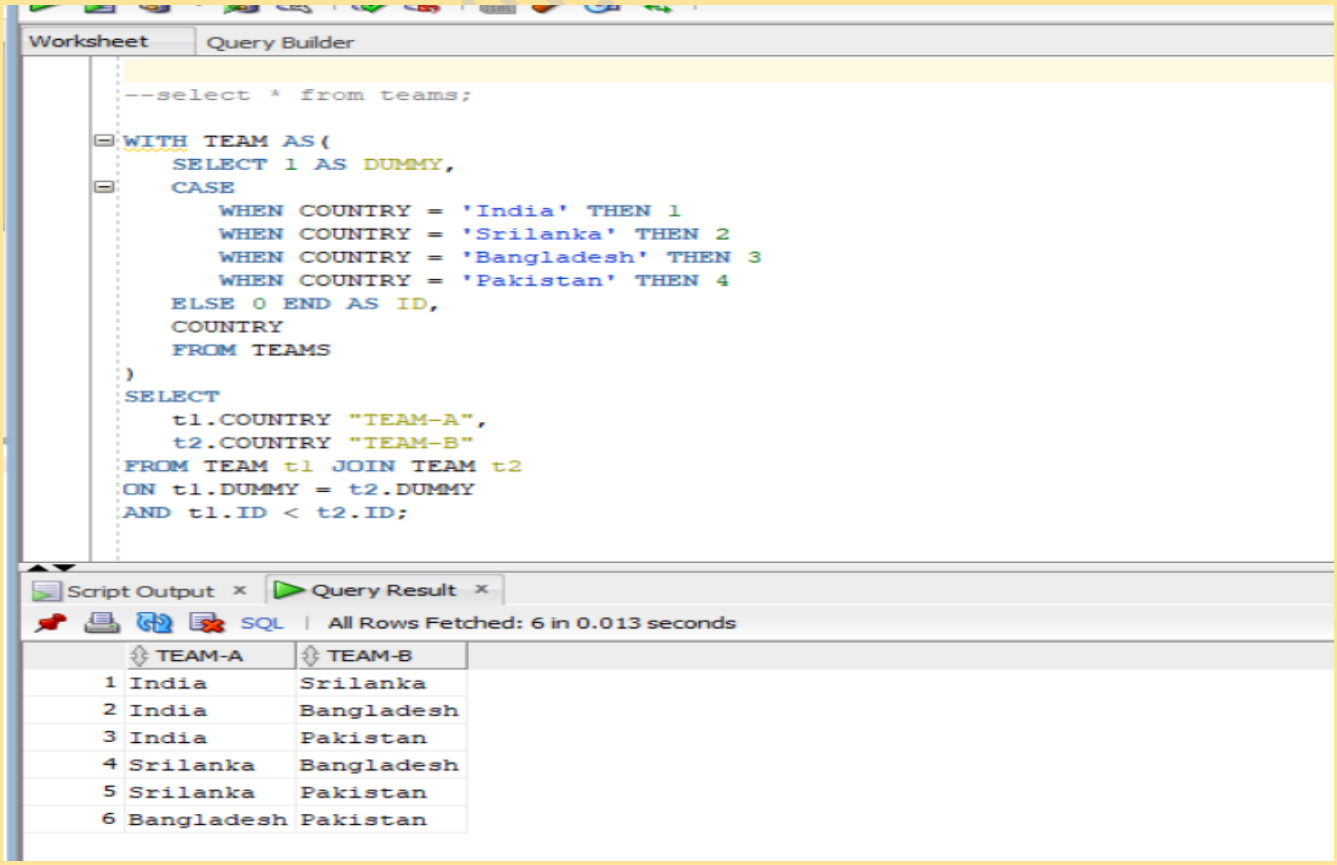
TEAM-A	TEAM-2
India	Srilanka
India	Bangladesh
India	Pakistan
Srilanka	Bangladesh
Srilanka	Pakistan
Bangladesh	Pakistan

Step by step Solution:

```
CREATE TABLE TEAMS (
  COUNTRY VARCHAR2(50));

INSERT INTO TEAMS VALUES('India');
INSERT INTO TEAMS VALUES('Srilanka');
INSERT INTO TEAMS VALUES('Bangladesh');
INSERT INTO TEAMS VALUES('Pakistan');
COMMIT;
```

1. Solution: Using Case Statement with Self join:



Question 4: Generate all possible cricket match fixtures for the Asia Cup in a round-robin format based on the sequential input order of the countries. The fixtures should cover all combinations without duplicates and maintain the given input sequence.

Input:

A TEAMS table with a single COUNTRY column, containing countries in sequential order:

COUNTRY
India
Srilanka
Pakistan
Bangladesh

Expected Output:
The output should list the match fixtures in the specified sequential order:

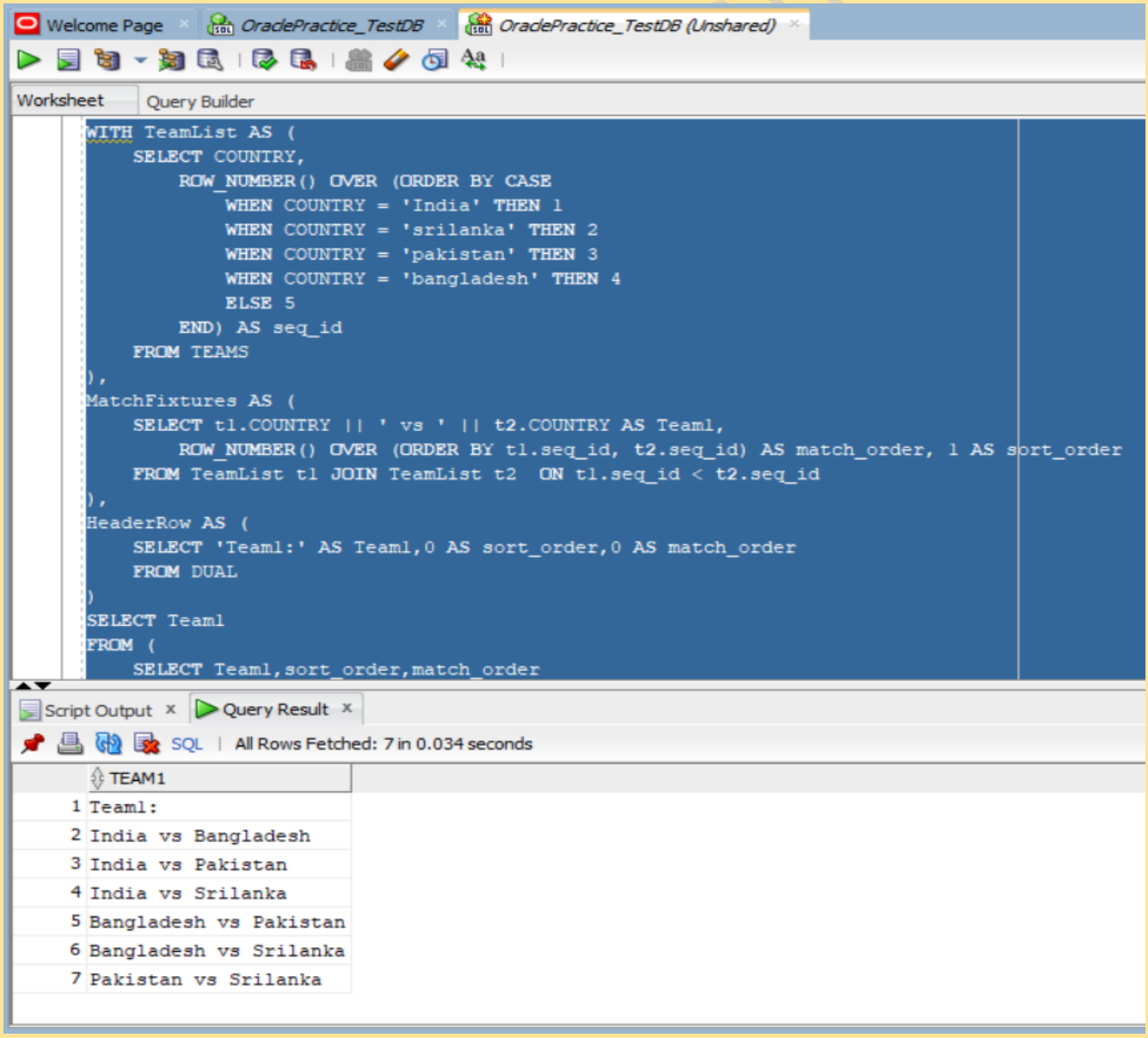
	TEAM1
1	Team1:
2	India vs Bangladesh
3	India vs Pakistan
4	India vs Srilanka
5	Bangladesh vs Pakistan
6	Bangladesh vs Srilanka
7	Pakistan vs Srilanka

Solution Approach:

Query:

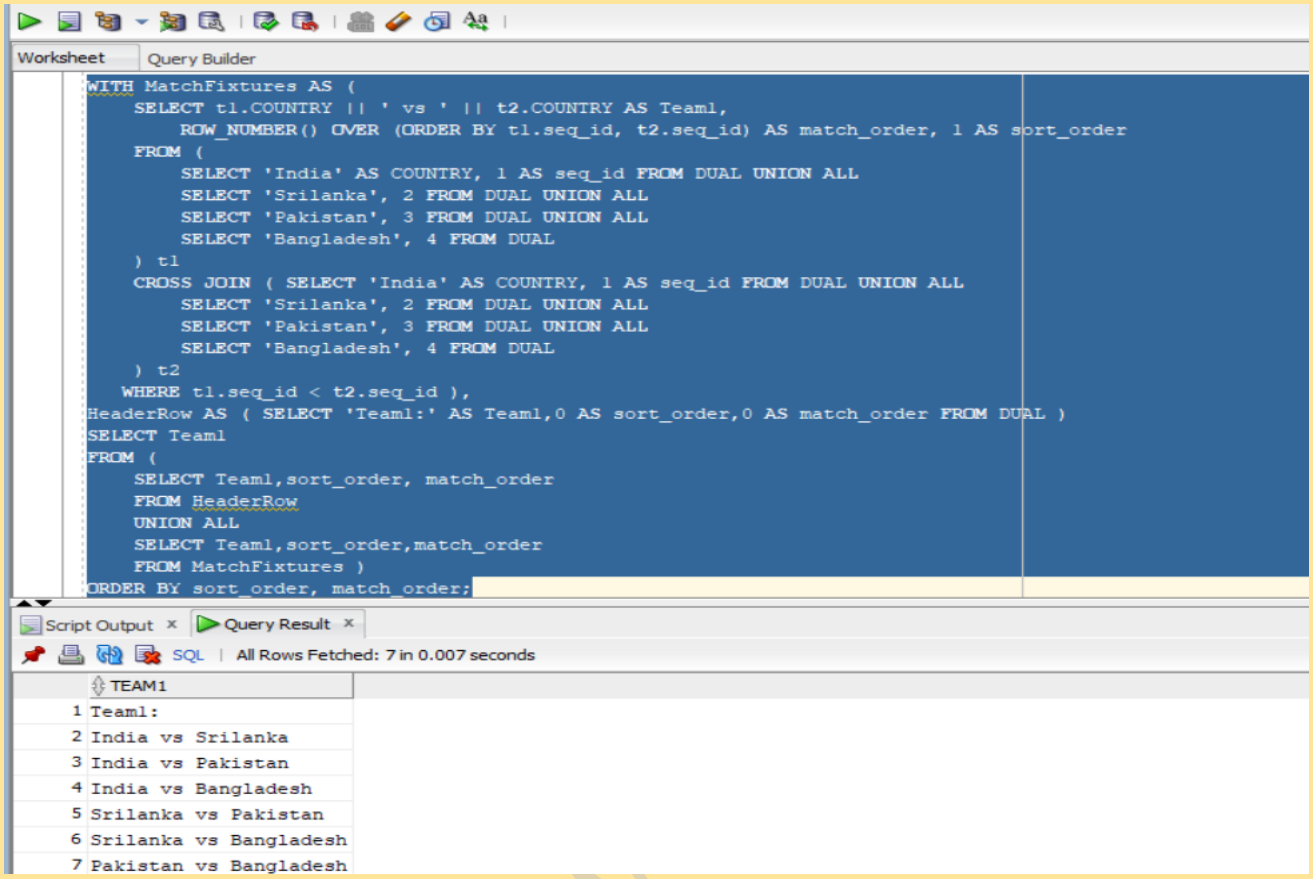
```
WITH TeamList AS (
   SELECT COUNTRY,
      ROW_NUMBER() OVER (ORDER BY CASE
         WHEN COUNTRY = 'India' THEN 1
         WHEN COUNTRY = 'srilanka' THEN 2
         WHEN COUNTRY = 'pakistan' THEN 3
         WHEN COUNTRY = 'bangladesh' THEN 4
         ELSE 5
      END) AS seq_id
   FROM TEAMS
),
MatchFixtures AS (
   SELECT t1.COUNTRY || ' vs ' || t2.COUNTRY AS Team1,
      ROW_NUMBER() OVER (ORDER BY t1.seq_id, t2.seq_id) AS match_order, 1 AS
sort_order
   FROM TeamList t1 JOIN TeamList t2  ON t1.seq_id < t2.seq_id
```

```
),
HeaderRow AS (
   SELECT 'Team1:' AS Team1,0 AS sort_order,0 AS match_order
   FROM DUAL
)
SELECT Team1
FROM (
   SELECT Team1,sort_order,match_order
   FROM HeaderRow
   UNION ALL
   SELECT Team1,sort_order, match_order
   FROM  MatchFixtures
)
ORDER BY
   sort_order, match_order;
```



Solution Approach -2 :

```
WITH MatchFixtures AS (
   SELECT t1.COUNTRY || ' vs ' || t2.COUNTRY AS Team1,
      ROW_NUMBER() OVER (ORDER BY t1.seq_id, t2.seq_id) AS match_order, 1 AS
sort_order
   FROM (
      SELECT 'India' AS COUNTRY, 1 AS seq_id FROM DUAL UNION ALL
      SELECT 'Srilanka', 2 FROM DUAL UNION ALL
      SELECT 'Pakistan', 3 FROM DUAL UNION ALL
      SELECT 'Bangladesh', 4 FROM DUAL
   ) t1
   CROSS JOIN (
      SELECT 'India' AS COUNTRY, 1 AS seq_id FROM DUAL UNION ALL
      SELECT 'Srilanka', 2 FROM DUAL UNION ALL
      SELECT 'Pakistan', 3 FROM DUAL UNION ALL
      SELECT 'Bangladesh', 4 FROM DUAL
   ) t2
   WHERE t1.seq_id < t2.seq_id
),
HeaderRow AS (
   SELECT 'Team1:' AS Team1,0 AS sort_order,0 AS match_order
   FROM DUAL
)
SELECT Team1
FROM (
   SELECT Team1,sort_order, match_order
   FROM HeaderRow
   UNION ALL
   SELECT Team1,sort_order,match_order
   FROM MatchFixtures
)
ORDER BY sort_order, match_order;
```



```
WITH MatchFixtures AS (
    SELECT t1.COUNTRY || ' vs ' || t2.COUNTRY AS Team1,
        ROW_NUMBER() OVER (ORDER BY t1.seq_id, t2.seq_id) AS match_order, 1 AS sort_order
    FROM (
        SELECT 'India' AS COUNTRY, 1 AS seq_id FROM DUAL UNION ALL
        SELECT 'Srilanka', 2 FROM DUAL UNION ALL
        SELECT 'Pakistan', 3 FROM DUAL UNION ALL
        SELECT 'Bangladesh', 4 FROM DUAL
    ) t1
    CROSS JOIN ( SELECT 'India' AS COUNTRY, 1 AS seq_id FROM DUAL UNION ALL
        SELECT 'Srilanka', 2 FROM DUAL UNION ALL
        SELECT 'Pakistan', 3 FROM DUAL UNION ALL
        SELECT 'Bangladesh', 4 FROM DUAL
    ) t2
   WHERE t1.seq_id < t2.seq_id ),
HeaderRow AS ( SELECT 'Team1:' AS Team1,0 AS sort_order,0 AS match_order FROM DUAL )
SELECT Team1
FROM (
    SELECT Team1,sort_order, match_order
    FROM HeaderRow
    UNION ALL
    SELECT Team1,sort_order,match_order
    FROM MatchFixtures )
ORDER BY sort_order, match_order;
```

Script Output | Query Result | SQL | All Rows Fetched: 7 in 0.007 seconds

	TEAM1
1	Team1:
2	India vs Srilanka
3	India vs Pakistan
4	India vs Bangladesh
5	Srilanka vs Pakistan
6	Srilanka vs Bangladesh
7	Pakistan vs Bangladesh

Question 5. Write a SQL Query to find the number of matches played, won, lost and tied by each team in Asia cup?

Source:

The Source table *Match_Results* consists of the results of matches played in Asia Cup.

Input:

TEAM_A	TEAM_B	RESULT
India	Bangladesh	India
India	Pakistan	India
India	Srilanka	(null)
Srilanka	Bangladesh	Srilanka
Srilanka	Pakistan	Pakistan
Bangladesh	Pakistan	Bangladesh

Match_Results

Expected Output:

The output should contain the number of matches played, won, lost and tied for each team as shown below.

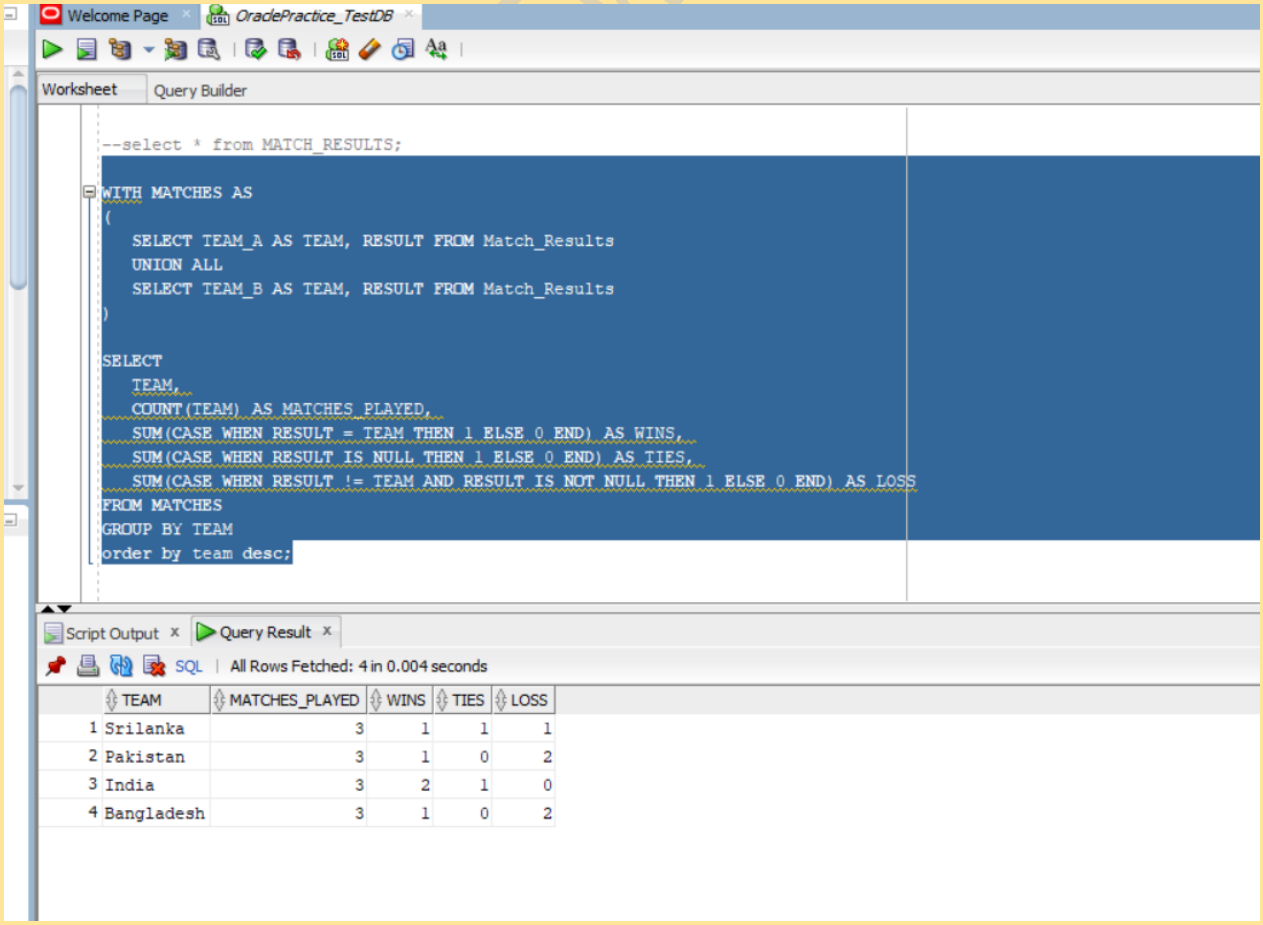
TEAM	MATCHES_PLAYED	WINS	TIES	LOSS
Srilanka	3	1	1	1
Pakistan	3	1	0	2
India	3	2	1	0
Bangladesh	3	1	0	2

Solution Approach-1:

```
CREATE TABLE MATCH_RESULTS(
  TEAM_A VARCHAR(10),
  TEAM_B VARCHAR(10),
  RESULT VARCHAR(10)
);

INSERT INTO MATCH_RESULTS VALUES('India','Bangladesh','India');
INSERT INTO MATCH_RESULTS VALUES('India','Pakistan','India');
INSERT INTO MATCH_RESULTS VALUES('India','Srilanka','');
INSERT INTO MATCH_RESULTS VALUES('Srilanka','Bangladesh','Srilanka');
INSERT INTO MATCH_RESULTS VALUES('Srilanka','Pakistan','Pakistan');
INSERT INTO MATCH_RESULTS VALUES('Bangladesh','Pakistan','Bangladesh');
Commit;
```

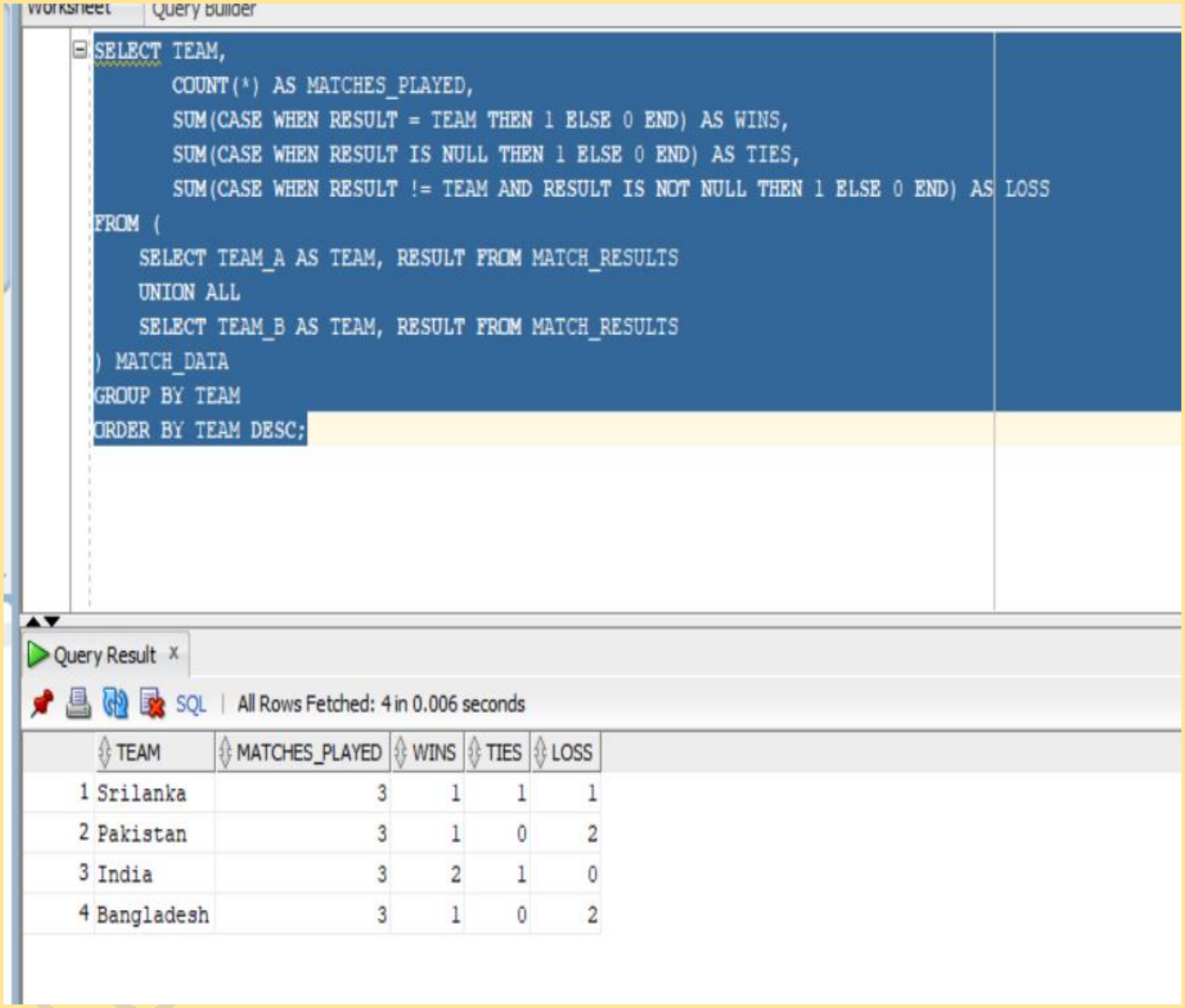
Solution Approach-2 : Using CTE and Conditional Aggregation



Explanation:

1. **The WITH MATCHES CTE creates a combined view of all teams and results.**
2. **The main query calculates matches played, wins, ties, and losses using aggregation functions.**
3. **The CASE statements handle conditional counting for wins, ties, and losses**

Solution Approach-3: Using Derived Tables and Joins:



```
SELECT TEAM,
       COUNT(*) AS MATCHES_PLAYED,
       SUM(CASE WHEN RESULT = TEAM THEN 1 ELSE 0 END) AS WINS,
       SUM(CASE WHEN RESULT IS NULL THEN 1 ELSE 0 END) AS TIES,
       SUM(CASE WHEN RESULT != TEAM AND RESULT IS NOT NULL THEN 1 ELSE 0 END) AS LOSS
FROM (
    SELECT TEAM_A AS TEAM, RESULT FROM MATCH_RESULTS
    UNION ALL
    SELECT TEAM_B AS TEAM, RESULT FROM MATCH_RESULTS
) MATCH_DATA
GROUP BY TEAM
ORDER BY TEAM DESC;
```

Query Result

All Rows Fetched: 4 in 0.006 seconds

	TEAM	MATCHES_PLAYED	WINS	TIES	LOSS
1	Srilanka	3	1	1	1
2	Pakistan	3	1	0	2
3	India	3	2	1	0
4	Bangladesh	3	1	0	2

Solution Approach - 4: Using FULL OUTER JOIN and Aggregation

```
--select * from MATCH_RESULTS;
SELECT
    TEAMS.COUNTRY AS TEAM,
    COUNT(MATCH_RESULTS.RESULT) AS MATCHES_PLAYED,
    SUM(CASE WHEN MATCH_RESULTS.RESULT = TEAMS.COUNTRY THEN 1 ELSE 0 END) AS WINS,
    SUM(CASE WHEN MATCH_RESULTS.RESULT IS NULL THEN 1 ELSE 0 END) AS TIES,
    SUM(CASE WHEN MATCH_RESULTS.RESULT != TEAMS.COUNTRY AND MATCH_RESULTS.RESULT IS NOT NULL THEN 1 ELSE 0 END) AS LOSS
FROM (
    SELECT TEAM_A AS COUNTRY FROM MATCH_RESULTS
    UNION
    SELECT TEAM_B AS COUNTRY FROM MATCH_RESULTS
) TEAMS
LEFT JOIN MATCH_RESULTS ON TEAMS.COUNTRY IN (MATCH_RESULTS.TEAM_A, MATCH_RESULTS.TEAM_B)
GROUP BY TEAMS.COUNTRY
ORDER BY TEAMS.COUNTRY DESC;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.006 seconds

	TEAM	MATCHES_PLAYED	WINS	TIES	LOSS
1	Srilanka	2	1	1	1
2	Pakistan	3	1	0	2
3	India	2	2	1	0
4	Bangladesh	3	1	0	2

Question 6: Write a SQL Query to find Min and Max values of continuous sequence in a group of elements.

Input

The source table 'ELEMENTS' consists of two fields: one with the details of an 'ELEMENT' and the other with a 'SEQUENCE' associated with the element.

INPUT:

ELEMENT	SEQUENCE
A	1
A	2
A	3
A	5
A	6
A	8
A	9
B	11
C	13
C	14
C	15

Elements

Expected Output:

The output should capture the element and the minimum and maximum of continuous sequence available for the element in the source table as shown below. In the source data, the first three

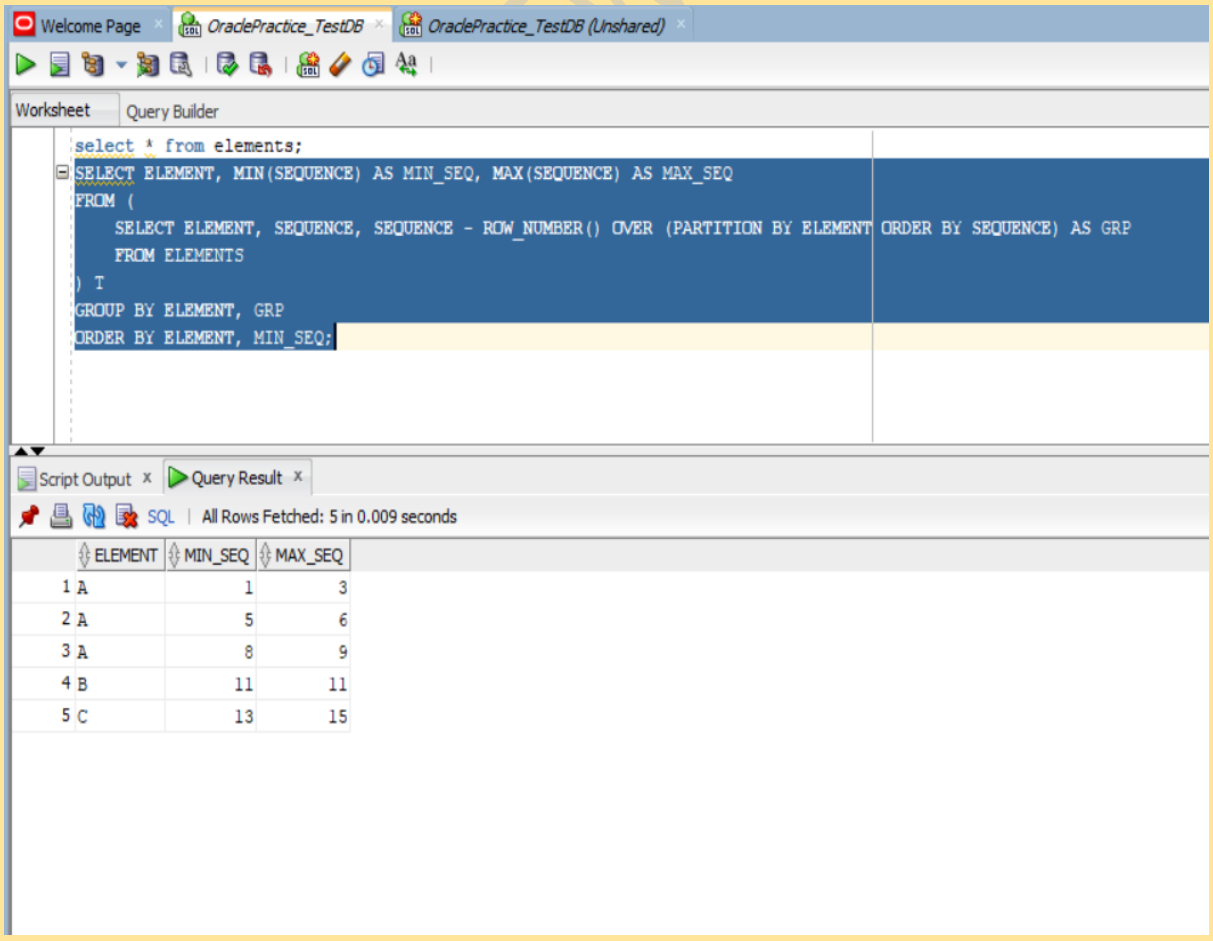
rows have a continuous sequence where the difference between consecutive rows is one. The sequence is broken from the fourth row. So the first three rows must be captured in a single row with element as A and the minimum and maximum sequence values as 1 and 3. Such continuous sequence patterns present in the entire source data must be captured.

ELEMENT	MIN_SEQ	MAX_SEQ
A	1	3
A	5	6
A	8	9
B	11	11
C	13	15

Solution Approach-1 :

The query to find the min and max values of continuous sequences in a group of elements is as follows:

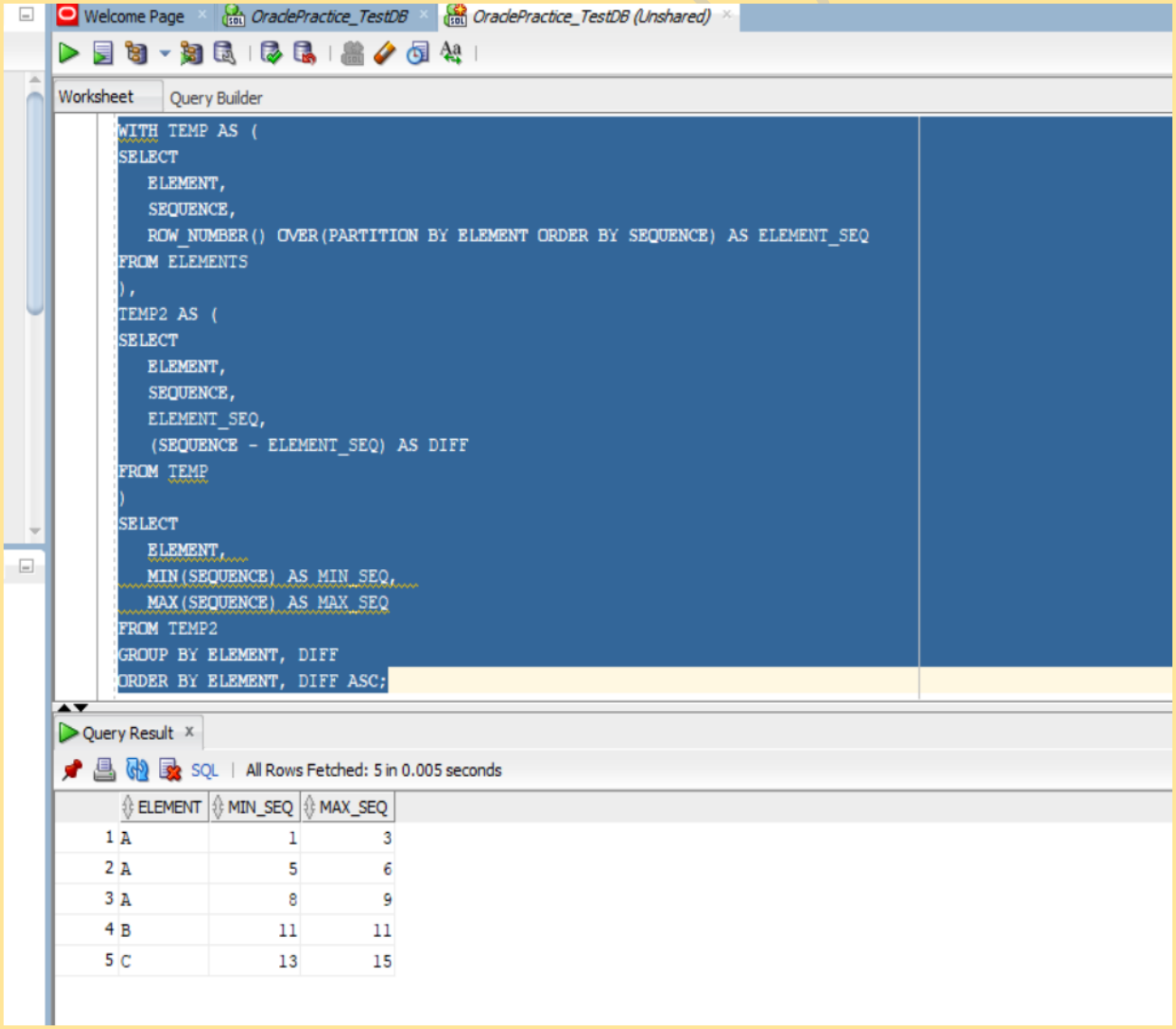
```
SELECT ELEMENT, MIN(SEQUENCE) AS MIN_SEQ, MAX(SEQUENCE) AS MAX_SEQ
FROM (
   SELECT ELEMENT, SEQUENCE, SEQUENCE - ROW_NUMBER() OVER (PARTITION
BY ELEMENT ORDER BY SEQUENCE) AS GRP
   FROM ELEMENTS
) T
GROUP BY ELEMENT, GRP
ORDER BY ELEMENT, MIN_SEQ;
```



Solution Approach 2: Using TEMP and DIFF

Query:

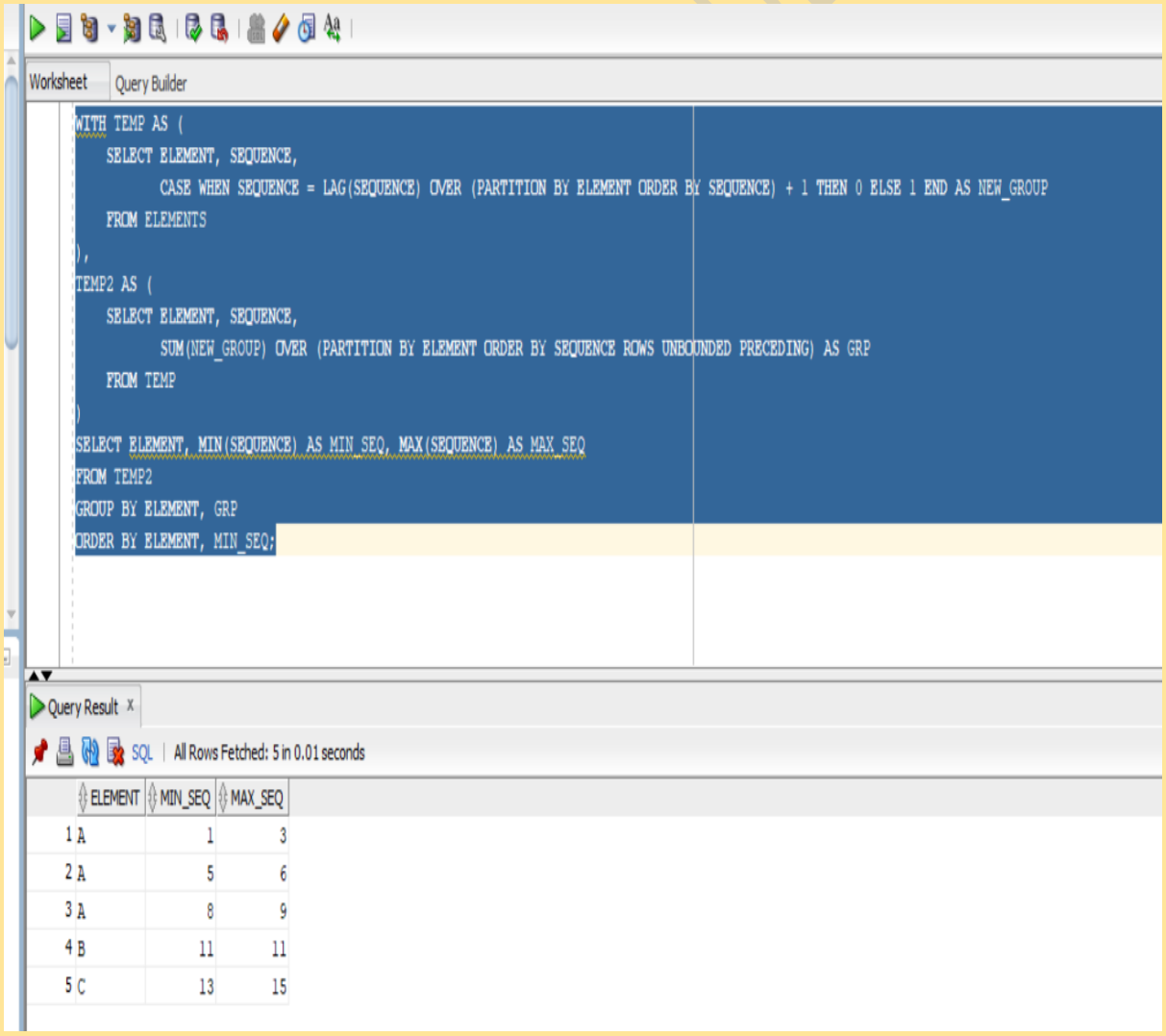
```
WITH TEMP AS (
SELECT ELEMENT,  SEQUENCE,
  ROW_NUMBER() OVER(PARTITION BY ELEMENT ORDER BY SEQUENCE) AS
ELEMENT_SEQ
FROM ELEMENTS
),
TEMP2 AS (
SELECT ELEMENT, SEQUENCE,  ELEMENT_SEQ, (SEQUENCE - ELEMENT_SEQ) AS
DIFF
FROM TEMP )
SELECT  ELEMENT,  MIN(SEQUENCE) AS MIN_SEQ, MAX(SEQUENCE) AS MAX_SEQ
FROM TEMP2
GROUP BY ELEMENT, DIFF
ORDER BY ELEMENT, DIFF ASC;
```



Solution: Approach 3: (Using LEAD and LAG Functions)

Query:

```
 WITH TEMP AS (
   SELECT ELEMENT, SEQUENCE,
        CASE WHEN SEQUENCE = LAG(SEQUENCE) OVER (PARTITION BY ELEMENT
ORDER BY SEQUENCE) + 1 THEN 0 ELSE 1 END AS NEW_GROUP
   FROM ELEMENTS
),
TEMP2 AS (
   SELECT ELEMENT, SEQUENCE,
        SUM(NEW_GROUP) OVER (PARTITION BY ELEMENT ORDER BY SEQUENCE
ROWS UNBOUNDED PRECEDING) AS GRP
   FROM TEMP
)
SELECT ELEMENT, MIN(SEQUENCE) AS MIN_SEQ, MAX(SEQUENCE) AS MAX_SEQ
FROM TEMP2
GROUP BY ELEMENT, GRP
ORDER BY ELEMENT, MIN_SEQ;
```



SQL Question 7:

You are given a table with cricket match results containing three columns: Home Team (H_T), Away Team (A_T), and Winner Team (W_T).
The table data is as shown below:

H_T	A_T	W_T
AUS	IND	IND
ENG	AUS	ENG
IND	AUS	AUS
AUS	ENG	AUS
ENG	IND	IND
IND	ENG	IND

Expected Output:
You need to write an SQL query to generate a report with the following columns:

- **Team Name: The name of the team.**
- **Total Matches Played: The total number of matches played by the team.**
- **Win Count: The total number of matches won by the team.**
- **Draw Count: The total number of matches drawn (in this case, assume no draws are present).**
- **Loss Count: The total number of matches lost by the team.**
- **Points: The total points of the team, calculated as:**
  - **Win = Win Count * 3**
  - **Draw = Draw Count * 1**
  - **Loss = Loss Count * 0**

The output should be sorted by points in descending order.

Team Name	Total Matches Played	Win Count	Draw Count	Loss Count	Points
IND	4	3	0	1	9
AUS	4	2	0	2	6
ENG	4	1	0	3	3

Step-by-Step Solution in Oracle SQL:

Step 1: Create the Table

```
CREATE TABLE Matches (
   H_T VARCHAR2(50),  -- Home Team
   A_T VARCHAR2(50),  -- Away Team
   W_T VARCHAR2(50)   -- Winner Team
);
```

Step 2: Insert Sample Data:

```
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('AUS', 'IND', 'IND');
```

```
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('ENG', 'AUS', 'ENG');
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('IND', 'AUS', 'AUS');
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('AUS', 'ENG', 'AUS');
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('ENG', 'IND', 'IND');
INSERT INTO Matches (H_T, A_T, W_T) VALUES ('IND', 'ENG', 'IND');

-- Commit the changes to the database
COMMIT;
```

Step 3: Verify Data Insertion:

```
SELECT * FROM Matches;
```

	H_T	A_T	W_T
1	AUS	IND	IND
2	ENG	AUS	ENG
3	IND	AUS	AUS
4	AUS	ENG	AUS
5	ENG	IND	IND
6	IND	ENG	IND

Solution Approach 1: Step 4: Generate the Required Report Query:

```
SELECT
    Team AS "Team Name",
    COUNT(*) AS "Total Matches Played",
    SUM(CASE WHEN Team = W_T THEN 1 ELSE 0 END) AS "Win Count",
    0 AS "Draw Count", -- No draws in the provided data
    SUM(CASE WHEN Team != W_T THEN 1 ELSE 0 END) AS "Loss Count",
    (SUM(CASE WHEN Team = W_T THEN 1 ELSE 0 END) * 3) AS "Points"
FROM (
    SELECT H_T AS Team, W_T FROM Matches
    UNION ALL
    SELECT A_T AS Team, W_T FROM Matches
) AllMatches
GROUP BY Team
ORDER BY "Points" DESC;
```

```sql
SELECT
    Team AS "Team Name",
    COUNT(*) AS "Total Matches Played",
    SUM(CASE WHEN Team = W_T THEN 1 ELSE 0 END) AS "Win Count",
    0 AS "Draw Count", -- No draws in the provided data
    SUM(CASE WHEN Team != W_T THEN 1 ELSE 0 END) AS "Loss Count",
    (SUM(CASE WHEN Team = W_T THEN 1 ELSE 0 END) * 3) AS "Points"
FROM (
    SELECT H_T AS Team, W_T FROM Matches
    UNION ALL
    SELECT A_T AS Team, W_T FROM Matches
) AllMatches
GROUP BY Team
ORDER BY "Points" DESC;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.007 seconds

	Team Name	Total Matches Played	Win Count	Draw Count	Loss Count	Points
1	IND	4	3	0	1	9
2	AUS	4	2	0	2	6
3	ENG	4	1	0	3	3

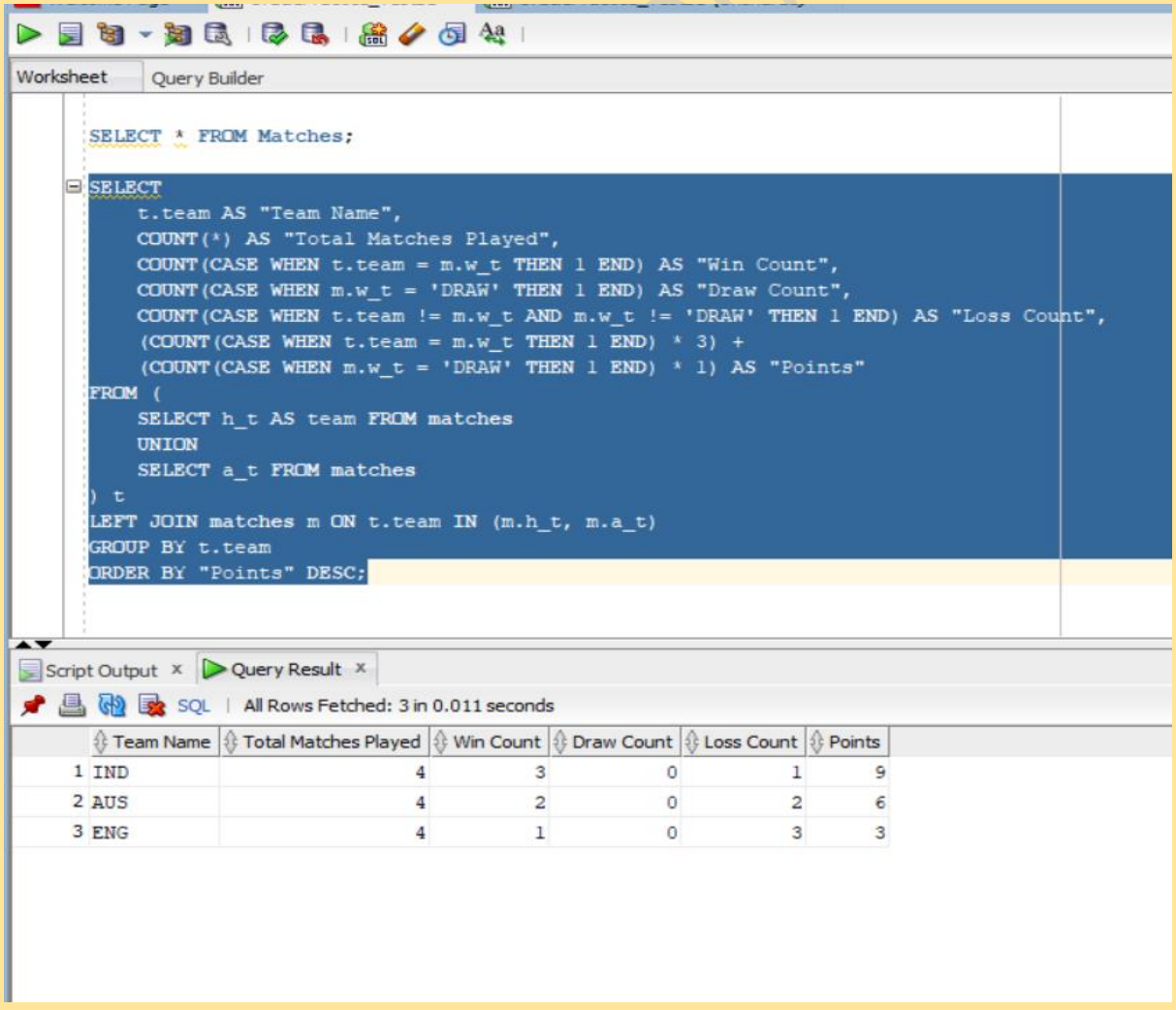
Explanation:

1. **Data Preparation: The UNION ALL combines the Home Team and Away Team into a unified list with corresponding Winner Team.**
2. **Match Count Aggregation: The COUNT(*) gives total matches played by each team.**
3. **Win, Draw, and Loss Calculation: SUM with CASE WHEN handles win and loss counts. The draw count is set to 0 as there are no draws in the data.**
4. **Points Calculation: Points are computed using the formula (Win Count * 3) + (Draw Count * 1) + (Loss Count * 0).**
5. **Sorting by Points: The ORDER BY clause sorts teams by points in descending order.**

Approach Solution-2: Using CASE statements and JOIN to generate the report with properly formatted headings:

Query:

```
SELECT
    t.team AS "Team Name",
    COUNT(*) AS "Total Matches Played",
    COUNT(CASE WHEN t.team = m.w_t THEN 1 END) AS "Win Count",
    COUNT(CASE WHEN m.w_t = 'DRAW' THEN 1 END) AS "Draw Count",
    COUNT(CASE WHEN t.team != m.w_t AND m.w_t != 'DRAW' THEN 1 END) AS "Loss
Count",
    (COUNT(CASE WHEN t.team = m.w_t THEN 1 END) * 3) +
    (COUNT(CASE WHEN m.w_t = 'DRAW' THEN 1 END) * 1) AS "Points"
FROM (
    SELECT h_t AS team FROM matches
    UNION
    SELECT a_t FROM matches
) t
LEFT JOIN matches m ON t.team IN (m.h_t, m.a_t)
GROUP BY t.team
ORDER BY "Points" DESC;
```



,

Explanation:

1. **Team Extraction (t Subquery):**

```
SELECT h_t AS team FROM matches
UNION
SELECT a_t FROM matches
```

1.

- Collects all unique team names from both Home Team and Away Team columns.

2. **Joining with Matches (LEFT JOIN):**

- Each team is joined with all matches where the team participated as either Home Team or Away Team.

3. **Counting Results with CASE:**

- Win Count: COUNT(CASE WHEN t.team = m.w_t THEN 1 END) counts wins.
- Draw Count: COUNT(CASE WHEN m.w_t = 'DRAW' THEN 1 END) handles draw scenarios.
- Loss Count: COUNT(CASE WHEN t.team != m.w_t AND m.w_t != 'DRAW' THEN 1 END) ensures losses are correctly counted without counting draws as losses.

4. **Calculating Points:**

- Points are calculated using (Win * 3) + (Draw * 1) + (Loss * 0), which is simplified in the formula.

5. **Sorting the Results:**

- The output is sorted by "Points" in descending order to rank the teams properly.

Question 8:

SQL Interview Question: Find the Team with the Most Wins

Problem Statement:

You are given a table named matches1 with the following columns:

- **match_id: Unique identifier for each match.**
- **team1: The first team that played the match.**
- **team2: The second team that played the match.**
- **winner: The team that won the match.**

Write a SQL query to find the team that has won the most matches

Input Table:

match_id	team1	team2	winner
1	India	Aus	India
2	Pak	Eng	Pak
3	India	Pak	India
4	Aus	Eng	Aus

Expected Output:

winner	wins
India	2

Table Creation and Sample Data:

```
-- Step 1: Create the 'matches1' table
CREATE TABLE matches1 (
    match_id INT PRIMARY KEY,
    team1 VARCHAR(50),
    team2 VARCHAR(50),
    winner VARCHAR(50)
);

-- Step 2: Insert sample data into the 'matches' table
INSERT INTO matches1 (match_id, team1, team2, winner) VALUES
(1, 'India', 'Aus', 'India'),
(2, 'Pak', 'Eng', 'Pak'),
(3, 'India', 'Pak', 'India'),
(4, 'Aus', 'Eng', 'Aus');
```

Solution Approach 1: Using ROWNUM

```
select * from matches1;

SELECT winner, wins
FROM (
    SELECT winner, COUNT(*) AS wins
    FROM matches1
    GROUP BY winner
    ORDER BY wins DESC
)
WHERE ROWNUM = 1;
```

Script Output | Query Result

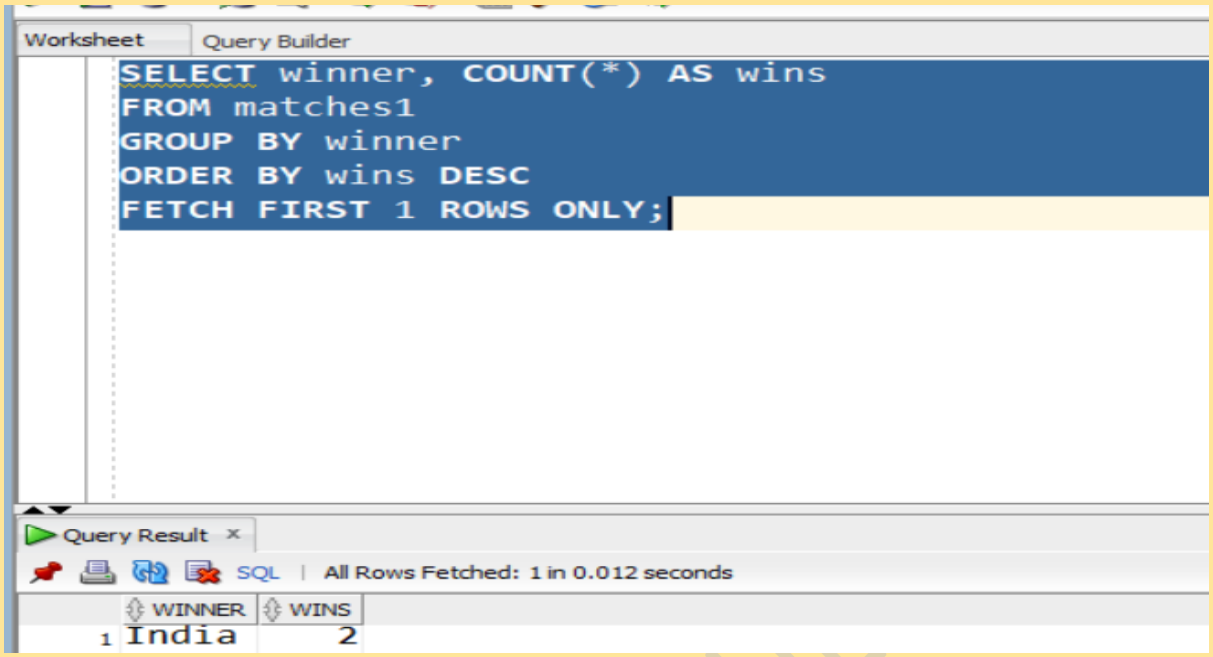
SQL | All Rows Fetched: 1 in 0.006 seconds

	WINNER	WINS
1	India	2

Explanation:

1. **Using Subquery with ROWNUM: The subquery is used to first sort the results by wins in descending order. The ROWNUM = 1 condition then fetches only the top result.**

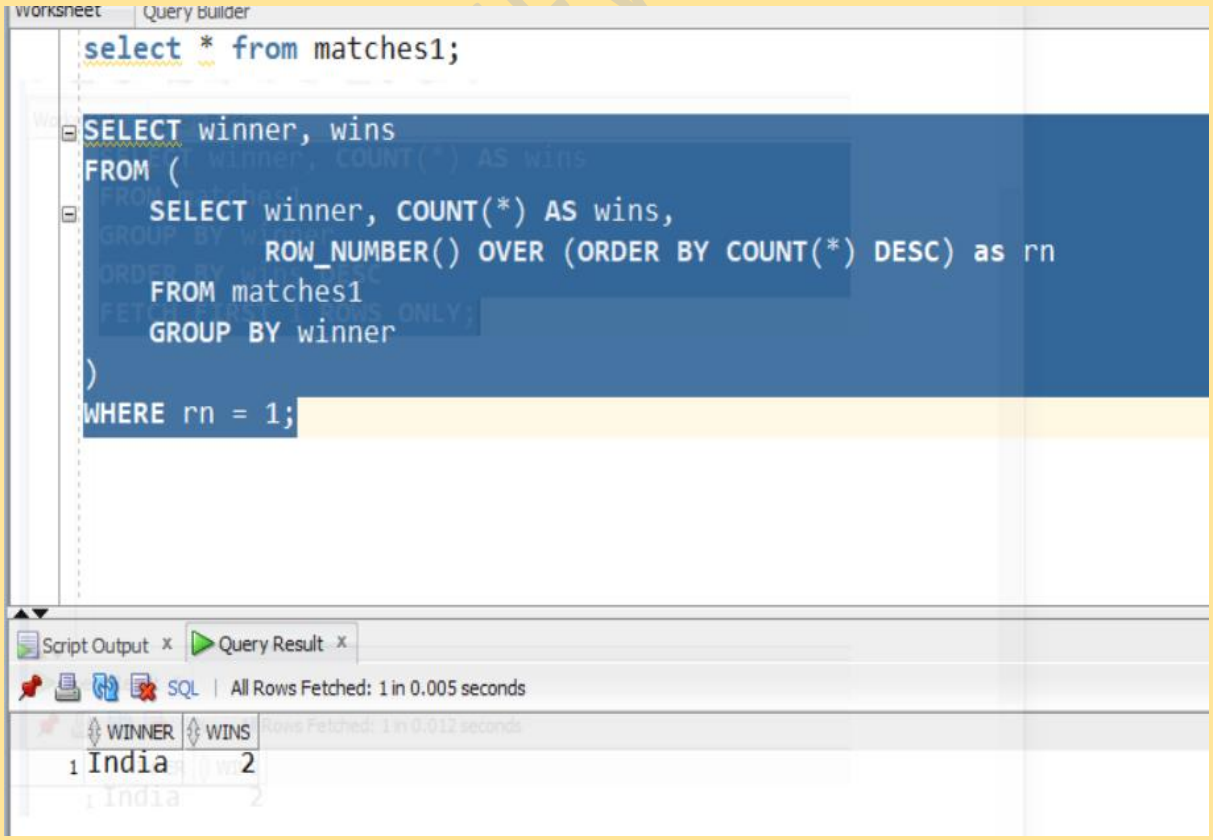
✅ Alternative Solution Approach Using FETCH FIRST:



Explanation:

FETCH FIRST Syntax: This is a more modern approach and is easier to understand. It directly limits the result set to one row.

✅ Using Solution Approach ROW_NUMBER () with ORDER BY:



Explanation:
ROW_NUMBER() Window Function: The ROW_NUMBER() function assigns a unique rank to rows ordered by wins. You can then filter to keep only the top-ranked row.

SQL Interview Question 9: Find the Top 3 Batsmen with the Most Runs

Problem Statement:
You are given a table named players with the following columns:

- **player_id: Unique identifier for each player.**
- **name: Name of the player.**
- **runs: The total runs scored by the player.**

Write a SQL query to find the top 3 batsmen with the most runs.

Input Table:

player_id	name	runs
1	Kohli	1000
2	Rohit	900
3	Dhoni	800
4	Raina	700

Expected Output:

name	runs
Kohli	1000
Rohit	900
Dhoni	800

1. Table Creation and Sample Data:

```
-- Step 1: Create the 'players' table
CREATE TABLE players (
    player_id INT PRIMARY KEY,
    name VARCHAR(50),
    runs INT
);

-- Step 2: Insert sample data into the 'players' table
INSERT INTO players (player_id, name, runs) VALUES
(1, 'Kohli', 1000),
(2, 'Rohit', 900),
(3, 'Dhoni', 800),
(4, 'Raina', 700);
```

Oracle SQL Query Solutions:

✅ **Approach 1: Using ROWNUM**

```
select * from players;

SELECT name, runs
FROM (
     SELECT name, runs
     FROM players
     ORDER BY runs DESC
)
WHERE ROWNUM <= 3;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.01 seconds

	NAME	RUNS
1	Kohli	1000
2	Rohit	900
3	Dhoni	800

Explanation:

1. **Using ROWNUM:**
   - **The subquery sorts the players by runs in descending order.**
   - **The ROWNUM <= 3 filters the top 3 rows.**

✅ **Solution Approach 2: Using FETCH FIRST 3 ROWS ONLY (Oracle 12c and Later)**

```
select * from players;

SELECT name, runs
FROM players
ORDER BY runs DESC
FETCH FIRST 3 ROWS ONLY;
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.001 seconds

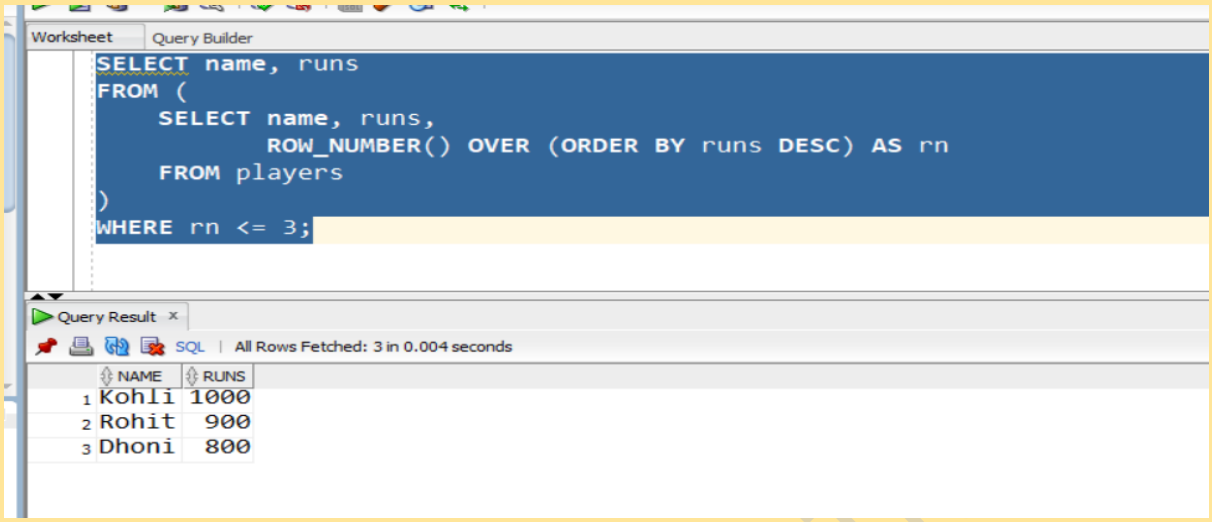
	NAME	RUNS
1	Kohli	1000
2	Rohit	900
3	Dhoni	800

Explanation:

Using FETCH FIRST:

- Available in **Oracle 12c** and later.
- Simplifies limiting the result set directly after sorting.

✅ Solution Approach 3: Using ROW_NUMBER() Window Function



Explanation:

Using ROW_NUMBER():

- Assigns a rank to each row based on the runs.
- The WHERE rn <= 3 condition filters the top 3 ranks.

SQL Interview Question 10 : Find the Team with the Most Losses

Problem Statement:

You have a table named matches1 with the following columns:

- **match_id: Unique identifier for each match.**
- **team1: The first team that played the match.**
- **team2: The second team that played the match.**
- **winner: The team that won the match.**

Write a SQL query to find the team that has lost the most matches.

Input Table:

match_id	team1	team2	winner
1	India	Aus	India
2	Pak	Eng	Pak
3	India	Pak	India
4	Aus	Eng	Aus

Expected Output:

team	losses
Eng	2

Sample table with Data:

```
select * from matches1;
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0.002 seconds

	MATCH_ID	TEAM1	TEAM2	WINNER
1	1	India	Aus	India
2	2	Pak	Eng	Pak
3	3	India	Pak	India
4	4	Aus	Eng	Aus

Oracle SQL Query Solution:

Worksheet | Query Builder

```
SELECT team, COUNT(*) AS losses
FROM (
    SELECT team1 AS team FROM matches1 WHERE team1 != winner
    UNION ALL
    SELECT team2 AS team FROM matches1 WHERE team2 != winner
) t
GROUP BY team
ORDER BY losses DESC
FETCH FIRST 1 ROW ONLY;
```

Query Result

SQL | All Rows Fetched: 1 in 0.007 seconds

	TEAM	LOSSES
1	Eng	2

Explanation:

1. **Identify Losing Teams:**
   The WHERE team1 != winner and WHERE team2 != winner conditions filter out winning teams, leaving only the losers.
2. **Combine Results with UNION ALL:**
   This ensures both teams from all matches are considered for losses.
3. **Count Losses:**
   The GROUP BY team groups the losses by team, and COUNT(*) AS losses counts them.
4. **Sort and Fetch the Top Result:**
   Using ORDER BY losses DESC to sort teams by losses in descending order, followed by FETCH FIRST 1 ROW ONLY to display only the team with the most losses.

Alternative Solution Using ROWNUM:

if your Oracle version does not support FETCH FIRST, you can use ROWNUM:

```
SELECT team, losses
FROM (
    SELECT team, COUNT(*) AS losses
    FROM (
        SELECT team1 AS team FROM matches1 WHERE team1 != winner
        UNION ALL
        SELECT team2 AS team FROM matches1 WHERE team2 != winner
    ) t
    GROUP BY team
    ORDER BY losses DESC
)
WHERE ROWNUM = 1;
```

Query Result | SQL | All Rows Fetched: 1 in 0.005 seconds

	TEAM	LOSSES
1	Eng	2

Solution Approach 3 : Using CASE WHEN and Aggregation

```
SELECT team, COUNT(*) AS losses
FROM (
    SELECT team1 AS team,
           CASE WHEN winner != team1 THEN 1 END AS loss
    FROM matches1
    UNION ALL
    SELECT team2 AS team,
           CASE WHEN winner != team2 THEN 1 END AS loss
    FROM matches1
) t
WHERE loss = 1
GROUP BY team
ORDER BY losses DESC
FETCH FIRST 1 ROW ONLY;
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.009 seconds

	TEAM	LOSSES
1	Eng	2

Explanation:

1. **Selecting Potential Losing Teams:**
   - **The CASE WHEN statement assigns a 1 to the loss column when the team is not the winner.**
2. **Combining Teams with UNION ALL:**
   - **The team1 and team2 columns are combined into a single team column.**
3. **Filtering Only Losses:**
   - **The WHERE loss = 1 condition keeps only the rows where the team has actually lost.**
4. **Counting and Sorting:**
   - **GROUP BY team aggregates the loss count per team.**
   - **ORDER BY losses DESC sorts teams by losses in descending order.**

5. **Limiting the Output:**
    - **The FETCH FIRST 1 ROW ONLY ensures only the team with the most losses is displayed.**

Solution Approach 4 : Using Analytic Function ROW_NUMBER()

```
select * from matches1;

SELECT team, losses FROM (
    SELECT team, COUNT(*) AS losses,
           ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) AS rn
    FROM (
        SELECT team1 AS team FROM matches1 WHERE team1 != winner
        UNION ALL
        SELECT team2 AS team FROM matches1 WHERE team2 != winner
    ) t
    GROUP BY team
)
WHERE rn = 1;
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.006 seconds

	TEAM	LOSSES
1	Eng	2

Explanation:

1. **Identify Losing Teams:**
   Using WHERE team1 != winner and WHERE team2 != winner to filter losses.
2. **Counting Losses:**
   The COUNT(*) AS losses aggregates the loss count per team.
3. **Using ROW_NUMBER() for Ranking:**
   The ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) ranks teams by losses.
4. **Selecting the Top Team:**
   The WHERE rn = 1 filters the team with the highest loss rank.

SQL Interview Question 11: Find the Team with the Most Wins Against a Specific Team (India)

Problem Statement:

You have a table named matches with the following columns:

- **match_id: Unique identifier for each match.**
- **team1: The first team that played the match.**
- **team2: The second team that played the match.**
- **winner: The team that won the match.**

Write a SQL query to find the team that has won the most matches against a specific team, such as India.

Input Table:

match_id	team1	team2	winner
1	India	Aus	India
2	Pak	India	Pak
3	India	Pak	India
4	Aus	India	Aus

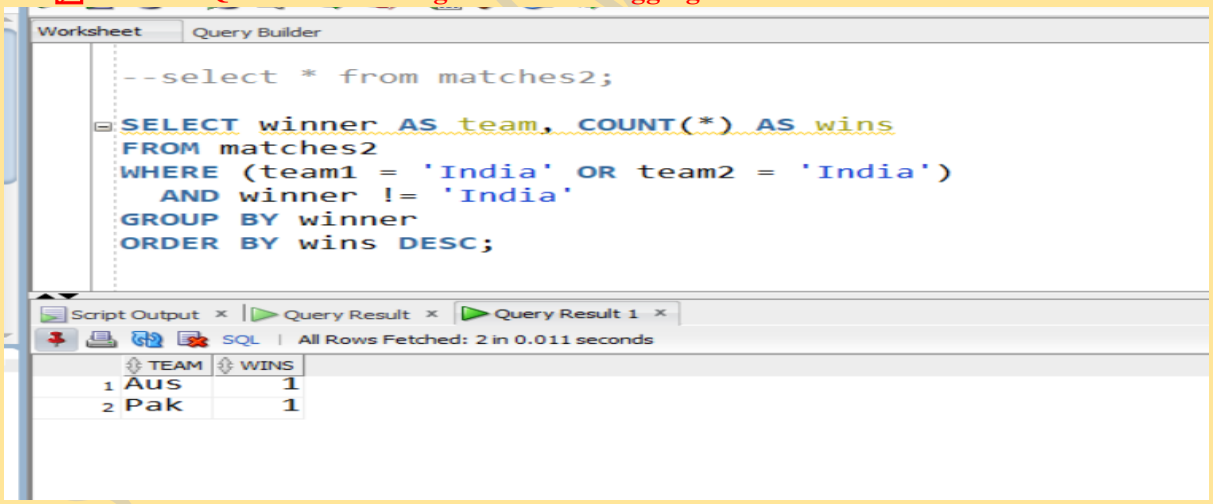
Expected Output:

team	wins
Aus	1
Pak	1

```
-- Create the 'matches2' table
CREATE TABLE matches2 (
    match_id INT PRIMARY KEY,
    team1 VARCHAR2(50),
    team2 VARCHAR2(50),
    winner VARCHAR2(50)
);

-- Insert sample data into the 'matches2' table INSERT INTO matches (match_id, team1, team2, winner) VALUES
 (1, 'India', 'Aus', 'India'),
(2, 'Pak', 'India', 'Pak'),
(3, 'India', 'Pak', 'India'),
(4, 'Aus', 'India', 'Aus');
-- Commit the changes to save the data
 COMMIT;
```

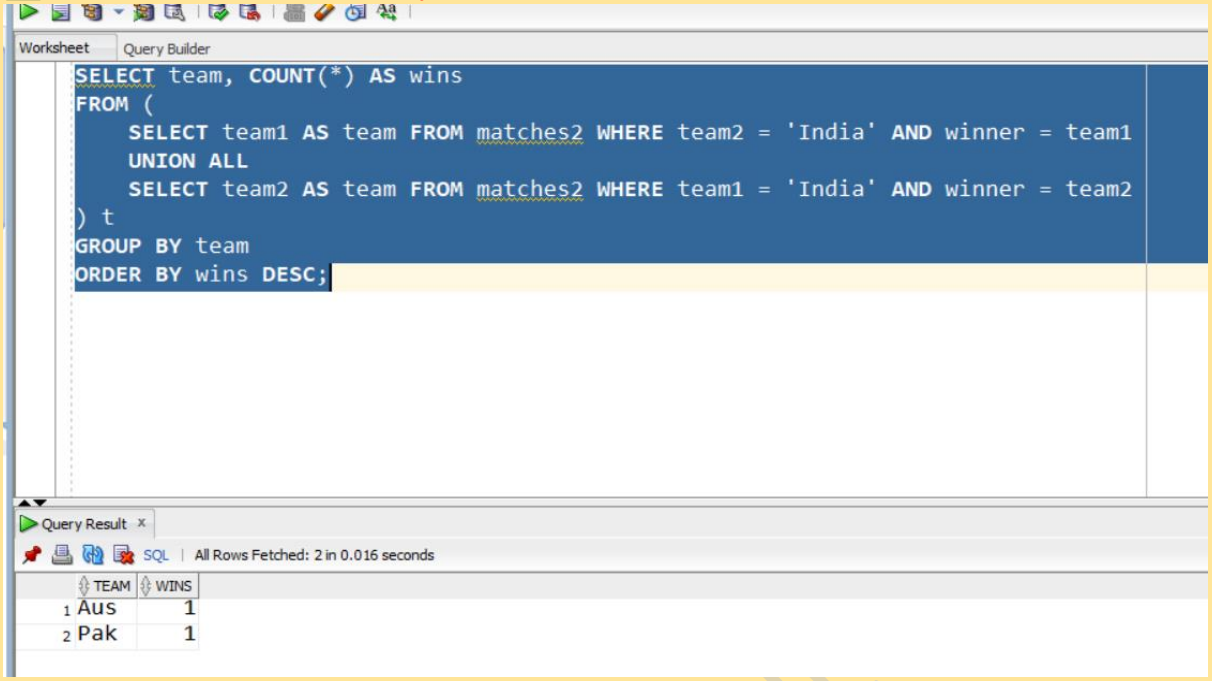
Oracle SQL Solution: Using Conditional Aggregation



Explanation:

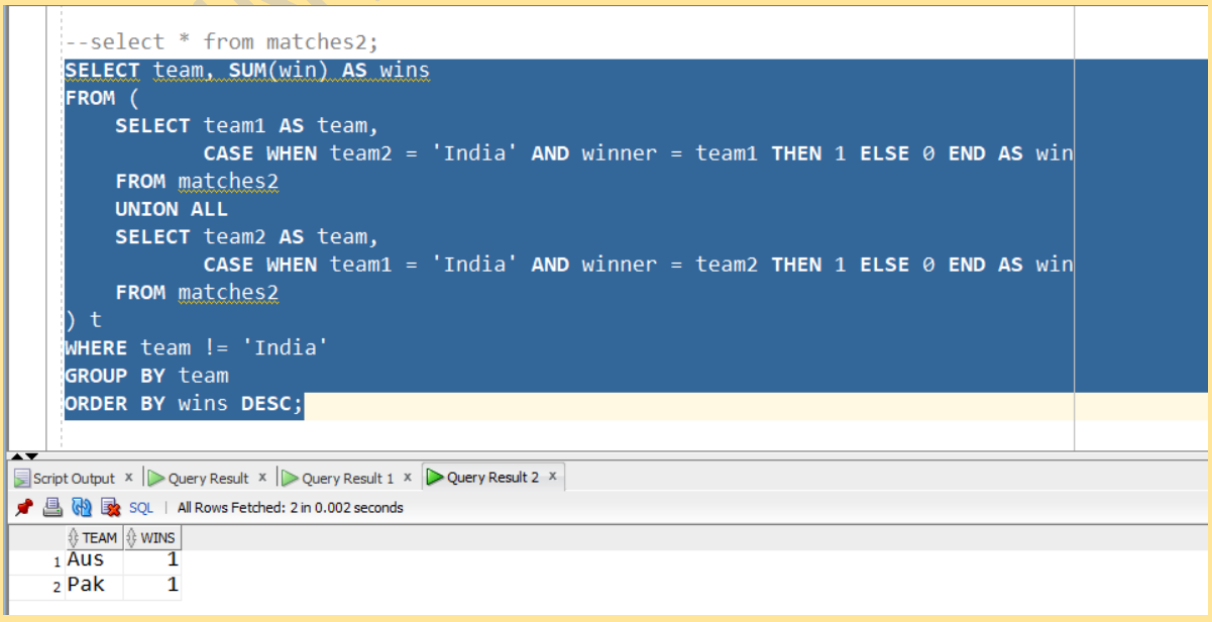
1. **Filter Matches Against India:**
   The WHERE (team1 = 'India' OR team2 = 'India') condition selects only matches where India participated.
2. **Exclude India's Wins:**
   The AND winner != 'India' ensures only matches where India lost are considered.
3. **Count Wins for Each Opponent:**
   The GROUP BY winner groups the wins by the winning team.
4. **Order the Results:**
   The ORDER BY wins DESC sorts the teams by win count in descending order, displaying all teams with their win counts.

✅ Alternative Oracle SQL Query:



📝 Explanation:

1. **Split Matches into Two Scenarios:**
   - **When India is team2, the opponent team1 must be the winner.**
   - **When India is team1, the opponent team2 must be the winner.**
2. **Combine Results with UNION ALL:**
   - **The UNION ALL is used to gather all possible wins against India.**
3. **Count Wins by Team:**
   - **The GROUP BY team groups wins by each team, and COUNT(*) AS wins aggregates the win count.**
4. **Sort the Results:**
   - **The ORDER BY wins DESC sorts the teams by the number of wins against India.**
5. **🔍 Another Approach: Using CASE WHEN with Aggregation**



Explanation:

1. **Conditional Win Assignment:**
   - **When India is team2 and team1 is the winner, the win is assigned to team1.**
   - **When India is team1 and team2 is the winner, the win is assigned to team2.**
2. **Use of CASE WHEN:**
   - **The CASE WHEN logic provides a 1 for wins and 0 otherwise.**
3. **Summing Up Wins:**
   - **The SUM(win) AS wins aggregates the wins by each team.**
4. **Exclude India as a Winner:**
   - **The WHERE team != 'India' condition filters out India as a possible winning team.**
5. **Sorting Results:**
   - **The ORDER BY wins DESC displays teams with the most wins at the top.**