DOCUMENTATION FOR ORACLE SQL

11-Feb-2022

from

KPI FRESHER'S TRAINING BATCH 3

under the guidance of

Mr. HARSHA MAHATI

CREATE TABLE

To create a new table in the database, Oracle provides the Oracle CREATE TABLE statement.

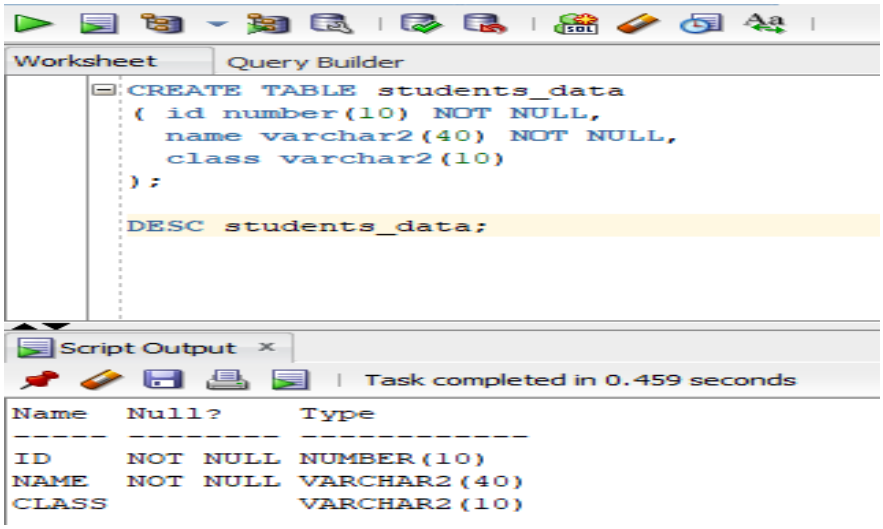
Syntax:

```
CREATE TABLE table_name (
        Col1 data_type col_constraint,
        Col2 data_type col_constraint,
        Col3 data_type col_constraint,
        …. Coln data_type  col_constraint );
```

table_name: It is used to specify the table name.

Column Definition: Column1, column2, … column n is used to specify the name of the multiple columns which you want to add in the table. The Oracle database does not allow a total number of columns more than 32. A datatype is a must for each column. The data type of a column can be NUMBER, VARCHAR2, etc. The column constraint mainly defines each column as “NULL” or “NOT NULL”, with the value of “NULL” as default. Some other common column constraints are Primary Key, Check and Foreign Key.

Example 1: Creating a table with NULL and NOT NULL table constraint.



Explanation:

Column_1:

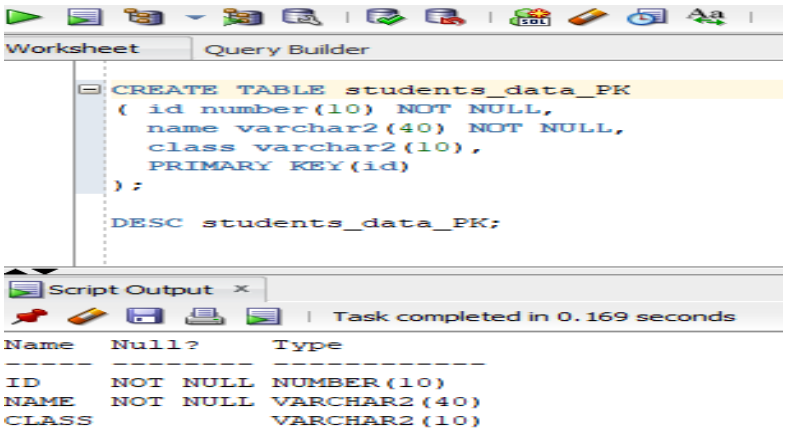
id: It is the name of the first column.
number: It is the datatype of the first column which also specifies a maximum limit of 10 digits in length for the “id”.
NOT NULL: It defines the column constraint of the first column and thus Column 1 cannot contain null values.

Column_2:

name: It is the name of the second column.
varchar: It is the datatype of the second column which also specifies a maximum limit of 40 characters in length for the “name”.
NOT NULL: It defines the column constraint of the second column and thus Column 2 cannot contain null values.

Column_3:
class: It is the name of the third column.
varchar: It is the datatype of the third column which also specifies a maximum limit of 10 characters in length for the “class”.
NULL: There is no column constraint defined for the third column, thus it is NULL by default and thus Column 3 can contain null values.

Example 2: Creating a table with a PRIMARY KEY table constraint.



Explanation:

All the three columns and their definitions are the same as that in example 1. The difference is in the existence of Primary Key in the second example.

The PRIMARY KEY clause is a field or combination of fields which specifies a column as the primary key column. The primary key column is used for distinguishing a unique row in a table. In Oracle, a table can hold only one primary key, and every field of the primary key must contain NOT NULL values.

In the above example, “id” is defined as the Primary Key Column.

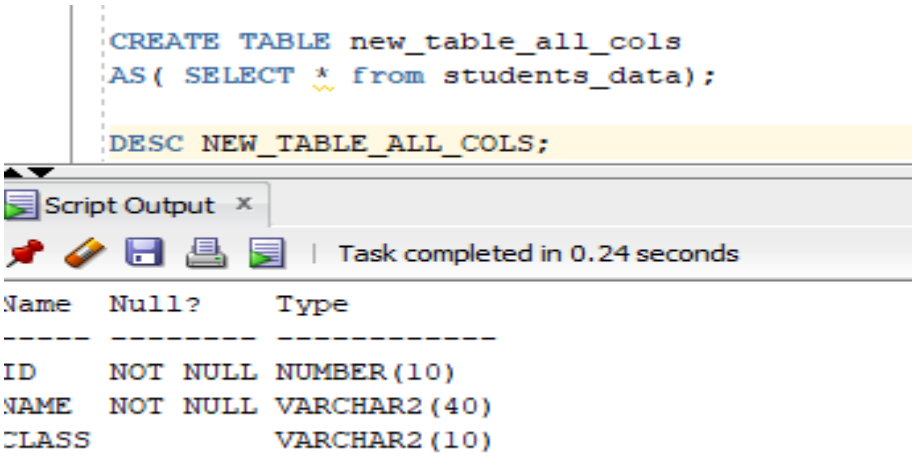
CREATE TABLE AS

To create a table from an existent table, the CREATE TABLE AS statement is used. It copies the columns of the existing table to the new table.

Syntax: For creating a table by copying all columns of another table.

```
CREATE TABLE current_table
AS (SELECT * FROM existing_table);
```

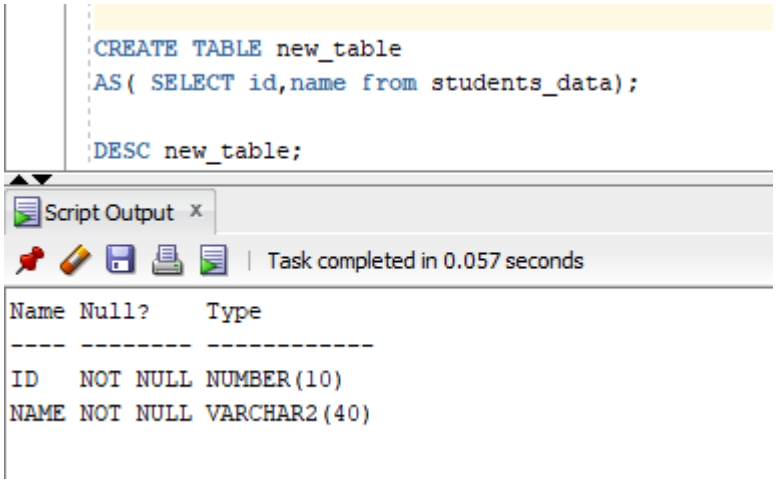
Example1: Creating a table by copying all columns of another table.



Syntax: For creating a table by copying selected columns of another table.

```
CREATE TABLE new_table
  AS (SELECT column_1, column_2, ... column_n
     FROM existing_table);
```

Example 2: Creating a table by copying selected columns of another table.



Syntax: For creating a table by copying selected columns from multiple tables

```
CREATE TABLE new_table
AS (SELECT column_1, column_2, ... column_n
    FROM existing_table1, existing_table2, ... existing_tablen);
```

Example 3: Creating a table by copying selected columns from multiple tables.

```
CREATE TABLE  "hindi_students"
   (    "HSTUDENTS_ID" NUMBER(10,0) NOT NULL ENABLE,
    "HSTUDENTS_NAME" VARCHAR2(40) NOT NULL ENABLE,
    "HSTUDENTS_CLASS" VARCHAR2(10)
   )

desc "hindi_students";
CREATE TABLE "english_students"
( "ESTUDENTS_ID" NUMBER(10,0) NOT NULL ENABLE,
"ESTUDENTS_NAME" VARCHAR2(40) NOT NULL ENABLE,
"ESTUDENTS_CLASS" VARCHAR2(10)
)

desc "english_students";

CREATE TABLE new_students_hindi_english
AS (SELECT "hindi_students".HSTUDENTS_ID, "hindi_students".HSTUDENTS_CLASS,
"english_students".ESTUDENTS_NAME
FROM "hindi_students","english_students");

desc new_students_hindi_english;
```

Script Output

Task completed in 0.05 seconds

```
Name              Null?     Type
----------------- --------- ------------
HSTUDENTS_ID      NOT NULL  NUMBER(10)
HSTUDENTS_NAME    NOT NULL  VARCHAR2(40)
HSTUDENTS_CLASS             VARCHAR2(10)
Name              Null?     Type
----------------- --------- ------------
ESTUDENTS_ID      NOT NULL  NUMBER(10)
ESTUDENTS_NAME    NOT NULL  VARCHAR2(40)
ESTUDENTS_CLASS             VARCHAR2(10)
Name              Null?     Type
----------------- --------- ------------
HSTUDENTS_ID      NOT NULL  NUMBER(10)
HSTUDENTS_CLASS             VARCHAR2(10)
ESTUDENTS_NAME    NOT NULL  VARCHAR2(40)
```

ALTER TABLE

To add, modify, drop or delete columns in a table ALTER TABLE statement is used. Along with all these, it is also used to rename a table. The ALTER TABLE statement allow the users to Add one or more columns, to Modify column definition, to Drop one or more columns, to Rename columns and to Rename a table.

Syntax:

```
ALTER TABLE table_name action;
```

ALTER TABLE ADD column

Syntax: To add a column in the existing table.

```
ALTER TABLE table_name
ADD column_name data_type constraint;
```

Example 1: Adding a new column to an already existing table.

```
ALTER TABLE students_data
ADD student_age number;

desc students_data;
```

Script Output

Task completed in 0.066 seconds

```
Table STUDENTS_DATA altered.

Name          Null?     Type
------------ -------- ------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(40)
CLASS                  VARCHAR2(10)
STUDENT_AGE            NUMBER
```

Syntax: To add multiple columns in the existing table.

```
ALTER TABLE table_name
  ADD (column_1 column_definition,
      column_2 column_definition,
      ...
      column_n column_definition);
```

Example 2: Adding multiple columns to an already existing table.

```
ALTER TABLE students_data
ADD std_rn number
ADD std_contact number;

desc students_data;
```

Script Output | Task completed in 0.082 seconds

```
Name          Null?     Type
------------ -------- ------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(40)
CLASS                  VARCHAR2(10)
STUDENT_AGE            NUMBER
STD_RN                 NUMBER
STD_CONTACT            NUMBER
```

ALTER TABLE MODIFY column

Syntax: To modify a single column of a table.

```
ALTER TABLE table_name
MODIFY column_name action;
```

Example 1: Modifying a single column of a table.

```
ALTER TABLE students_data
MODIFY name varchar2(50);

desc students_data;
```

Script Output | Task completed in 0.067 seconds

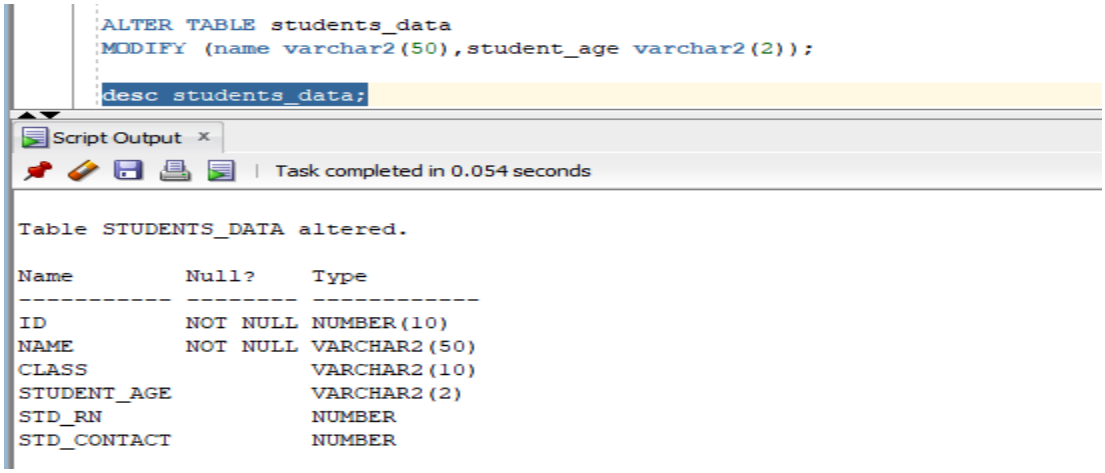
```
Table STUDENTS_DATA altered.

Name          Null?     Type
------------ -------- ------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(50)
CLASS                  VARCHAR2(10)
STUDENT_AGE            NUMBER
STD_RN                 NUMBER
STD_CONTACT            NUMBER
```

Syntax: To modify multiple columns of a table.

```
ALTER TABLE table_name
MODIFY (
    column_name_1 action,
    column_name_2 action,
    ...
);
```

Example 2: Modifying multiple columns of a table.



```
ALTER TABLE students_data
MODIFY (name varchar2(50),student_age varchar2(2));

desc students_data;
```

Script Output | Task completed in 0.054 seconds

```
Table STUDENTS_DATA altered.

Name          Null?    Type
------------- -------- -------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(50)
CLASS                  VARCHAR2(10)
STUDENT_AGE            VARCHAR2(2)
STD_RN                 NUMBER
STD_CONTACT            NUMBER
```

ALTER TABLE DROP COLUMN

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

ALTER TABLE RENAME COLUMN

Syntax:

```
ALTER TABLE table_name
RENAME COLUMN existing_column_name to new_column_name;
```

Example :

```
ALTER TABLE students_data
RENAME COLUMN student_age to std_age;

desc students_data;
```

Script Output

Task completed in 0.064 seconds

```
Table STUDENTS_DATA altered.

Name          Null?    Type
------------ -------- ------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(50)
CLASS                  VARCHAR2(10)
STD_AGE                VARCHAR2(2)
STD_RN                 NUMBER
STD_CONTACT            NUMBER
```

ALTER TABLE RENAME TO

Syntax:

```
ALTER TABLE existing_table_name
RENAME TO new_name;
```

Example :

```
ALTER TABLE students_data
RENAME TO children;

desc children;
```

Script Output | Task completed in 0.239 seconds

```
Table STUDENTS_DATA altered.

Name          Null?     Type
------------ --------- -------------
ID            NOT NULL NUMBER(10)
NAME          NOT NULL VARCHAR2(50)
CLASS                  VARCHAR2(10)
STD_AGE                VARCHAR2(2)
STD_RN                 NUMBER
STD_CONTACT            NUMBER
```

DROP TABLE:

To move a table to recycle bin or to completely delete a table from the Oracle database, Oracle DROP TABLE statement is used.

Syntax:

DROP TABLE table_name;

table_name: It is used to specify the name of the table to be eliminated from the Oracle database.

Example 1: Remove a table.

```
drop table students_data_PK;
```

Script Output | Task completed in 0.051 seconds

```
Table STUDENTS_DATA_PK dropped.
```

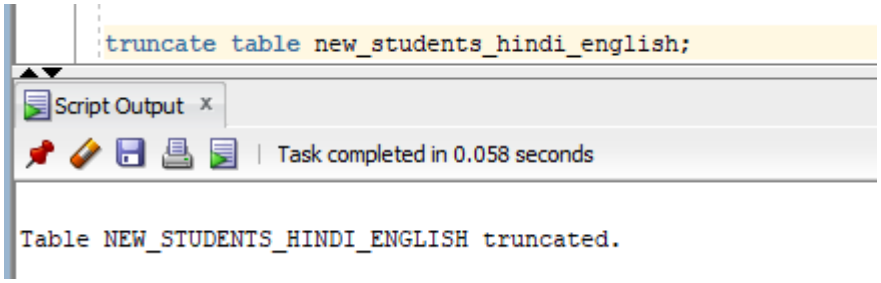
DROP Multiple Tables:

To Drop multiple tables at once, there is no direct way hat Oracle facilitates. However, a User can create a PL/SQL block to Drop multiple tables at once.

TRUNCATE TABLE

To truncate or remove records of a table but not the structure, the Oracle TRUNCATE TABLE statement is used. It is much similar to the DELETE statement but the TRUNCATE TABLE statement cannot be rolled back, which is possible in the case of the DELETE statement. Also, this statement does not affect indexes, triggers or dependencies of the table to be truncated.

Syntax: TRUNCATE TABLE table_name;



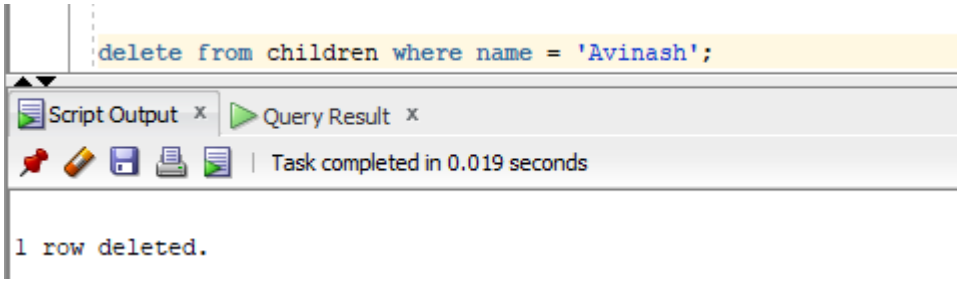
Oracle Delete Query:

Uses: To delete records of a table.

Syntax:

DELETE from table_name WHERE conditions;

Example:



ORACLE QUERIES

Select, Insert, Update, Delete, Drop, Alter table, and Create are the few queries among many that can be executed in an Oracle database.

Oracle Select Query:

Uses: To fetch records from the database.

Syntax: To select records from a table.

```
SELECT * from table_name;
```

```
select * from children;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.001 seconds

	ID	NAME	CLASS	STD_AGE	STD_RN	STD_CONTACT
1	101	Avinash	MCA	25	1219	7507504270

Oracle Insert Query:

Uses: To insert records into a table.

Syntax:

```
INSERT into table_name VALUES(value1, value2, .. valuen);
```

```
insert into children values(101,'Avinash','MCA',25,1219,7507504270);
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.001 seconds

	ID	NAME	CLASS	STD_AGE	STD_RN	STD_CONTACT
1	101	Avinash	MCA	25	1219	7507504270

Oracle Update Query:

Uses: To update records of a table.

Syntax:

UPDATE table_name SET column_name1 = value1, column_name2 = value2

WHERE conditions;

```
update children set std_age = 26 where name = 'Avinash';
select * from children;
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.001 seconds

	ID	NAME	CLASS	STD_AGE	STD_RN	STD_CONTACT
1	101	Avinash	MCA	26	1219	7507504270

ORACLE INSERT ALL

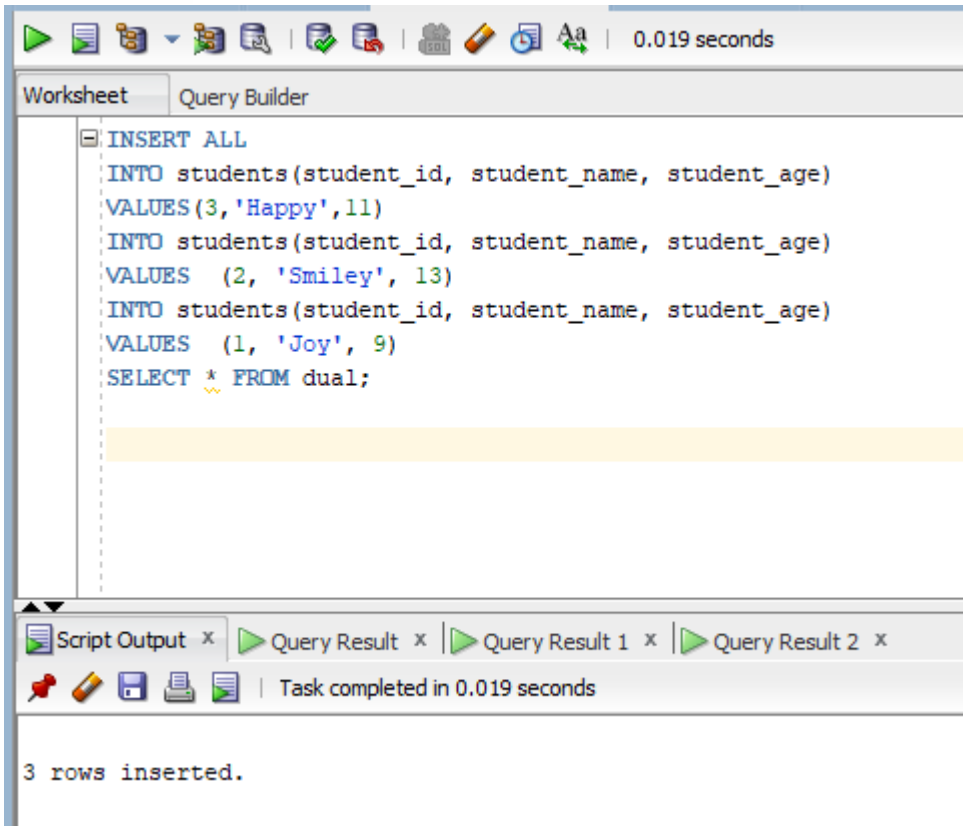
To insert multiple rows, the Oracle INSERT ALL statement is used. A single INSERT statement, as a SQL command thus can insert the rows into one table or multiple tables.

Syntax:

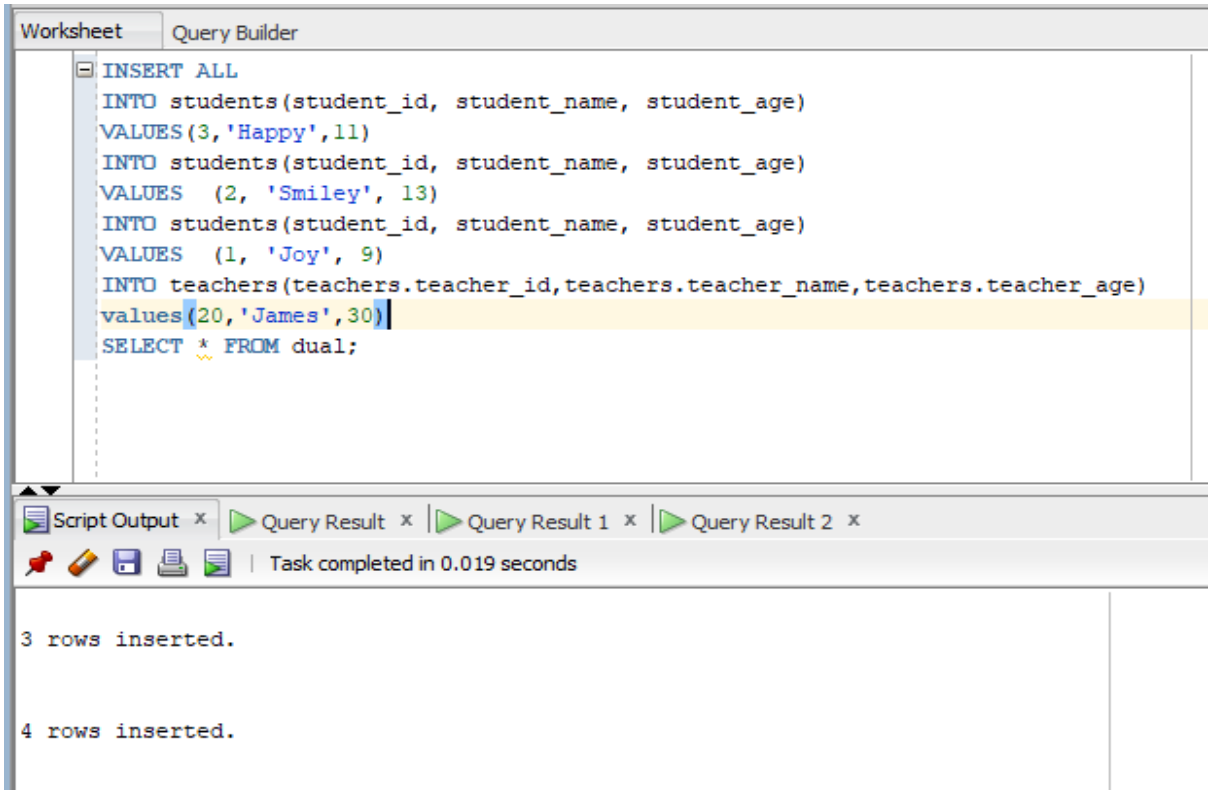
```
INSERT ALL
INTO table_name (column_1, column_2, column_n)
VALUES (expression_1, expression_2, ... expression_n)
INTO table_name(column_1, column_2, column_n)
VALUES (expression_1, expression_2, ... expression_n)
INTO table_name (column_1, column_2, column_n)
VALUES (expression_1, expression_2, ... expression_n)
Subquery;
```

Output:

Example: Inserting multiple rows into a single table.



Example: Inserting multiple rows into multiple tables.



GROUP BY CLAUSE:

- This clause is used with select statement and used to collect the data from multiple records.
- Group the results by one or more columns.
- This are often used with aggregate functions like COUNT(), SUM(), AVG(), MAX(), MIN().

Syntax:

```
SELECT expressions
FROM table_name
GROUP BY columns;
```

Parameters:

expressions: It is use to specify the columns or calculations to be retrieved.

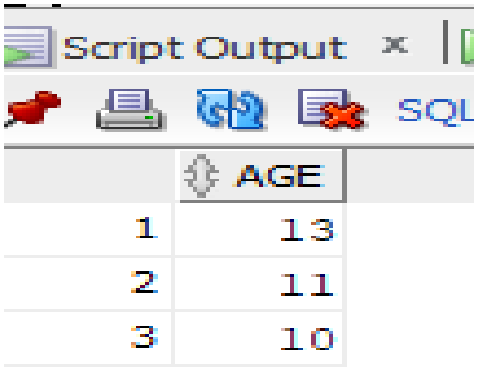
table_name: Used to specify name of table.

columns: Specify the list of columns to be grouped.

Query:

```
SELECT student_age AS age
FROM student
GROUP BY student_age;
```

Output:



GROUP BY clause with WHERE clause:

Query:

```
SELECT student_age
FROM student
WHERE student_age > 10
GROUP BY student_age;
```

Output:

Script Output | Query

SQL | All Rc

	STUDENT_AGE
1	13
2	11

GROUP BY with aggregate functions:

Query:

```
SELECT COUNT(student_id), student_name
FROM student
GROUP BY student_name;
```

Output:

Script Output | Query Result 1

SQL | All Rows Fetched: 5 in 0.00

	COUNT(STUDENT_ID)	STUDENT_NAME
1	1	happy
2	1	bond
3	1	smiley
4	1	joy
5	1	james

HAVING CLAUSE:

- Having clause is used with GROUP BY Clause.
- Return groups of row only when condition is TRUE.

Syntax:

SELECT expressions
FROM table_name
GROUP BY columns,
HAVING conditions;

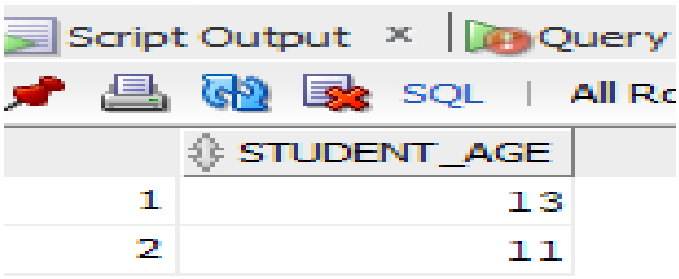
Parameters:

expressions: It is use to specify the columns or calculations to be retrieved.
table_name: Used to specify name of table.
columns: Specify the list of columns to be grouped.
having_conditions: It is used to specify the conditions that are applicable only to the result obtained after successful execution of the GROUP BY command, to restrict the groups of returned rows.

Query:

```
SELECT student_age FROM student
GROUP BY student_age
HAVING student_age > 10;
```

Output:

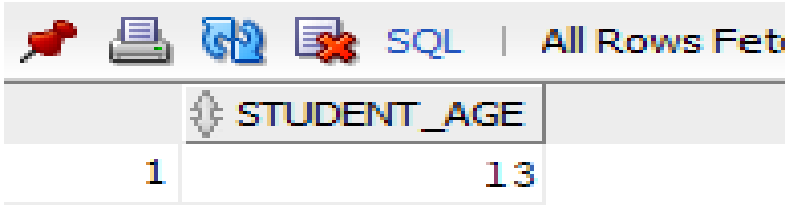


HAVING with WHERE clause:

Query:

```
SELECT student_age
FROM student
WHERE   student_id = 2
GROUP BY student_age
HAVING student_age > 10;
```

Output:

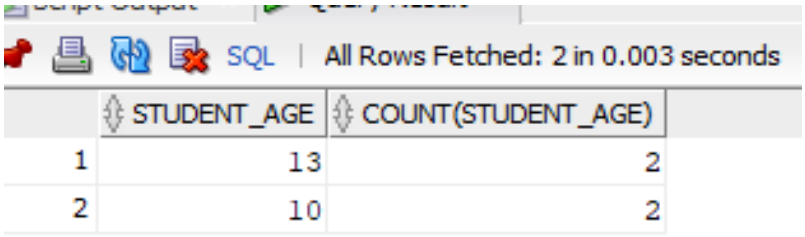


HAVING with aggregate function:

Query:

```
SELECT student_age, COUNT(student_age)
FROM student
GROUP BY student_age
HAVING COUNT(student_age) > 1;
```

Output:



	STUDENT_AGE	COUNT(STUDENT_AGE)
1	13	2
2	10	2

ORDER BY clause:

- Used to sort or re-arrange the records in the result set.
- The ORDER BY keyword sorts the records in ascending order by default.
- To sort the records in descending order, use keyword DESC keyword.

Syntax:

```
SELECT expressions
FROM table_name
WHERE conditions
ORDER BY expression [ASC | DESC];
```

Parameters:

expressions: It is use to specify the columns or calculations to be retrieved.

table_name: Used to specify name of table.

conditions: It is used to specify the conditions to be strictly followed for selection.

ASC: Sort the records in ascending order.

DESC: Sort the records in descending order.

Ascending Order:

Query:

```
SELECT student_name, student_age
FROM student
WHERE student_age > 10
ORDER BY student_name;
```

Output:

	STUDENT_NAME	STUDENT_AGE
1	happy	11
2	james	13
3	smiley	13

Descending Order:

Query:

```
SELECT student_name, student_age
FROM student
WHERE student_age > 10
ORDER BY student_name DESC;
```

Output:

	STUDENT_NAME	STUDENT_AGE
1	smiley	13
2	james	13
3	happy	11

Minus:

To return all the rows of the first SELECT statement which are not common to the rows of the second SELECT statement, the Oracle MINUS operator is used. After the subtraction process, the MINUS operator returns the uncommon or unique records from the corresponding rows of the first SELECT statement. There are however two mandatory conditions for using the MINUS operator in Oracle.

*Each SELECT statement must have the same number of expressions.

*Each corresponding expression in the different SELECT statement should be of the same data type.

Output:

```
SELECT student_name
FROM student
minus
SELECT teacher_name
FROM teacher;
```

Query Result

SQL | All Rows Fetched: 3 in 0.172 seconds

	STUDENT_NAME
1	Happy
2	Joy
3	Smiley

Intersect:

To compare the rows of two or more Oracle SELECT statements, the Oracle INTERSECT operator is used. After the comparing process, the INTERSECT operator returns the common or intersecting records from the corresponding columns of the selected expressions.

There are however two mandatory conditions for using the INTERSECT operator in Oracle.

*Each SELECT statement must have the same number of expressions.

*Each corresponding expression in the different SELECT statement should be of the same data type.

Output:

```
SELECT student_name
FROM student
INTERSECT
SELECT teacher_name
FROM teacher;
```

Script Output | Query Result | SQL | All Rows Fetched: 2 in 0.002 seconds

	STUDENT_NAME
1	Bond
2	James

View:

The view is a virtual table in Oracle with no physical existence as such and thus it does not store any data. The view is saved in the data dictionary of the Oracle and when called, it can be simply executed.

View creation:

```
CREATE VIEW stu_teach AS
SELECT student.student_id, student.student_name, teacher.teacher_id,teacher.teacher_name
FROM student
INNER JOIN teacher
ON student.student_id = student_id
WHERE  student.student_name = 'joy';
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Task completed in 0.017 seconds

```
View STU_TEACH created.
```

Update view:

```
CREATE or replace VIEW stu_teach AS
SELECT student.student_id, student.student_name, teacher.teacher_id,teacher.teacher_name,student.student_age,teacher.teacher_age
FROM student
INNER JOIN teacher
ON student.student_id = student_id
WHERE  student.student_name = 'joy';
```

Script Output | Query Result | Query Result 1 | Query Result 2

Task completed in 0.025 seconds

```
View STU_TEACH created.
```

Drop view:

```
drop view stu_teach
```

Script Output | Query Result | Query Result 1 | Query Result 2

Task completed in 0.023 seconds

```
View STU_TEACH dropped.
```

DISTINCT Clause:

- To eliminate the duplicate records from the result set, DISTINCT clause is used , but only with the SELECT statement.

Syntax:

SELECT DISTINCT columns
FROM
WHERE conditions;

Parameter:

columns: It is used to specify the columns to be selected.

table_name: Used to specify name of table.

conditions: It is used to specify the conditions to be strictly fulfilled for the action to complete.

Query:

```
SELECT DISTINCT student_name
FROM student
WHERE student_id = 2;
```

Output:

	STUDENT_NAME
1	smiley

DISTINCT clause with multiple columns:

Query:

```
SELECT DISTINCT student_name, student_age
FROM student
WHERE student_id >= 2;
```

Output:

	STUDENT_NAME	STUDENT_AGE
1	happy	11
2	bond	10
3	smiley	13
4	james	13

Union:

To combine the output sets of two or more Oracle SELECT statements, the Oracle UNION operator is used. During the combining process, the UNION operator removes the duplicate rows between the SELECT statements' results. There are however two mandatory conditions for using the UNION operator in Oracle.

*Each SELECT statement must have the same number of expressions.

*Each corresponding expression in the different SELECT statement should be of the same data type.

Output:

```
SELECT student_id as person_id,student_name as person_name,student_age as person_age
FROM student
UNION
SELECT teacher_id as person_id,teacher_name as person_name,teacher_age as person_age
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 8 in 0.003 seconds

	PERSON_ID	PERSON_NAME	PERSON_AGE
1	1	Joy	10
2	2	Smiley	13
3	3	Happy	11
4	4	James	13
5	5	Bond	10
6	101	James	30
7	102	Bond	25
8	103	Smith	40

Union All:

To combine the output sets of two or more Oracle SELECT statements, the Oracle UNION ALL operator is used. During the combining process, the UNION ALL operator does not remove the duplicate rows between the SELECT statements' results, but returns all of them and this feature makes it different from the Oracle UNION operator.

There are however two mandatory conditions for using the UNION ALL operator in Oracle.

*Each SELECT statement must have the same number of expressions.

*Each corresponding expression in the different SELECT statement should be of the same data type.

```
SELECT student_id as ID, student_age as AGE
FROM student
WHERE student_id > 2
UNION  ALL
SELECT teacher_id as ID, teacher_age as AGE
FROM teacher
WHERE teacher_age >= 30;
```

Script Output | Query Result

SQL | All Rows Fetched: 5 in 0.003 seconds

	ID	AGE
1	3	11
2	4	13
3	5	10
4	101	30
5	103	40

Joins:

Joins are used to combine rows from two or more tables based on relations

Create students table :

student_id	name	age
1	Joy	20
2	Smiley	19
3	Happy	21
4	James	22
5	Bond	25
6	Geetha	23

Teachers table:

Teacher_id	Teacher_name	Teacher_age
101	James	30
102	Bond	25
103	Smith	40
104	Geetha	23

Outer join:

Oracle supports three major types of Outer joins namely,

Left Outer Join, Right Outer Join and Full Outer Join.

Left Outer join:

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).

Syntax:

SELECT column_name(s)

FROM table1

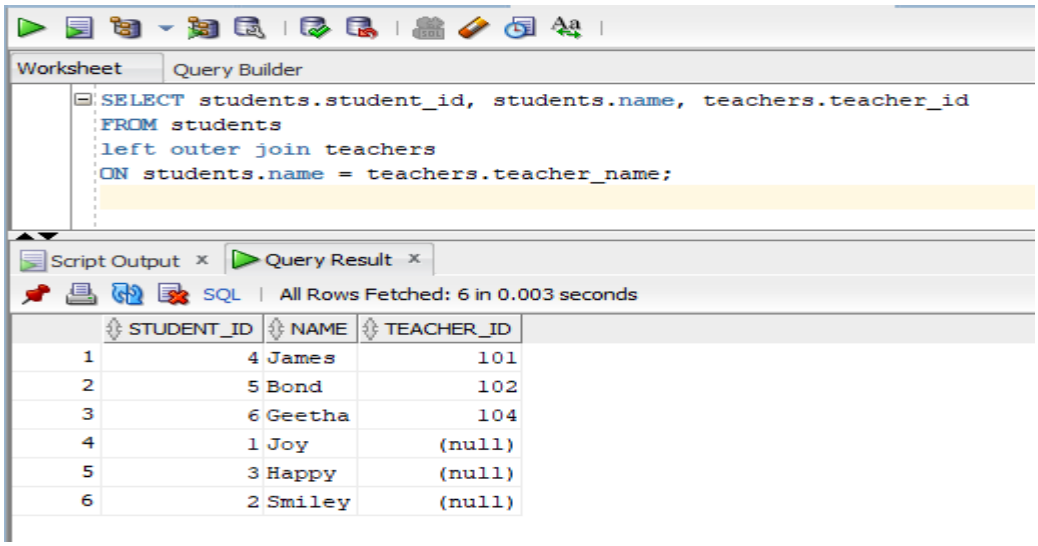
LEFT JOIN table2

ON table1.column_name = table2.column_name;

Query:

SELECT students.student_id, students.name, teachers.teacher_id
FROM students
left outer join teachers ON students.name = teachers.teacher_name;

output:



Right outer join:

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

Syntax:

SELECT column_name(s)

FROM table1

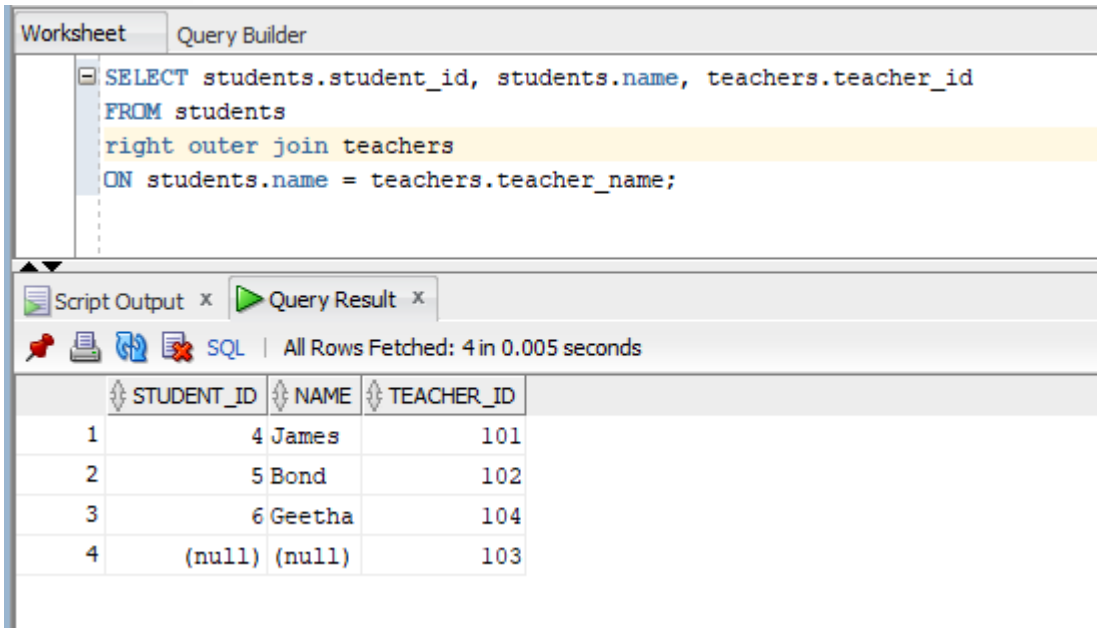
RIGHT JOIN table2

ON table1.column_name = table2.column_name;

QUERY:

SELECT students.student_id, students.name, teachers.teacher_id
FROM students
right outer join teachers
ON students.name = teachers.teacher_name;

Output:



Full outer join:

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Syntax:

SELECT column_name(s)

FROM table1

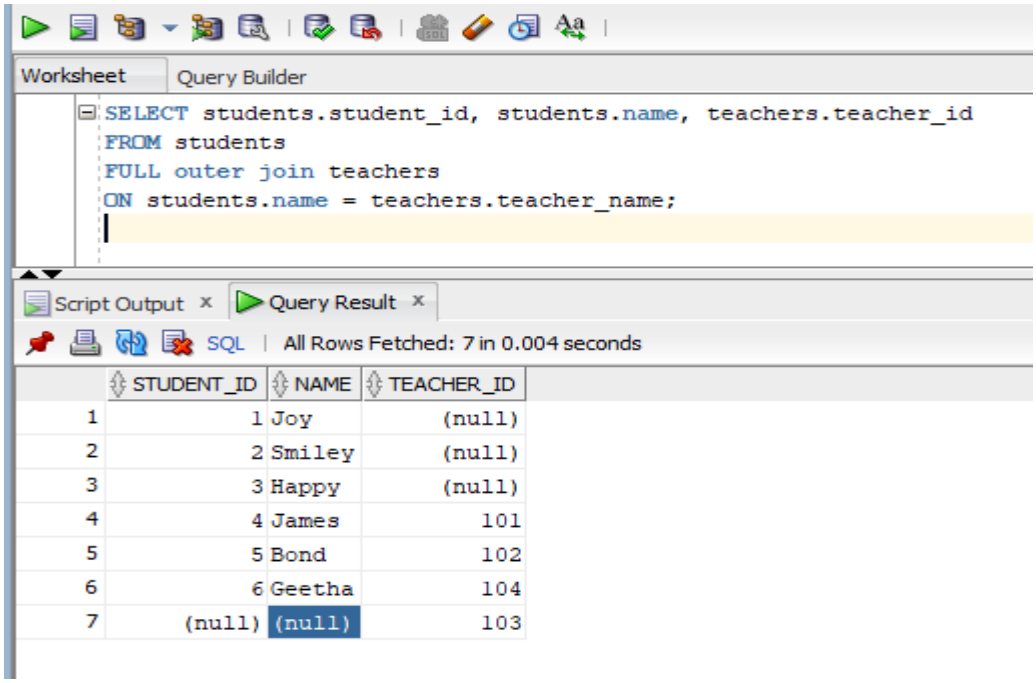
FULL OUTER JOIN table2

ON table1.column_name = table2.column_name;

QUERY:

SELECT students.student_id, students.name, teachers.teacher_id
FROM students
FULL outer join teachers
ON students.name = teachers.teacher_name;

OUTPUT:



Inner join:

INNER Join is often called a SIMPLE Join as it is the simplest among all the kinds of joins.

The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

SELECT expr_1, expr_2, ... expr_n
FROM table_1
INNER JOIN table_2
ON join_predicate;

Example:

Query:

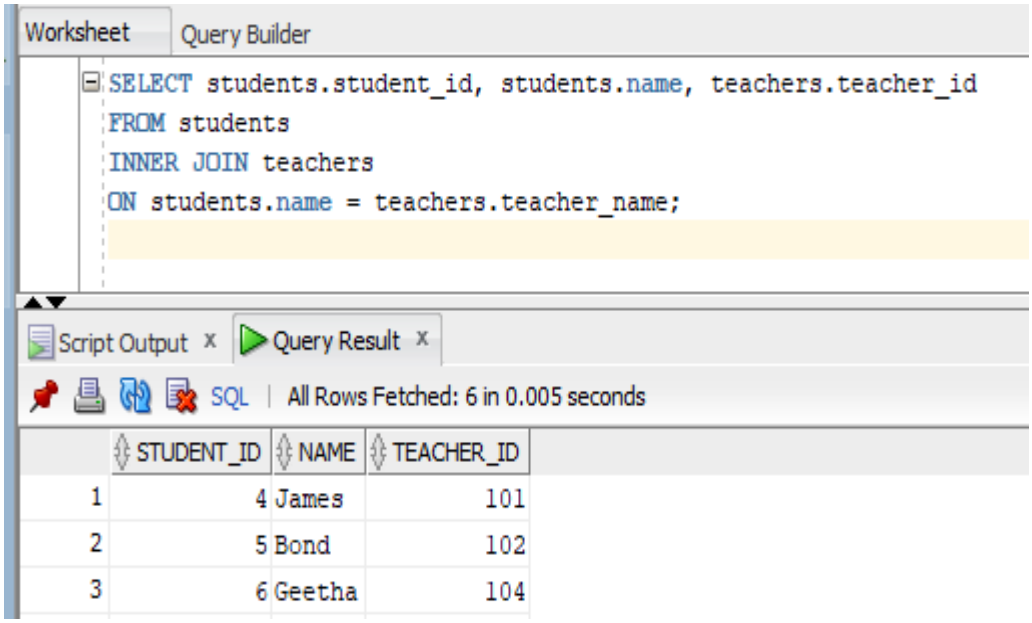
SELECT students.student_id, students.name, teachers.teacher_id

FROM students

INNER JOIN teachers

ON students.name = teachers.teacher_name

Output:



Cross join:

The Oracle CROSS Join query joins all the rows of one table with all the rows of another table and then displays the result.
For instance, if the FIRST table has x rows and the Second Table has y rows than the resultant table will have x*y rows.

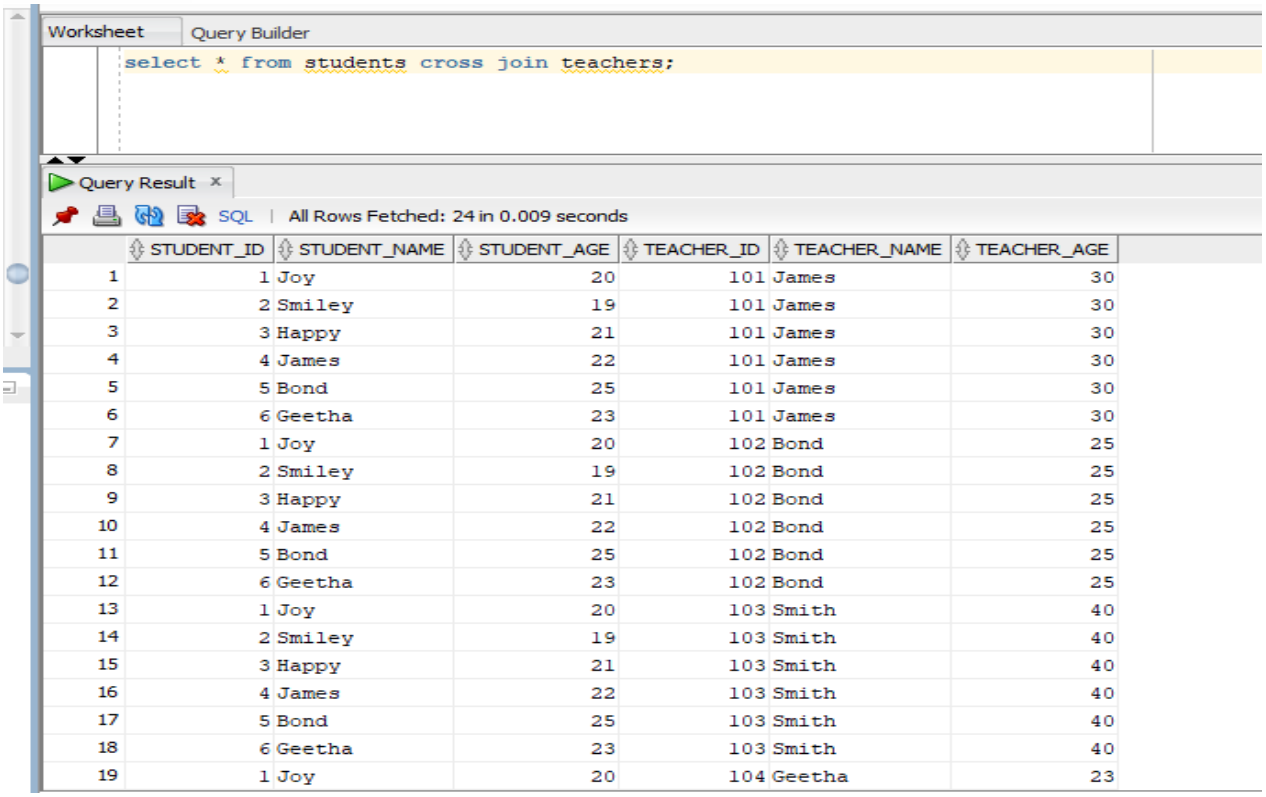
Syntax:

SELECT * FROM
table_1 CROSS JOIN
table_2;

Query:

Select * from students cross join teachers

Output:



	STUDENT_ID	STUDENT_NAME	STUDENT_AGE	TEACHER_ID	TEACHER_NAME	TEACHER_AGE
1	1	Joy	20	101	James	30
2	2	Smiley	19	101	James	30
3	3	Happy	21	101	James	30
4	4	James	22	101	James	30
5	5	Bond	25	101	James	30
6	6	Geetha	23	101	James	30
7	1	Joy	20	102	Bond	25
8	2	Smiley	19	102	Bond	25
9	3	Happy	21	102	Bond	25
10	4	James	22	102	Bond	25
11	5	Bond	25	102	Bond	25
12	6	Geetha	23	102	Bond	25
13	1	Joy	20	103	Smith	40
14	2	Smiley	19	103	Smith	40
15	3	Happy	21	103	Smith	40
16	4	James	22	103	Smith	40
17	5	Bond	25	103	Smith	40
18	6	Geetha	23	103	Smith	40
19	1	Joy	20	104	Geetha	23

EQUI JOIN:

The Oracle EQUI Join query always uses a comparison operator to check the matching columns of the associated tables, and then the matching columns of the two tables are displayed as the result.

Syntax:

SELECT expr_1, expr_2, ... expr_n
FROM table_1 JOIN
table_2 ON
join_predicate;

Query:

SELECT students.student_id, students.student_name, teachers.teacher_id

FROM students, teachers

WHERE students.student_name = teachers.teacher_name;

Or

Using With join

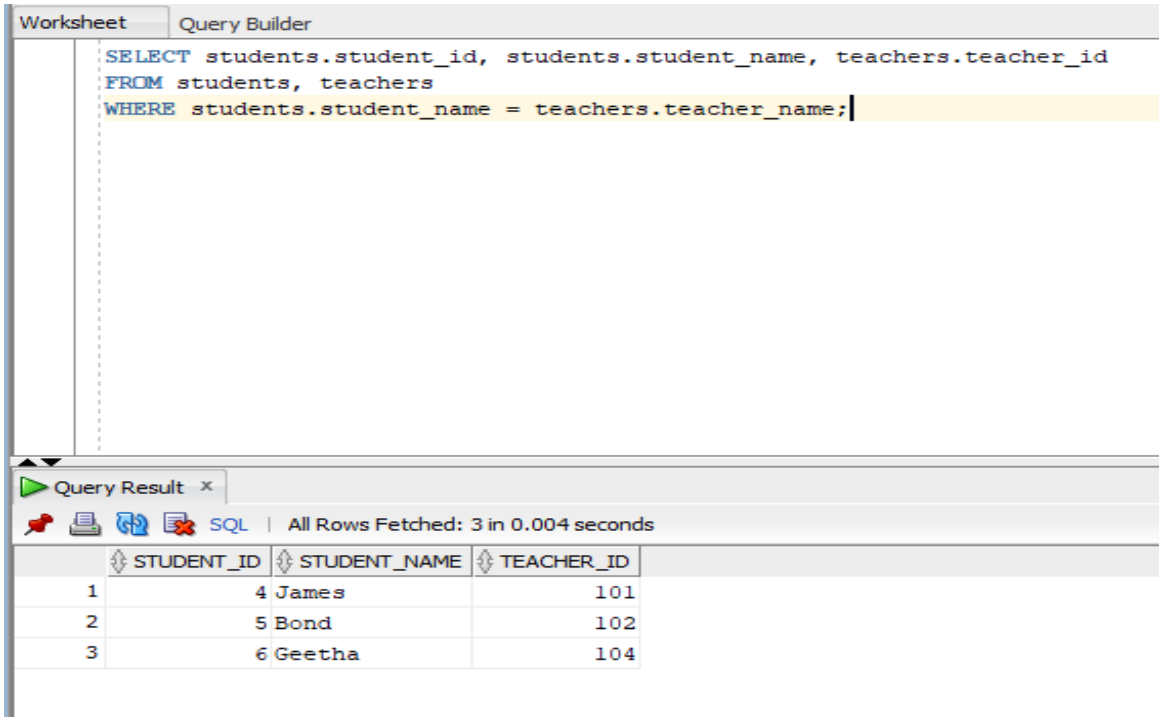
SELECT students.student_id, students.student_name, teachers.teacher_id

FROM students join

teachers on

students.student_name = teachers.teacher_name;

Output:



SEMI join:

SEMI Join is used to return one copy of those rows from a table where at least one match is found in the values with the other mentioned table, and to serve this purpose,

EXISTS construct is used instead of any JOIN keyword.

The main advantage of this kind of Join query is that it makes the queries run faster and thus is a very powerful SQL construct.

Syntax:

SELECT columns

FROM table_1

WHERE EXISTS (

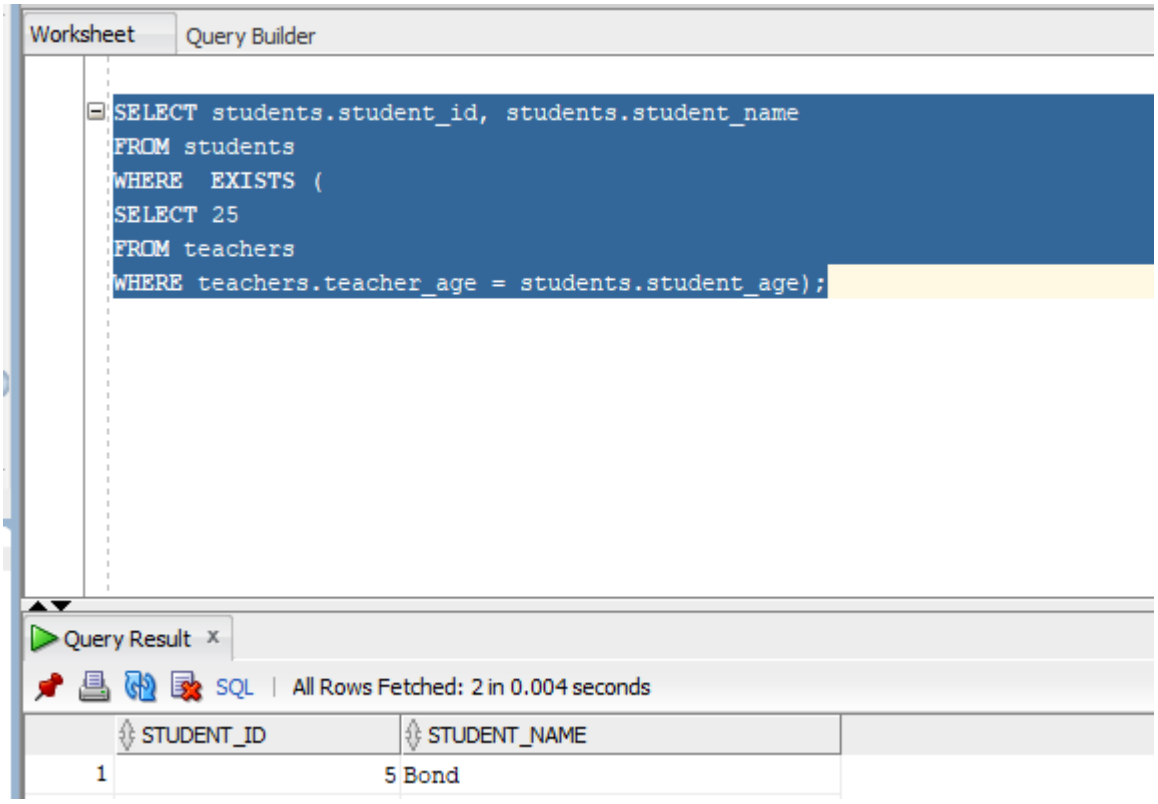
SELECT values FROM

table_2
WHERE table_2.column = table_1.column);

Query:

```
SELECT students.student_id, students.student_name
FROM students
WHERE  EXISTS (
SELECT 25
FROM teachers
WHERE teachers.teacher_age = students.student_age);
```

Output:



SELF JOIN:

A self join is a regular join, but the table is joined with itself. i,e, each row is combined with itself and every other rows of the table.

SYNTAX:

SELECT column_name(s)

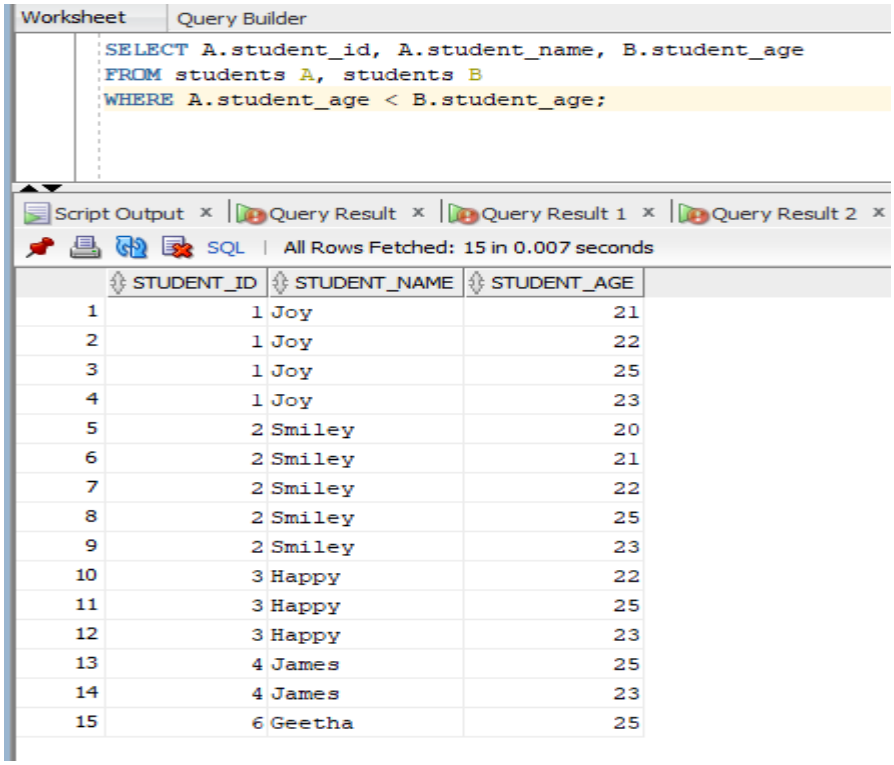
FROM table1 T1, table1 T2

WHERE condition;

Query:

```
SELECT A.student_id, A.student_name, B.student_age
FROM students A, students B
WHERE A.student_age < B.student_age;
```

Output:



	STUDENT_ID	STUDENT_NAME	STUDENT_AGE
1	1	Joy	21
2	1	Joy	22
3	1	Joy	25
4	1	Joy	23
5	2	Smiley	20
6	2	Smiley	21
7	2	Smiley	22
8	2	Smiley	25
9	2	Smiley	23
10	3	Happy	22
11	3	Happy	25
12	3	Happy	23
13	4	James	25
14	4	James	23
15	6	Geetha	25

Anti join:

Anti-join is used to make the queries run faster.

It is a very powerful SQL construct Oracle offers for faster queries.

Anti-join between two tables returns rows from the first table where no matches are found in the second table.

It is opposite of a semi-join. An anti-join returns one copy of each row in the first table for which no match is found. Anti-joins are written using the NOT EXISTS or NOT IN constructs.

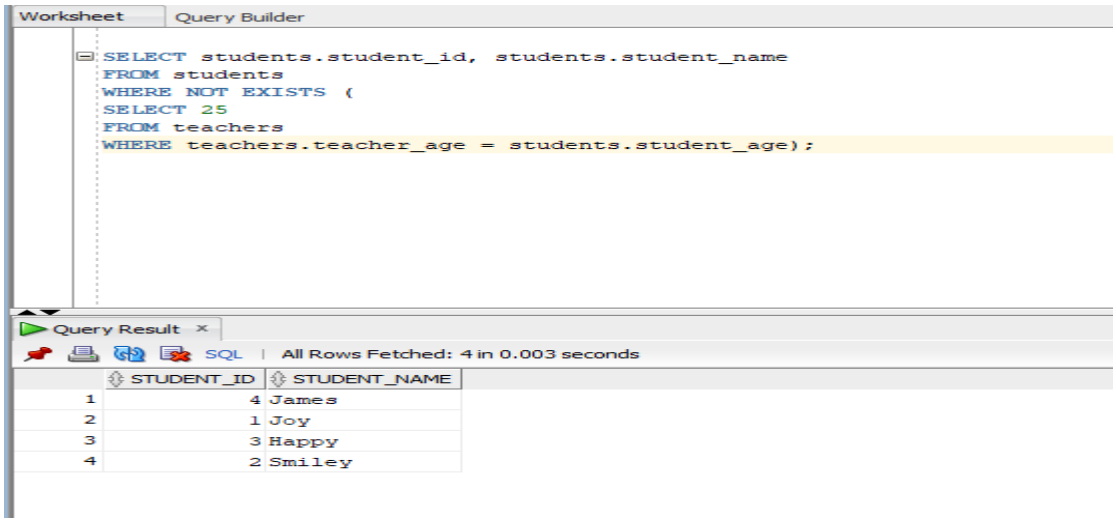
Syntax:

SELECT columns
FROM table_1
WHERE NOT EXISTS (
SELECT values FROM
table_2
WHERE table_2.column = table_1.column);

Query:

SELECT students.student_id, students.student_name
FROM students
WHERE NOT EXISTS (
SELECT 25
FROM teachers
WHERE teachers.teacher_age = students.student_age);

Output:



Rownum:

ROWNUM is one of the vital Numeric/Math functions of Oracle. It is used to get a number that represents the order in which a row from a table or joined tables is selected by the Oracle. The ROWNUM function is supported in the

various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Example:

```
SELECT ROWNUM, student.*
FROM student
WHERE student_age > 10
```

Script Output | Query Result

SQL | All Rows Fetched: 3 in 0.004 seconds

	ROWNUM	STUDENT_ID	STUDENT_NAME	STUDENT_AGE
1	1	2	Smiley	13
2	2	3	Happy	11
3	3	4	James	13

USERENV:

USERENV is an advanced function that the Oracle database supports. It is used to get the information about the current Oracle session.

The USERENV function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Example:

```
select USERENV('sessionid') FROM DUAL
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 1 in 0 seconds

	USERENV('SESSIONID')
1	4294967295

Acos( ):

ACOS is one of the vital Numeric/Math functions of Oracle. It is used to get the arc cosine of a number.

The ACOS function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Example:

```
select acos(-1) from dual
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 1 in 0.001 seconds

	ACOS(-1)
1	3.14159265358979323846264338327950288842

Add_months:

ADD_MONTHS is one of the vital Date/Time functions of Oracle. It is used to get a date with a specified number of months added.

The ADD_MONTHS function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Example:

```
select sysdate,
add_months(sysdate,2)
from dual;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.001 seconds

	SYSDATE	ADD_MONTHS(SYSDATE,2)
1	31-01-22	31-03-22

```
select sysdate,
add_months(sysdate,-2)
from dual;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.001 seconds

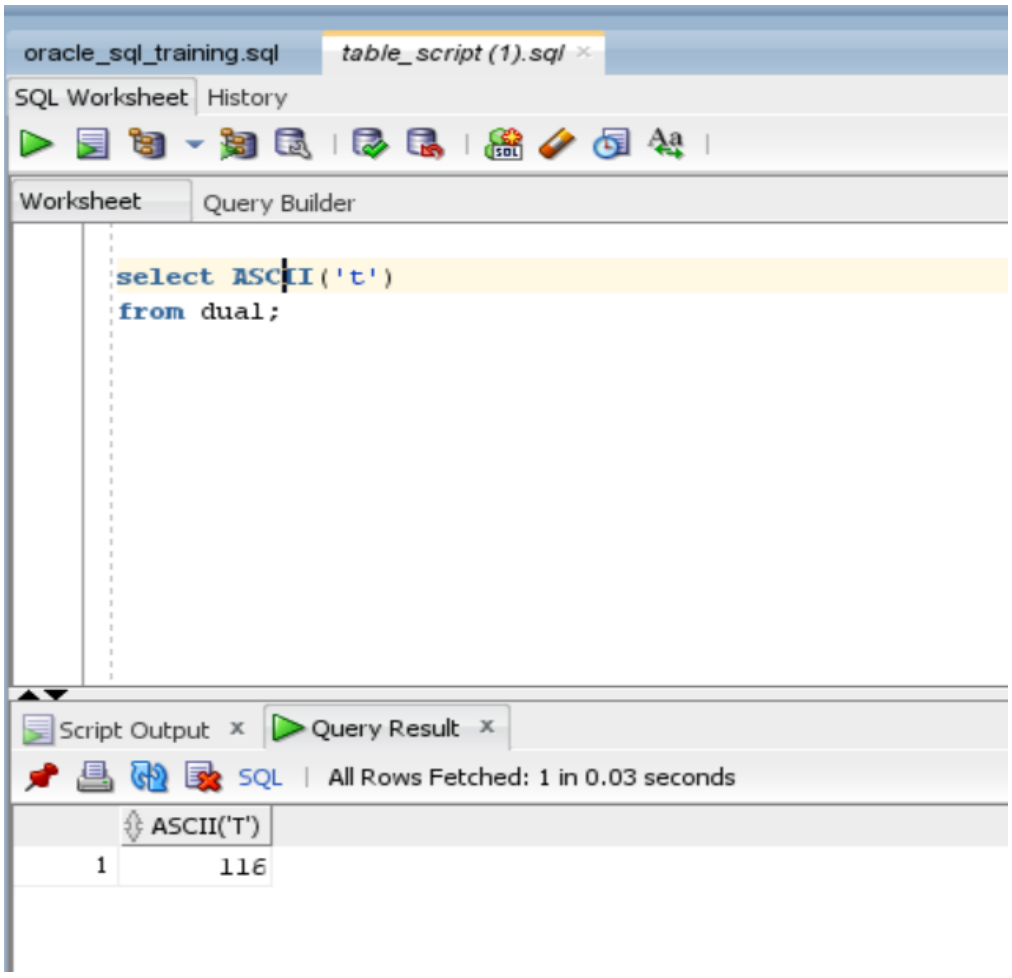
	SYSDATE	ADD_MONTHS(SYSDATE,2)
1	31-01-22	31-03-22

ASCII function in Oracle

The Oracle/PLSQL ASCII function returns the NUMBER code that represents the specified character.

SYNTAX :

ASCII('single character')



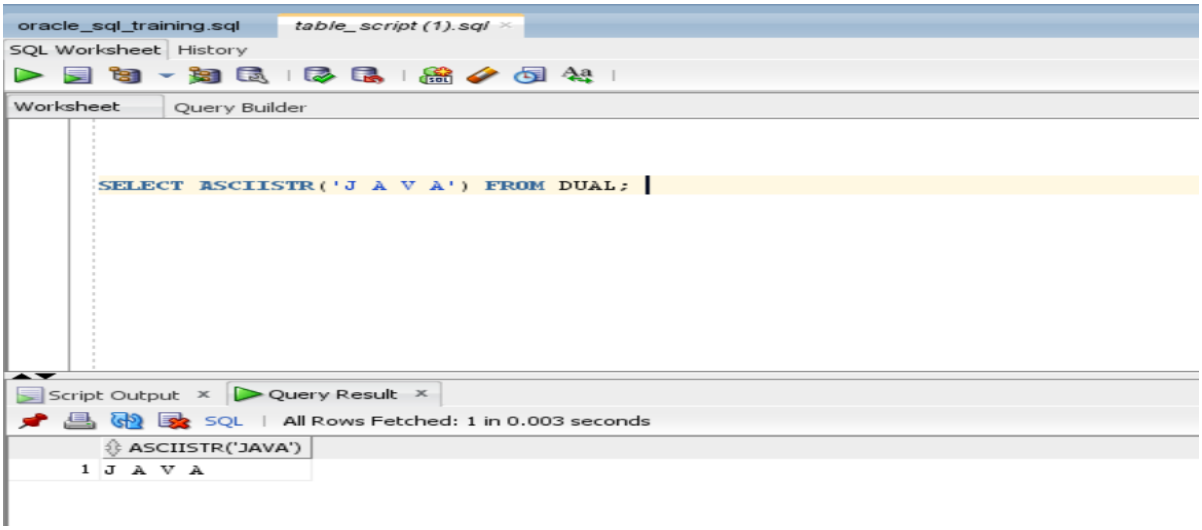
ASCIISTR function in Oracle

ASCIISTR converts a string in any character set to ASCII in the database character set. Non-ASCII characters are represented as \ xxxx, where xxxx is a UTF-16 code unit. char can be any character string.

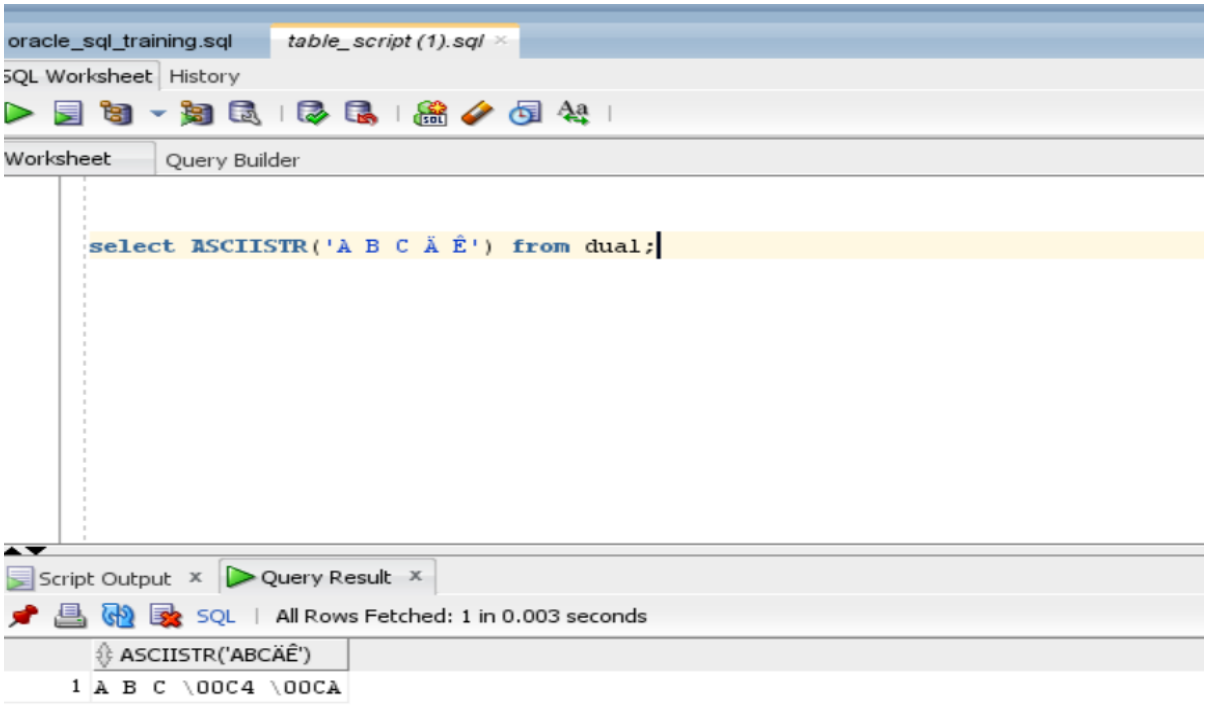
ASCIISTR('Skåne') returns the value Sk\00E5ne .

SYNTAX: ASCIISTR(STRING)

EXAMPLE1



EXAMPLE2



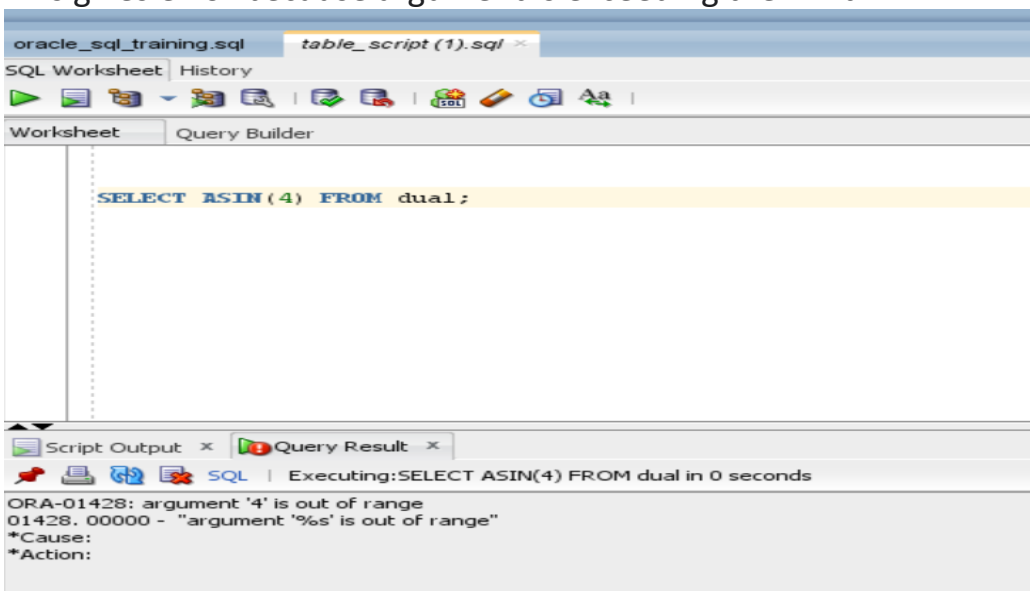
ASIN function in Oracle

The ASIN() is used to calculate the angle value (in radians) of a specified sine.

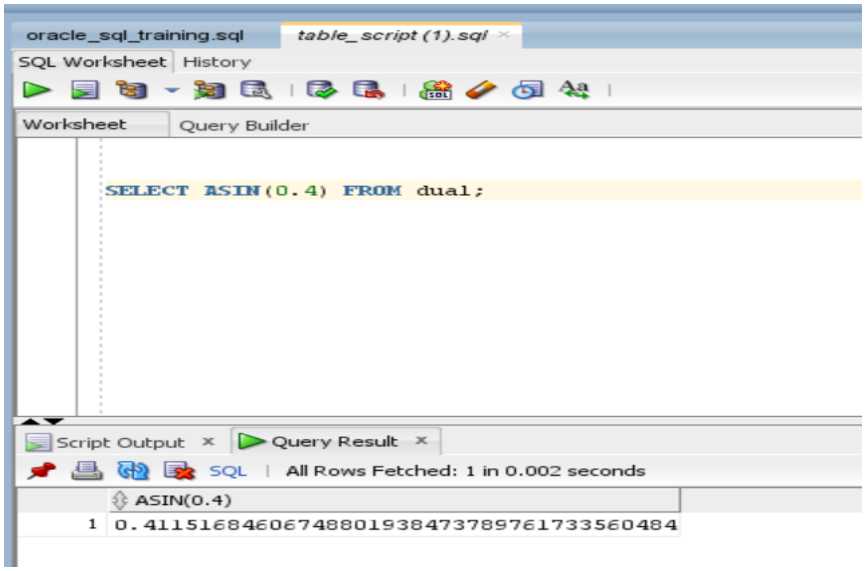
SYNTAX :

ASIN(VALUE)

EXAMPLE1:
This gives error because argument is exceeding the limit.

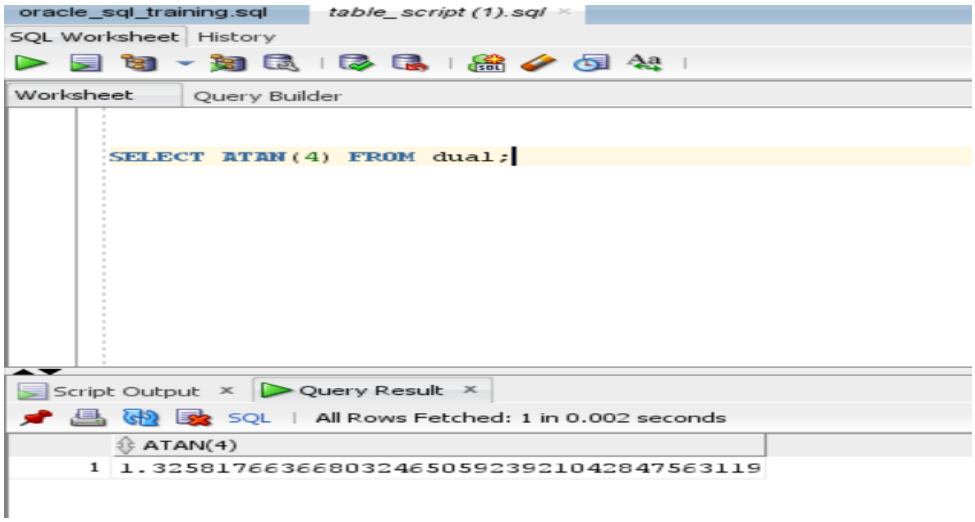


EXAMPLE 2



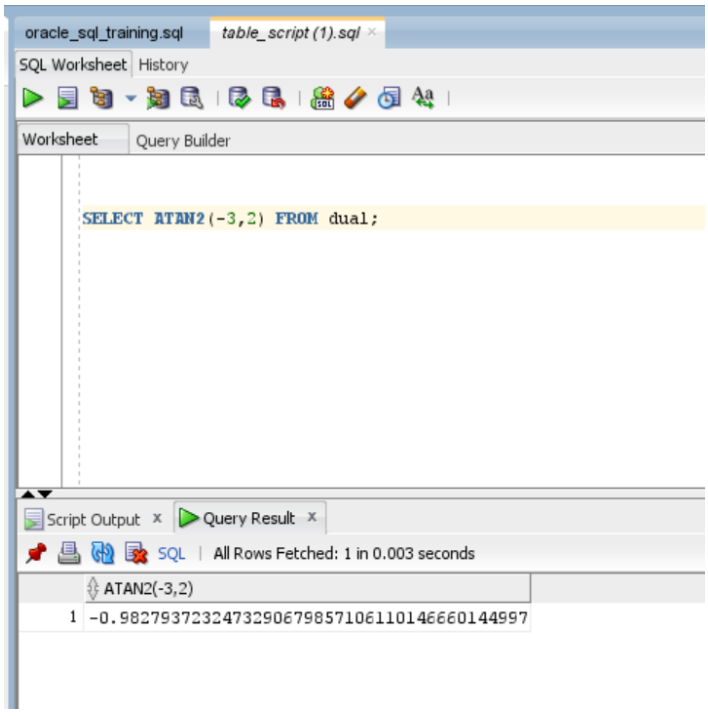
ATAN function in Oracle

The ATAN() function is **used to calculate the angle value (in radians) of a specified tangent**. The specified number can be in an unbounded range and returns a value in the range of -pi/2 to pi/2, expressed in radians.



ATAN2 function in Oracle

The ATAN2() function is used to calculate the arc tangent of two numbers for a point on a Cartesian plane. The first number in the argument can be in an unbounded range and returns a value in the range of -pi to pi, depending on the signs of num1 and num2 specified in the argument, expressed in radians.

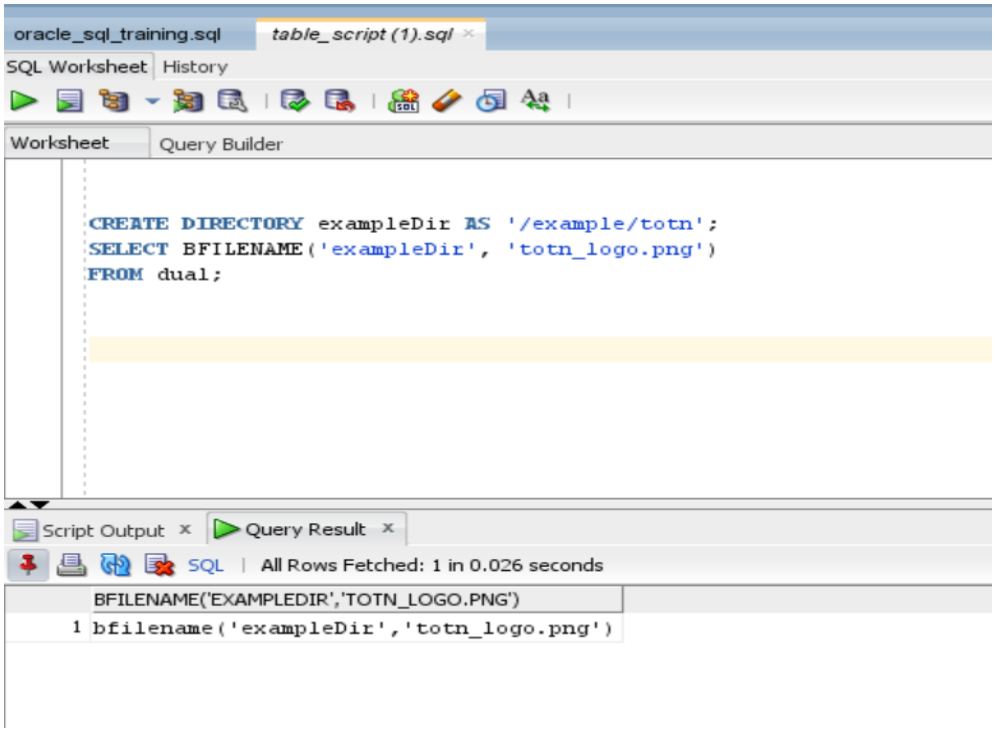


BFILENAME function in Oracle

The Oracle/PLSQL BFILENAME function returns a BFILE locator for a physical LOB binary file.

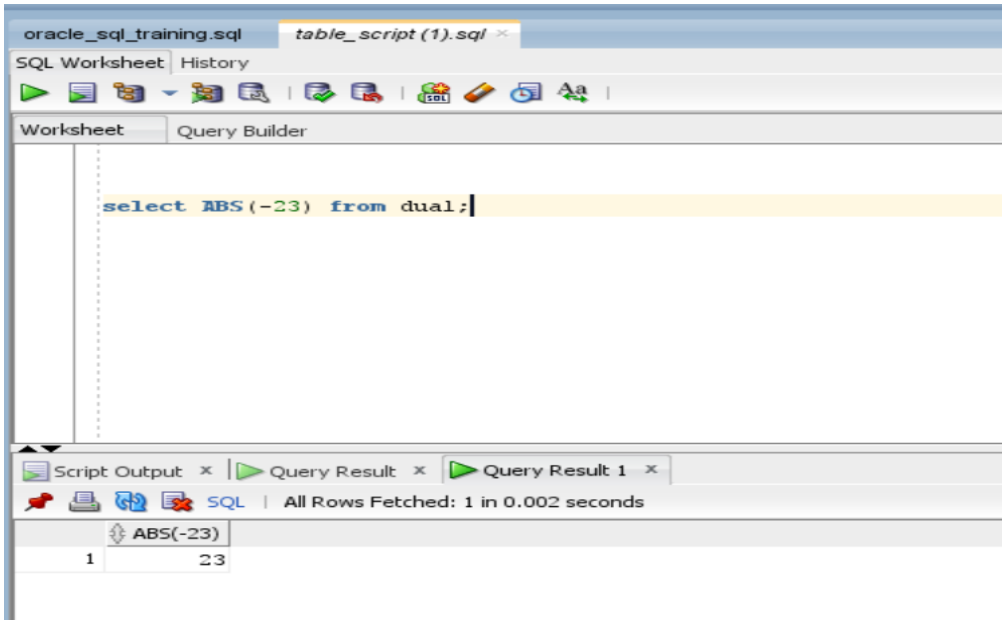
SYNTAX:

BFILENAME (‘directory’ , ’filename’)



ABS function in Oracle

The Oracle/PLSQL ABS function returns the absolute value of a number.



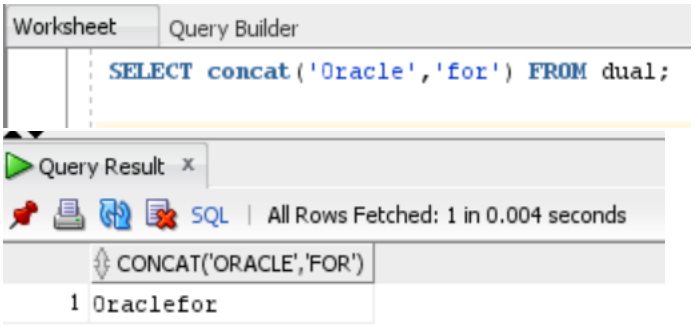
ORACLE SQL FUNCTIONS

CONCAT Function: - The Oracle CONCAT function allows you to join, or concatenate, two strings together.

Example 1: - Concatenate 2 strings

Syntax: - SELECT concat('string1','string2') FROM dual;

OUTPUT

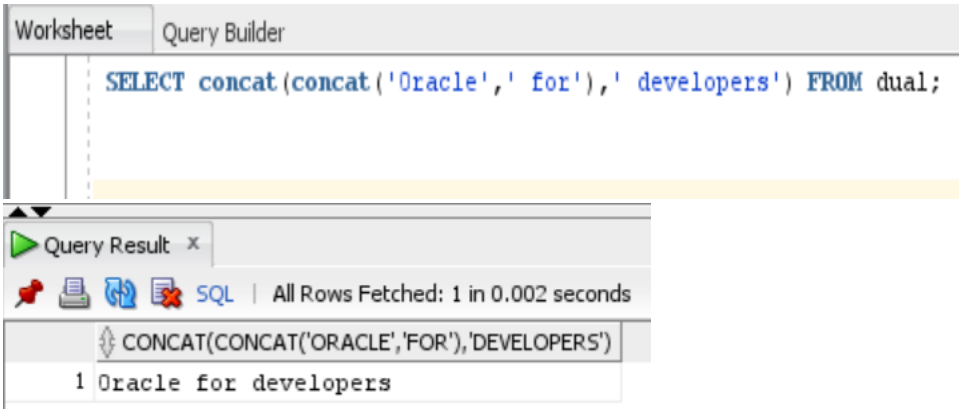


Example 2: - Concatenate more than 2 strings

In Oracle, the CONCAT function will only allow to concatenate two values together. If we have to concatenate more values than two, we can nest multiple CONCAT function calls.

Syntax: - SELECT concat(concat('string1,' string2'),' string3') FROM dual;

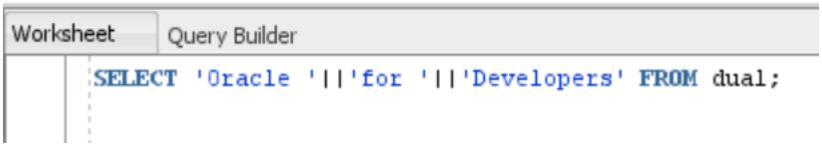
OUTPUT



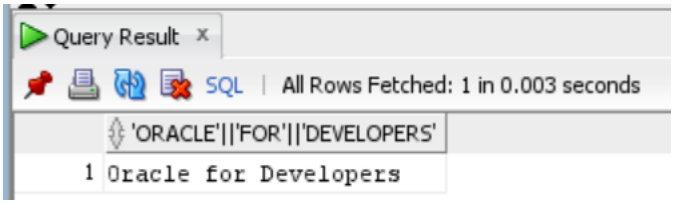
Example 3: - | | Operator

The CONCAT function is one method to concatenate strings in Oracle. An alternative to using the CONCAT function would be to use the | | operator to concatenate multiple strings.

Syntax: - SELECT 'string1 '| |'string2 '| |'string3' FROM dual;

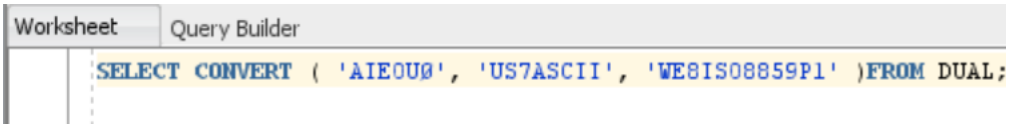


OUTPUT: -

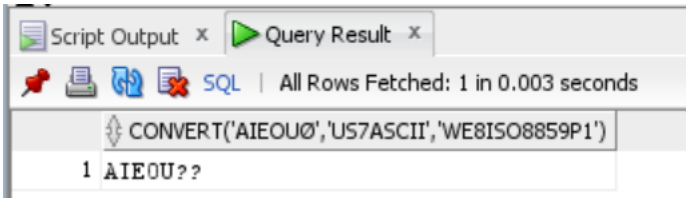


CONVERT Function: - The CONVERT function converts a string from one-character set to another.

Syntax: - SELECT CONVERT(string, chat_set_to, char_set_from)



OUTPUT: -

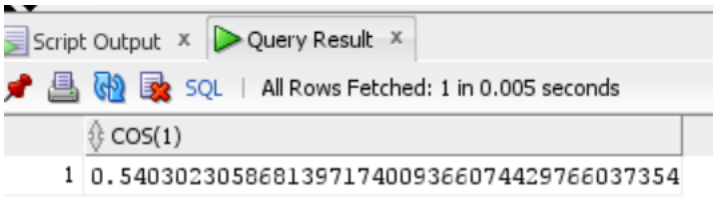


The required string is converted from West European 8-bit character set to US 7-bit ASCII character set.

COS Function: - It is used to get the cosine of a number.

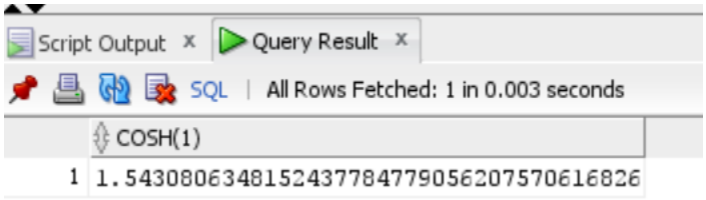
Syntax: - COS (number)

OUTPUT: -

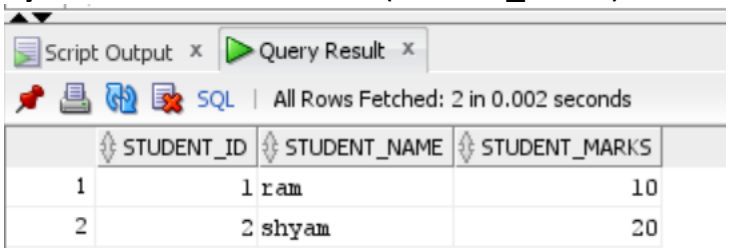


COSH Function: - It is used to get the hyperbolic cosine of a number.
Syntax: -COSH(number)

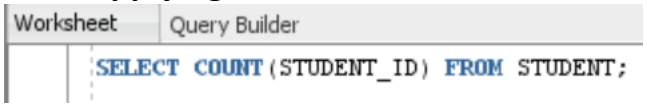
OUTPUT: -



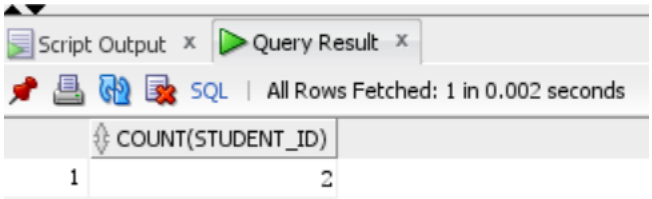
COUNT Function: - COUNT() function returns the number of items including NULL and duplicate values.
Syntax: - SELECT COUNT (column_name) FROM TABLE_NAME;



After applying COUNT Function: -



OUTPUT: -



CURRENT_TIMESTAMP Function: - It is used to get the current date and time in the time zone of the current SQL session.
Syntax: - SELECT CURRENT_TIMESTAMP FROM DUAL;

```
SELECT CURRENT_TIMESTAMP FROM DUAL;
```

OUTPUT: -

	CURRENT_TIMESTAMP
1	31-01-22 4:17:25.277000000 PM ASIA/CALCUTTA

CORR Function: - The *CORR function* is used to calculate the *correlation coefficient.*

Correlation coefficient: - The correlation coefficient describes how one variable moves in relation to another.

Syntax: - CORR (x, y)

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Where,

r = Pearson Correlation Coefficient

x_i = x variable samples

y_i = y variable sample

$\bar{x}$ = mean of values in x variable

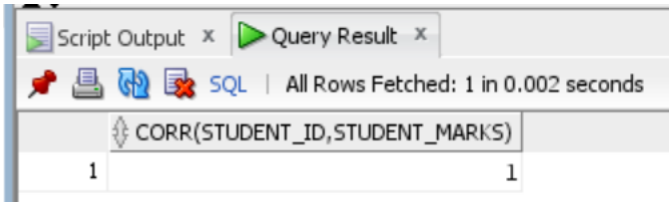
$\bar{y}$ = mean of values in y variable

	STUDENT_ID	STUDENT_NAME	STUDENT_MARKS
1	1	ram	10
2	2	shyam	20

After using CORR Function

```
select corr(student_id,student_marks) from student;
```

OUTPUT: -

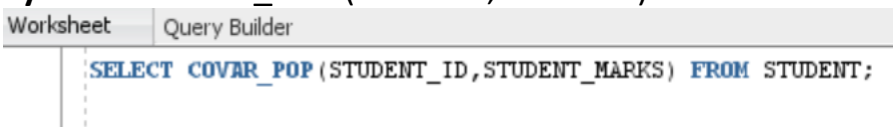


COVAR_POP Function: - It is used to get the population covariance of a set of number pairs.
Population covariance refers to the value of variance which calculated from population data.
Formula: -

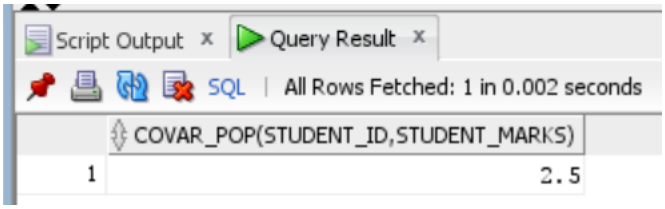
$$\sigma^2 = \sum (X_i - \bar{X})^2 / N$$

σ^2 = variance
X_i = the value of the ith element
$\bar{X}$ = the mean of X
N = the number of elements

Syntax: - COVAR_POP (column1, column2)



OUTPUT: -



COVAR_SAMP Function: - It is used to get the sample covariance of a set of number pairs.
Sample covariance -It is variance calculated from sample data.
Formula: -

$$s^2 = \sum (x_i - \bar{x})^2 / (n - 1)$$

s^2 = *sample variance*
x_i = *ith element of the sample*
$\bar{x}$ = *mean of the sample*
n = *sample size*

```
Worksheet  Query Builder
    SELECT COVAR_SAMP(STUDENT_ID,STUDENT_MARKS) FROM STUDENT;
```

OUTPUT: -

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.003 seconds

	COVAR_SAMP(STUDENT_ID,STUDENT_MARKS)
1	5

CUME_DIST Function: - It is used to get a cumulative distribution of a value in a group of values.

Cumulative distribution: - A function that gives the probability that a random variable is less than or equal to the independent variable of the function.

Syntax: - CUME_DIST (expression) WITHIN GROUP (ORDER BY expression)

```
Worksheet  Query Builder
    select CUME_DIST(20) WITHIN GROUP (ORDER BY student_marks)
    from student;
```

OUTPUT: -

Script Output | Query Result | Query Result 1 | Query Result 2 | SQL | All Rows Fetched: 1 in 0.001 seconds

	CUME_DIST(20)WITHINGROUP(ORDERBYSTUDENT_MARKS)
1	1

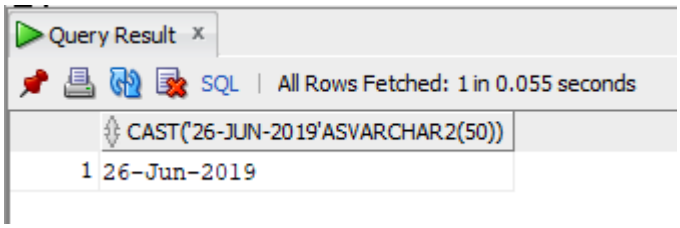
CAST function in Oracle:

CAST is one of the vital Conversion functions of Oracle. It is used to convert a datatype to another.

Syntax: select CAST('26-Jun-2019' AS VARCHAR2(50)) from dual;

```
select CAST( '26-Jun-2019' AS VARCHAR2(50) )
from dual;
```

Result: *26-Jun-2019 is converted from date to varchar2 datatype.*



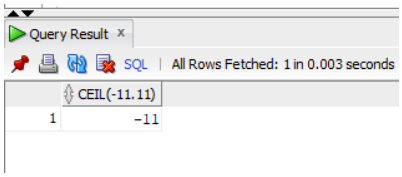
CEIL function in Oracle:

CEIL is one of the vital Numeric/Math functions of Oracle. It is used to get the smallest integer value which is either greater than or equal to the mentioned number.

Syntax: select CEIL (-11.11) from dual;

```
select CEIL (-11.11) from dual;
```

Result: *Rounded of to greater value from -11.11 to -11.*



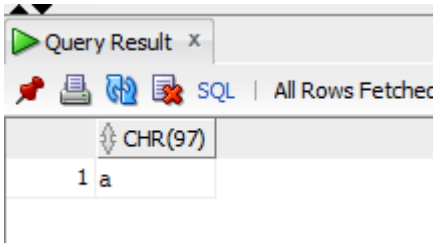
CHR Function In Oracle:

CHR is another vital string/char functions of Oracle. It is used to return the actual character or symbol for the desired ASCII code.

Syntax: select CHR(97)from dual;

```
select CHR( 97 )from dual;
```

Result: *The value of chr is taken as ASCII and converted to its character.*



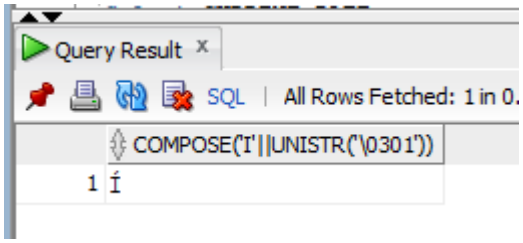
COMPOSE function in oracle:

COMPOSE is another vital string/char functions of Oracle. It is used to return the UNICODE String for the desired string.

Syntax: select COMPOSE('I' || unistr('\0301')) from dual;

```
select COMPOSE( 'I' || unistr( '\0301' )) from dual;
```

Result: *This function gives different font to given string using given unicodes.*



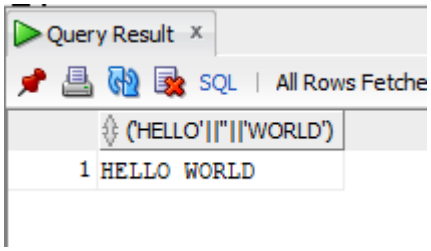
CONCAT WITH || function in oracle:

The || is an operator which is used to simplify the work of the CONCAT function and thus is used to concatenate two or more strings together.

Syntax: SELECT ('HELLO' || ' ' || 'WORLD') FROM dual;

```
SELECT ('HELLO' || ' ' || 'WORLD')
FROM dual;
```

Result: *This concatenates multiple strings into a single string.*



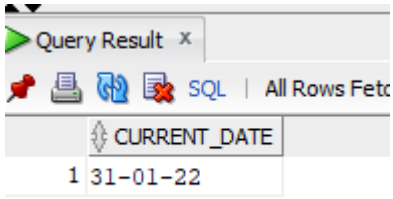
CURRENT_DATE function in oracle:

CURRENT_DATE is one of the vital Date/Time functions of Oracle. It is used to get the current date in the time zone of the current SQL session.

Syntax: Select CURRENT_DATE from dual;

```
Select CURRENT_DATE
from dual;
```

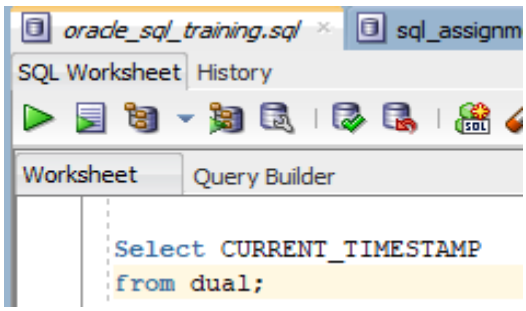
Result: *This provides us the current date of sql session ongoing.*

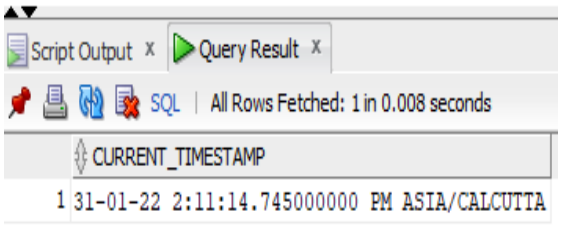


CURRENT_TIMESTAMP function in Oracle

The CURRENT_TIMESTAMP() function returns the current date and time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE

Syntax: Select CURRENT_TIMESTAMP from dual;

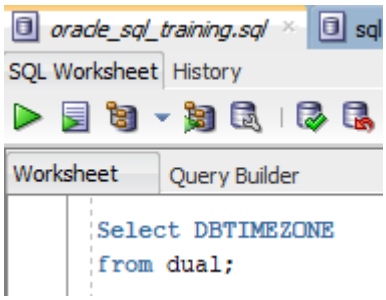


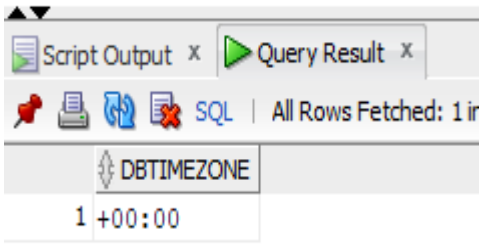


DBTIMEZONE function in Oracle

DBTIMEZONE **returns the value of the database time zone**. The return type is a time zone offset (a character type in the format '[+|-]TZH:TZM') or a time zone region name, depending on how the user specified the database time zone value

Syntax: Select DBTIMEZONE from dual;

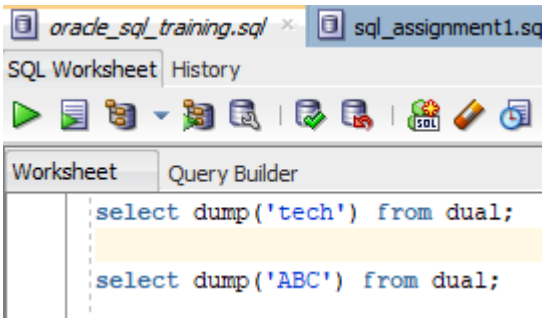


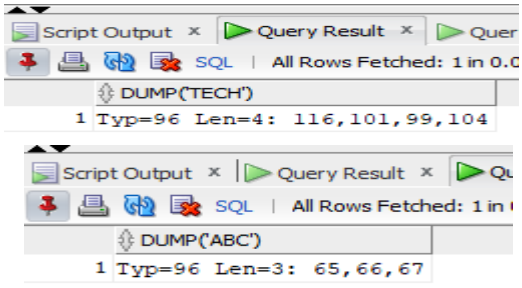


DUMP function in Oracle

The Oracle/PLSQL DUMP function returns a varchar2 value that includes the datatype code, the length in bytes, and the internal representation of the expression

Syntax: select dump('tech') from dual;

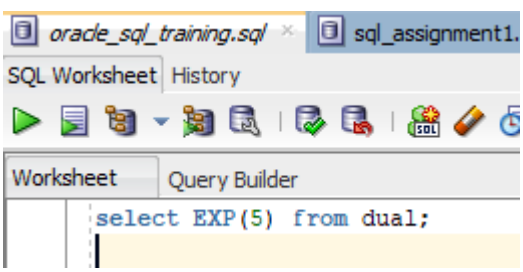


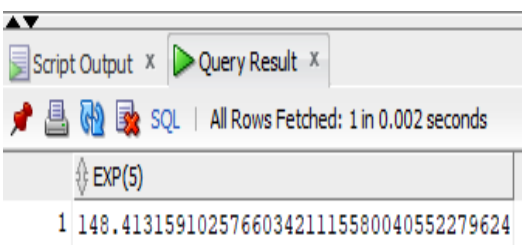


EXP function in Oracle

The Oracle/PLSQL EXP function returns e raised to the nth power, where e = 2.71828183.

Syntax: select EXP(5) from dual;

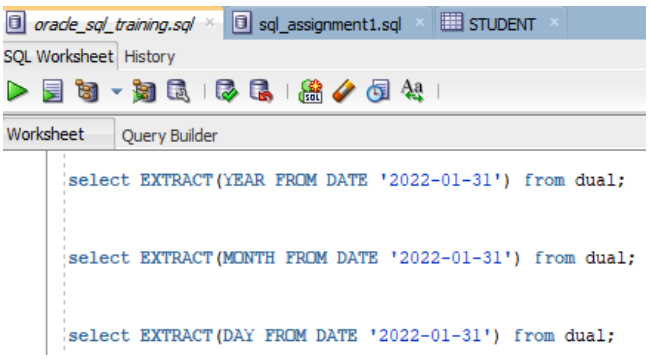


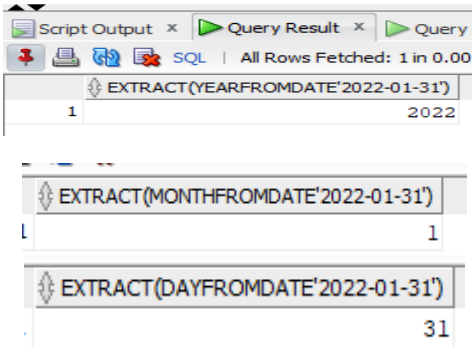


EXTRACT function in Oracle

The Oracle/PLSQL EXTRACT function extracts a value from a date or interval value.

Syntax: select EXTRACT(YEAR FROM DATE '2022-01-31') from dual;

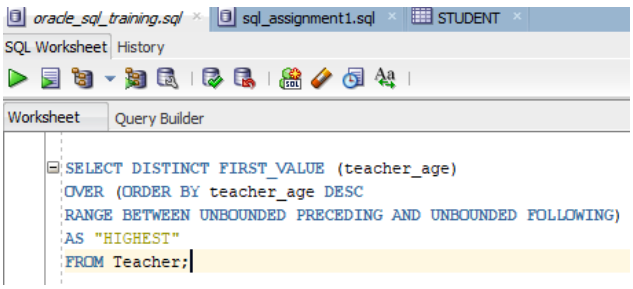


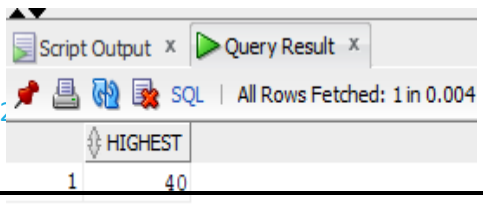


FIRST_VALUE function in Oracle

The Oracle/PLSQL FIRST_VALUE function returns the first value in an ordered set of values from an analytic window.

Syntax: SELECT DISTINCT FIRST_VALUE (teacher_age) OVER (ORDER BY teacher_age DESC
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
AS "HIGHEST" FROM Teacher;

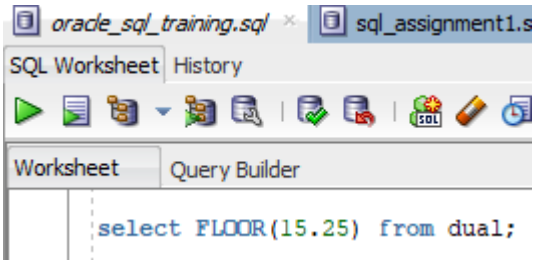


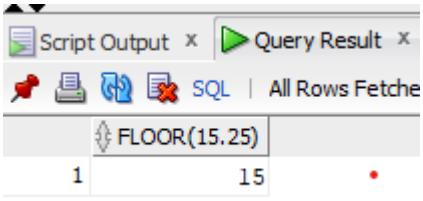


FLOOR function in Oracle

The Oracle/PLSQL FLOOR function returns the smallest integer value that is equal to or less than a *number*.

Syntax: select FLOOR(15.25) from dual;





FROM_TZ function in Oracle

FROM_TZ is one of the vital Conversion functions of Oracle. It is used to convert a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value. The FROM_TZ function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

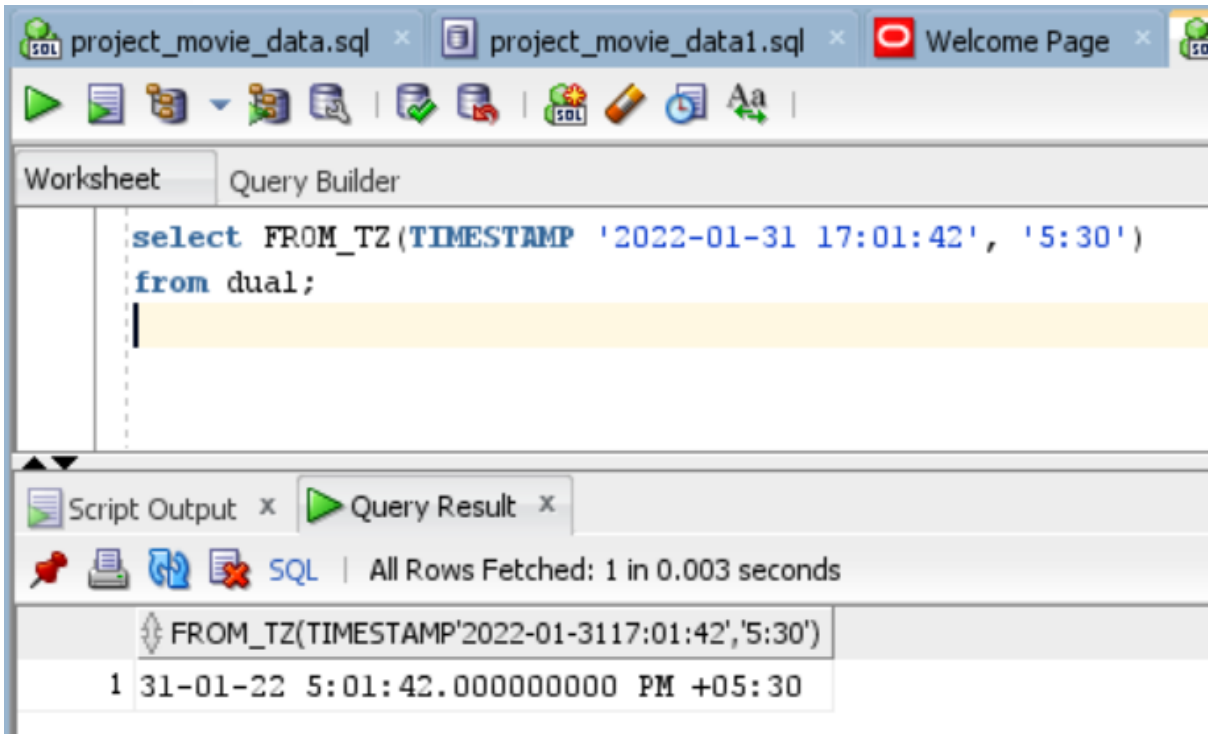
Syntax:

```
FROM_TZ ( timestamp_value, time_zone_value)
```

Parameters:

timestamp_value: It is used to specify the timestamp value to be converted.
time_zone_value: It is used to specify the time zone value to be used for conversion.

Example:



Explanation:

The TIMESTAMP WITH TIME ZONE value is the result.

GLOBAL TEMP TABLES

Temporary tables are much like an ordinary table, with an only vital difference of not having foreign keys related to other tables. It contains all the attributes of a table including information about rows and block, triggers, join cardinality, etc.

Syntax: To create a Global Temporary Table with a single column.

CREATE GLOBAL TEMPORARY TABLE table_name

(column_name data_type [NULL | NOT NULL]);

Syntax: To create a Global Temporary Table with multiple columns.

CREATE GLOBAL TEMPORARY TABLE table_name

```
( column_1 data_type [ NULL | NOT NULL ],
 column_2 data_type [ NULL | NOT NULL ],
 ...
column_n data_type [ NULL | NOT NULL ]
);
```

Parameters:

table_name: It is used to specify the name of the global temporary table to be created.

column definition:

It is used to specify the name of the column to be created in the global temporary table. The next definition is to specify the datatype of the column. The column constraint must also be defined as either NULL or NOT NULL. The column constraint holds a default value of NULL.

Example: Creating a GLOBAL TEMPORARY TABLE with multiple columns.

```
CREATE GLOBAL TEMPORARY TABLE retailers
( retailer_id number(10) NOT NULL,
  retailer_name varchar2(30) NOT NULL,
  retailer_city varchar2(50)  NULL,
  retailer_state varchar2(50)  );
```

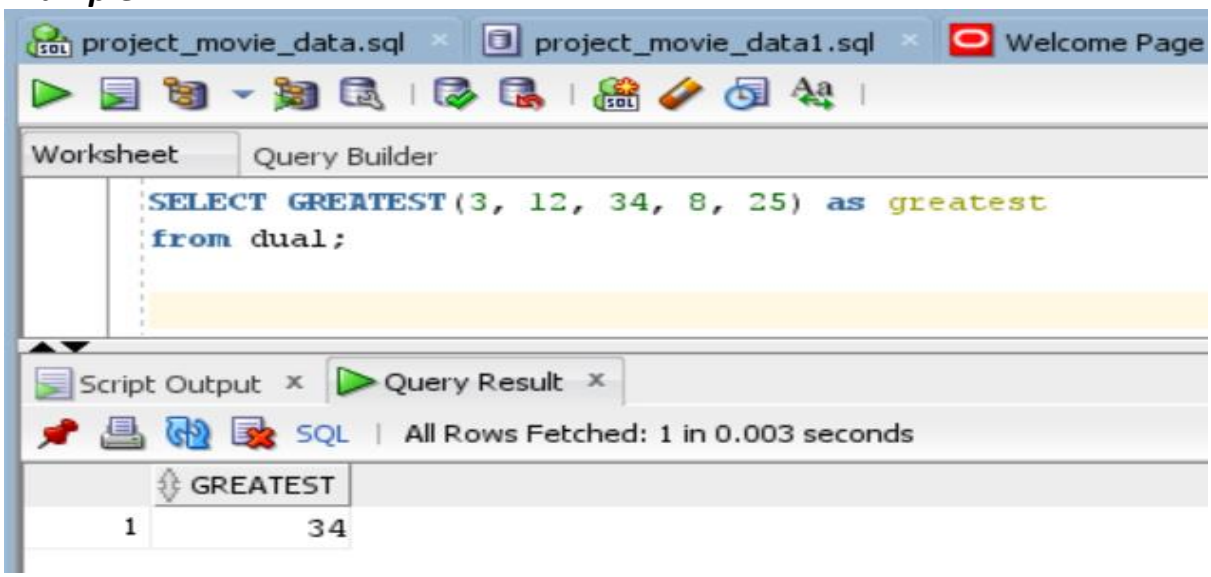
Explanation: A global temporary table called RETAILERS will be created. The next would be the creation of the four columns of the table, namely, the retailer_id, retailer_name, retailer_city and the retailer_state. The first column is of a NUMERIC data type with a maximum limit of 10 digits and it doesn't accept NULL values. The second column is VARCHAR data type with a maximum limit of 30 characters and it also doesn't accept NULL values. The third column is also of VARCHAR datatype with a maximum limit of 50 characters and it does accept NULL values. Similarly, the fourth column is also of VARCHAR datatype with a maximum limit of 50 characters and it also does accept NULL values.

GREATEST function in Oracle

GREATEST is one of the vital Numeric/Math functions of Oracle. It is used to get the greatest value from a list of expressions. The GREATEST function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax: GREATEST(expr_1, expr_2, ... expr_n)
Parameters:
expr_1, expr_2, … expr_n: It is used to specify the expression to be evaluated.

Example :



Explanation:

The greatest value is 34 in the above expression.

GROUP_ID function in Oracle

GROUP_ID is an advanced function that the Oracle database supports. It is used to assign a number to each group resulting from a GROUP BY clause. The GROUP_ID function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

SELECT column1, column2, ... column_n, GROUP_ID()

FROM tables

WHERE conditions

GROUP BY column1, column2, ... column_n;

Example:

```
SELECT SUM(marks), class, extra_marks, GROUP_ID()
FROM students
WHERE extra_marks > 10
GROUP BY class,
ROLLUP (class, extra_marks);
```

Explanation:

The GROUP_ID function will return 0, for each unique group in the result set. It will return a value greater than 0, if a duplicated group is found.

HEXTORAW function in Oracle

HEXTORAW is one of the vital Conversion functions of Oracle. It is used to convert a hexadecimal value to a raw value. The HEXTORAW function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

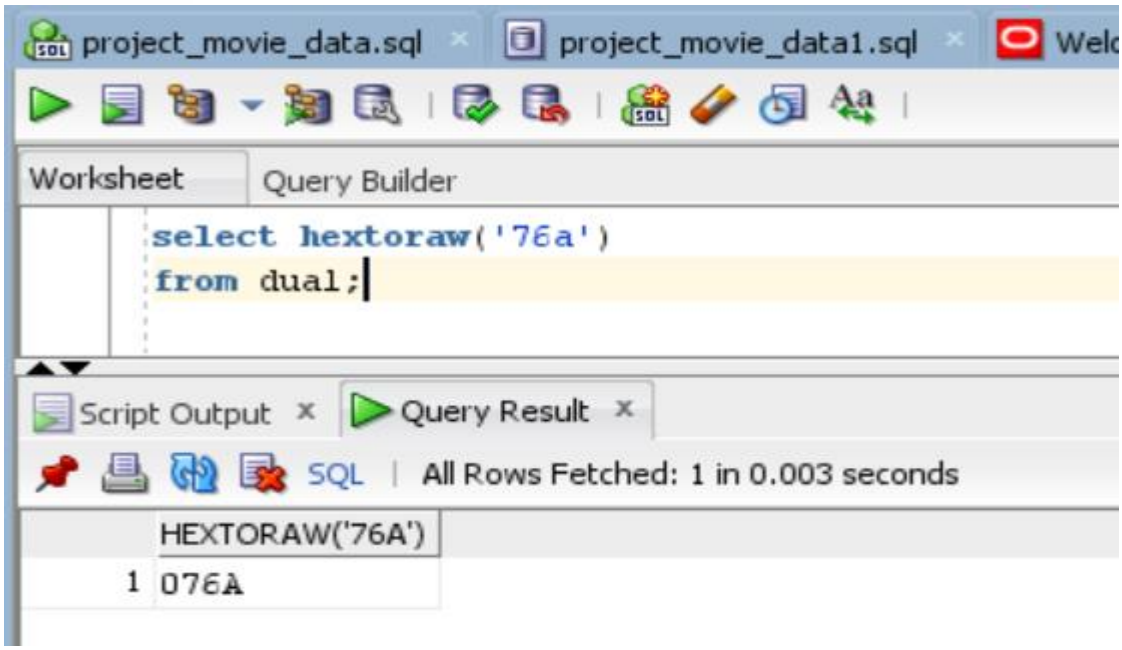
Syntax:

HEXTORAW (char)

Parameters:

char: It is used to specify the hexadecimal value to be converted.

Example :



Explanation:

The result is a raw value of the hexadecimal value passed.

INITCAP function in Oracle

INITCAP is one of the vital string/char functions of Oracle. It is used to set the first character in each word to UPPERCASE. The rest characters are set to lowercase. The INITCAP function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

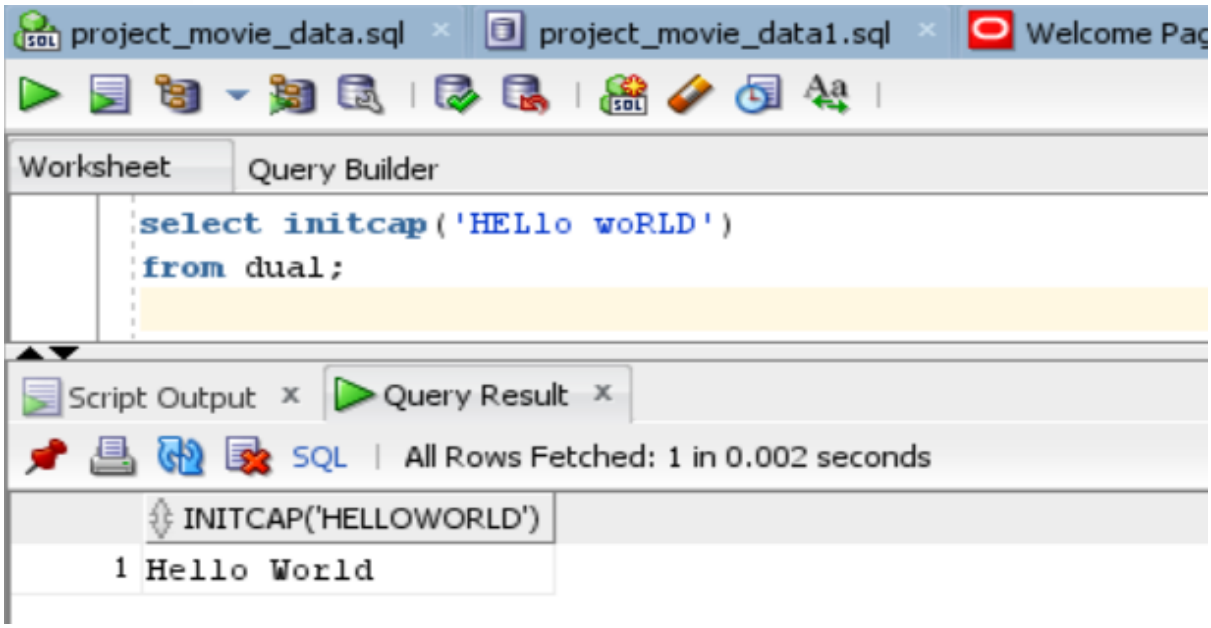
Syntax:

INITCAP(string)

Parameters:

string: : It is used to specify the string to set.

Example :



Explanation:

The first character in each word is left as UPPERCASE and the rest characters are set to lowercase.

INSTR function in Oracle

INSTR is one of the vital string/char functions of Oracle. It is used to get the location of a substring, where a substring is a part of a string. The INSTR function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

INSTR(string, substring, start_position, nth_appearance)

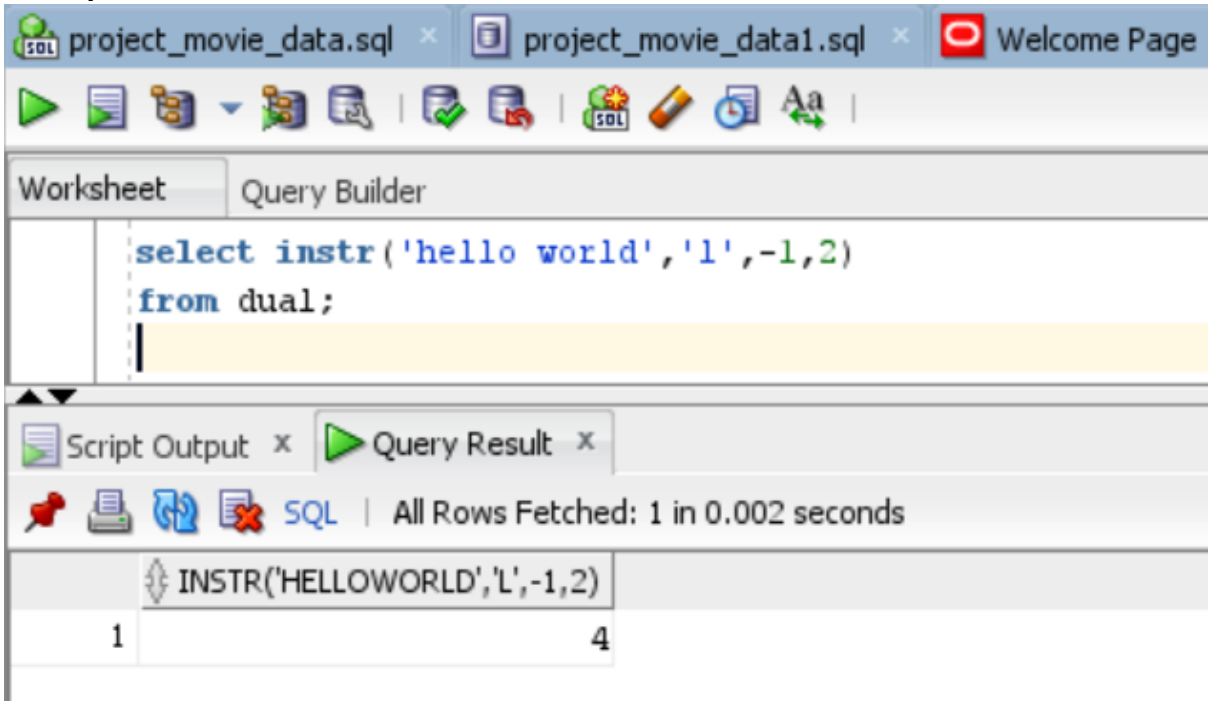
Parameters:

string: It is used to specify the string to set.
substring: It is used to specify the substring to search for.
start_position: It is an optional parameter which is used to specify the position in string where the search will start. Its default value is 1, i.e, the first position. It also

accepts negative value, and in that case it counts back from the end of string and then starts the search backwards towards the beginning of the string. However, whatever be the case the value of the position will be same as counted from the start, i.e, the first position belongs to the first character of the string only the start position is from the beginning or the end.

nth_appearance: It is also an optional parameter which is used to specify the nth appearance of the substring. Its default value is 1.

Example:



Explanation:

The second occurence of 'L' from the ending of the given string is at the fourth position.

INSTR2 function in Oracle

INSTR2 is one of the vital string/char functions of Oracle. It is used to get the location of a substring, where a substring is a part of a string. It works same as INSTR function with the only difference that it uses UCS2 code points. The INSTR2 function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

INSTR2(string, substring, start_position, nth_appearance)

Parameters:

string: It is used to specify the string to set.

substring: It is used to specify the substring to search for.

start_position: It is an optional parameter which is used to specify the position in string where the search will start. Its default value is 1, i.e, the first position. It also accepts negative value, and in that case it counts back from the end of string and then starts the search backwards towards the beginning of the string. However, whatever be the case the value of the position will be same as counted from the start, i.e, the first position belongs to the first character of the string only the start position is from the beginning or the end.

nth_appearance: It is also an optional parameter which is used to specify the nth appearance of the substring. Its default value is 1.

Example 1:

Output:

3

Explanation:

The first occurence of 'L' from the beginning of the given string is at the third position.

VARIANCE function in Oracle

VARIANCE is one of the vital Analytic functions of Oracle. It is used to get the variance of a set of numbers. The VARIANCE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

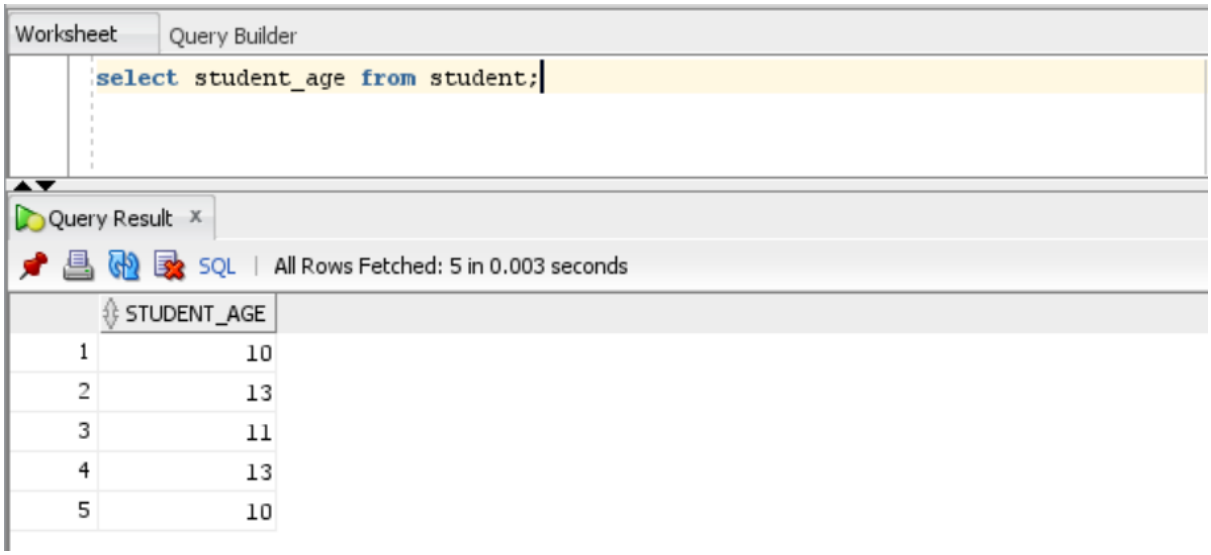
Syntax:

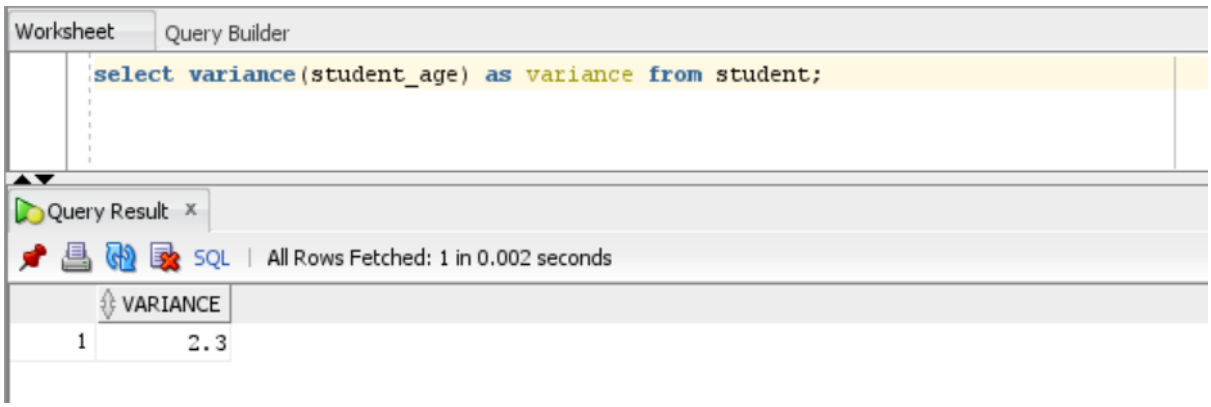
VARIANCE (expression)

Parameters:

expression: It is used to specify the numerical expression to be used for calculation.

Example:





Explanation:

Here, the variance will be returned for the field called ‘student_age’ of the ‘student’ table.

VAR_SAMP function in Oracle

VAR_SAMP is one of the vital Analytic functions of Oracle. It is used to get the sample variance of a set of numbers. The VAR_SAMP function is supported in the various

versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

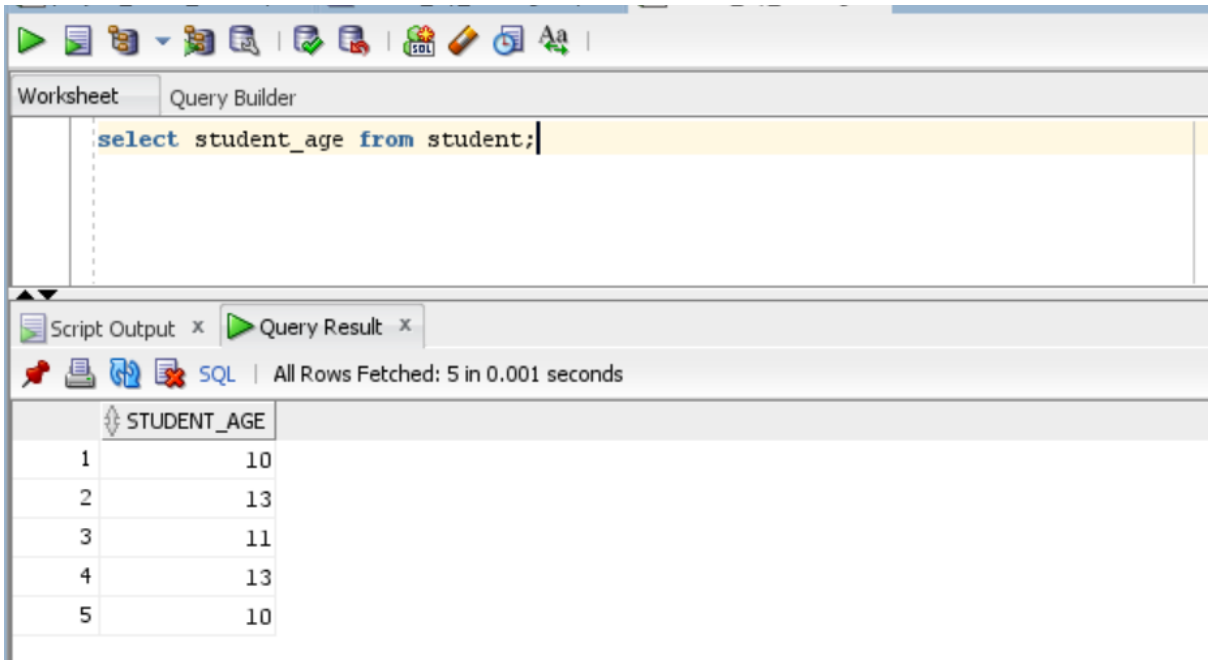
Syntax:

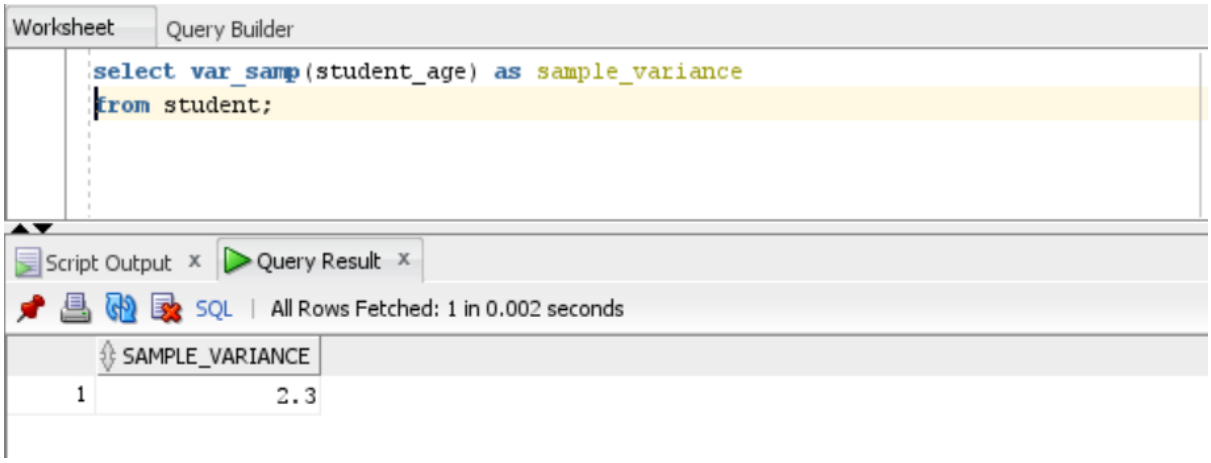
VAR_SAMP (expression)

Parameters:

expression: It is used to specify the numerical expression to be used for calculation.

Example:





Explanation:

Here, the sample variance will be returned for the field called 'student_age' of the 'student' table.

VAR_POP function in Oracle

VAR_POP is one of the vital Analytic functions of Oracle. It is used to get the population variance of a set of numbers. The VAR_POP function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

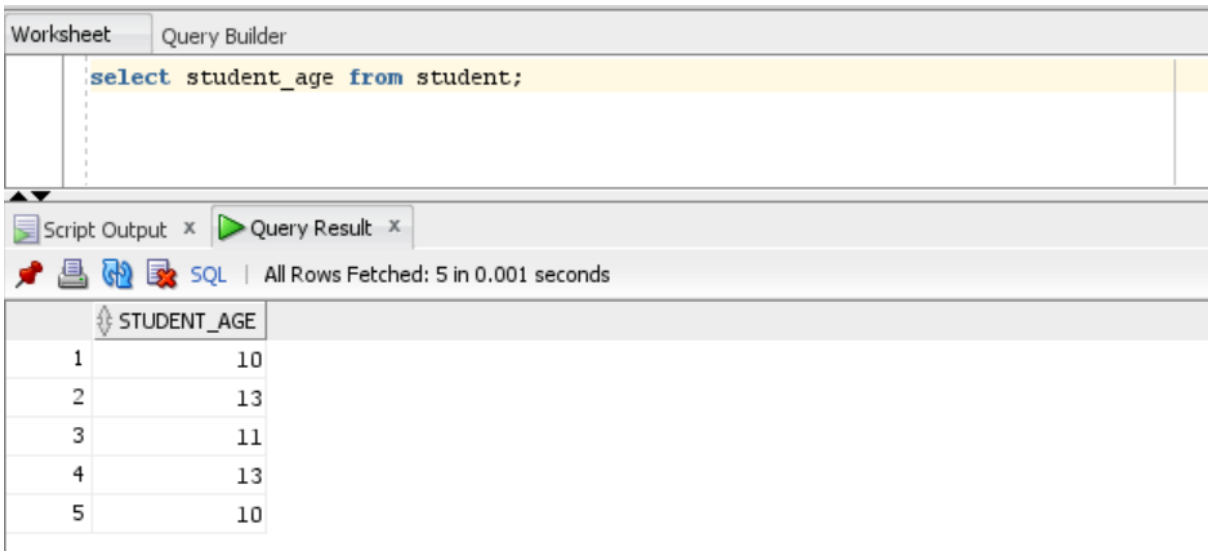
Syntax:

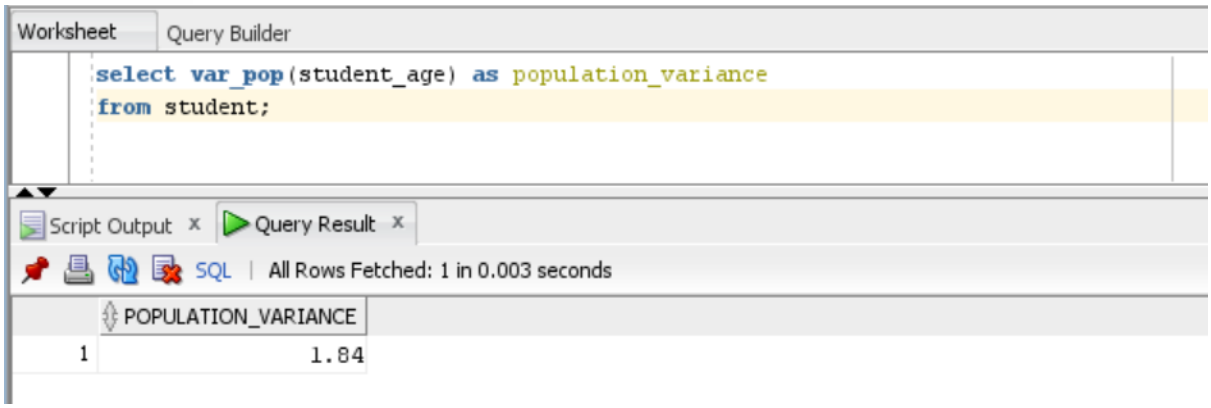
VAR_POP (expression)

Parameters:

expression: It is used to specify the numerical expression to be used for calculation.

Example:





Explanation:

Here, the population variance will be returned for the field called 'marks' of the 'students' table.

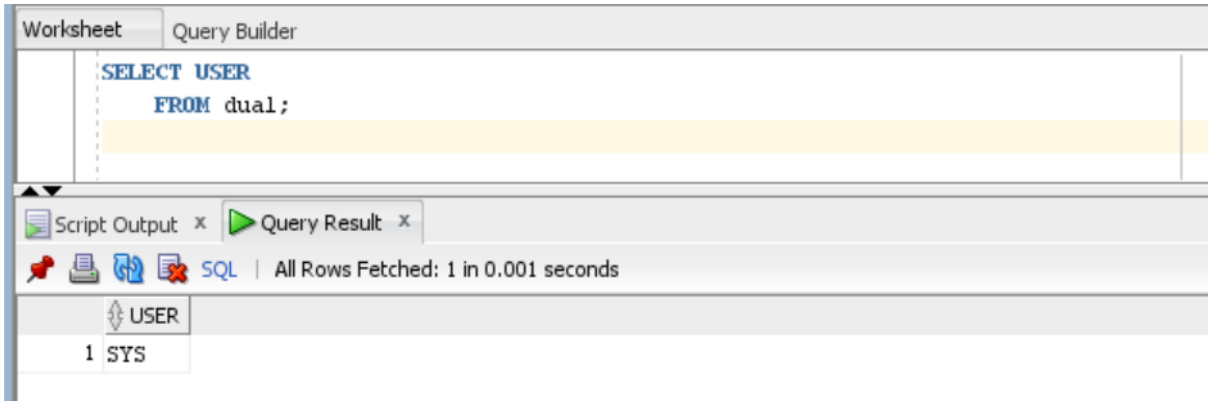
USER function in Oracle

USER is an advanced function that the Oracle database supports. It is used to get the user_id from the current Oracle session. The USER function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

USER

Example:



Explanation:

Here, SYS is the user_id.

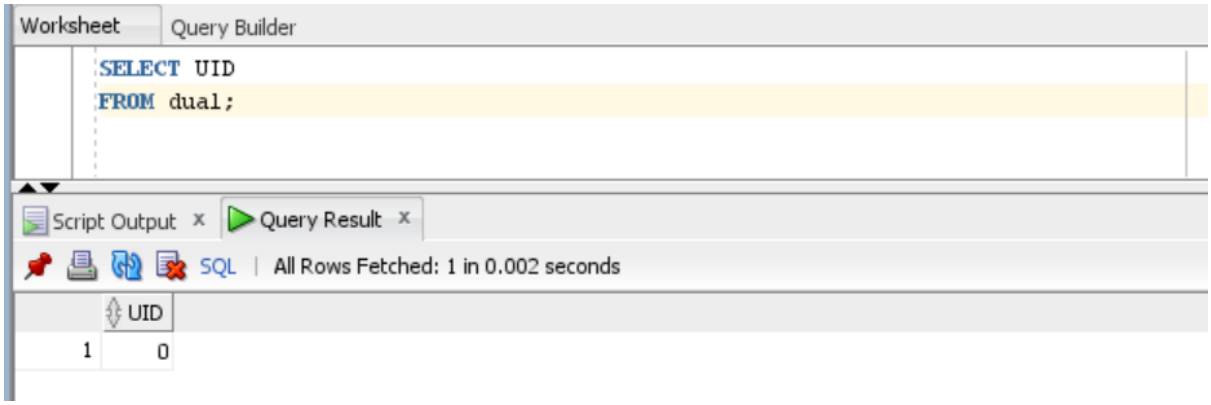
UID function in Oracle

UID is an advanced function that the Oracle database supports. It is used to get the id number for a user's session. The UID function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

UID

Example:



Explanation:

Here, 0 is the id number for user session.

TO_NUMBER Function In Oracle:

TO_NUMBER is one of the vital Conversion functions of Oracle. It is used to convert a string to a number. The TO_NUMBER function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

TO_NUMBER (char, format_mask, nls_language)

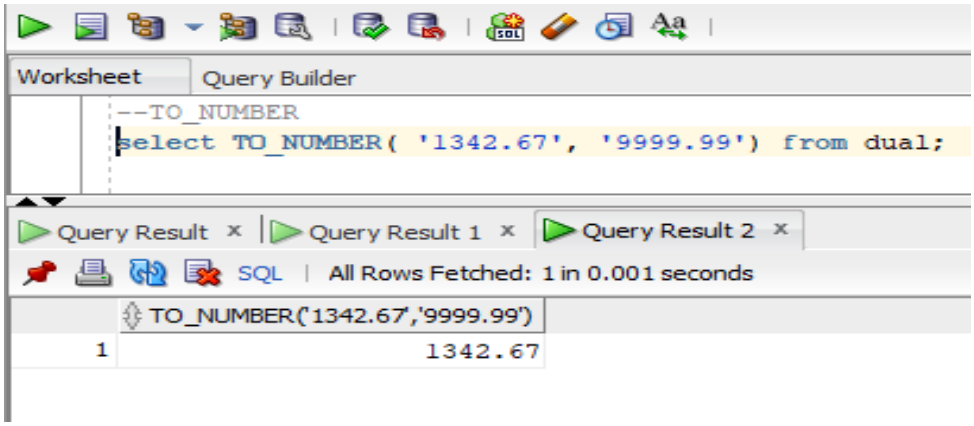
Parameters:

char: It is used to specify the string to be converted.

format_mask: It is an optional parameter which is used to specify the format to be used for conversion.

nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

Example:



Explanation:

Here, a string is converted to a number.

TO_MULTI_BYTE Function In Oracle:

TO_MULTI_BYTE is one of the vital Conversion functions of Oracle. It is used to convert a character value with all of the single-byte characters to multibyte characters. The TO_MULTI_BYTE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

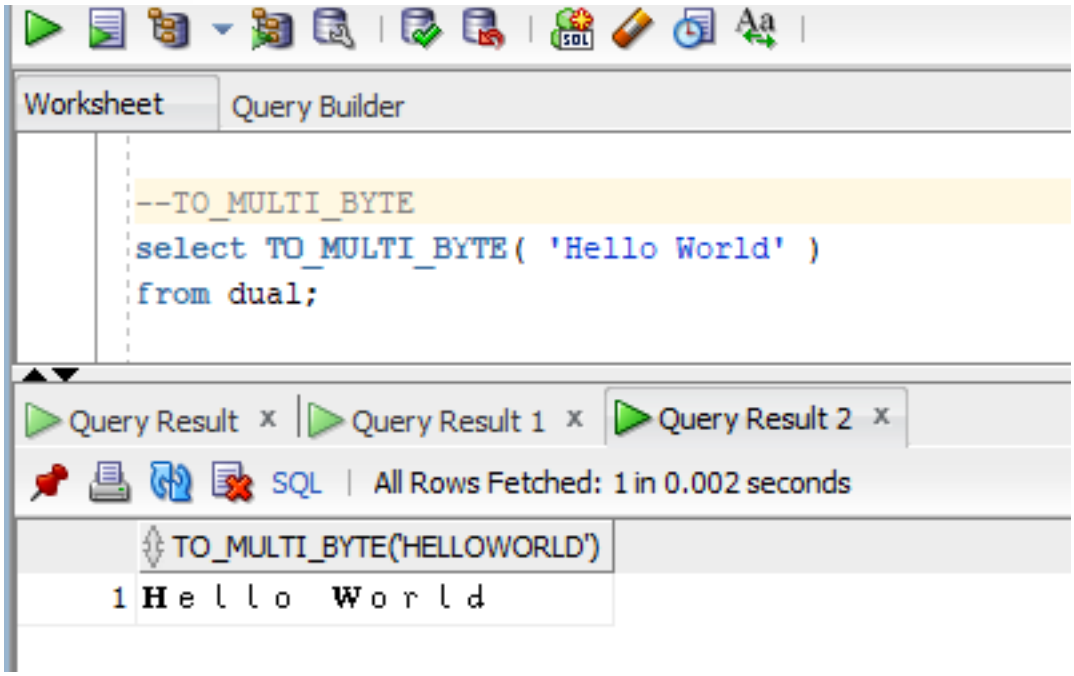
Syntax:

TO_MULTI_BYTE (char)

Parameters:

char: It is used to specify the character value to be converted.

Example:



Explanation:

Here, the result will be a multibyte character value.

TO_DSINTERVAL Function In Oracle:

TO_DSINTERVAL is one of the vital Conversion functions of Oracle. It is used to convert a string to an INTERVAL DAY TO SECOND value. The TO_DSINTERVAL function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.In the SQL format, days is an integer between 0 and 999999999, hours is an integer between 0 and 23, and minutes and seconds are integers between 0 and 59. frac_secs is the fractional part of seconds between .0 and .999999999. One or more blanks separate days from hours. Additional blanks are allowed between format elements.

Syntax:

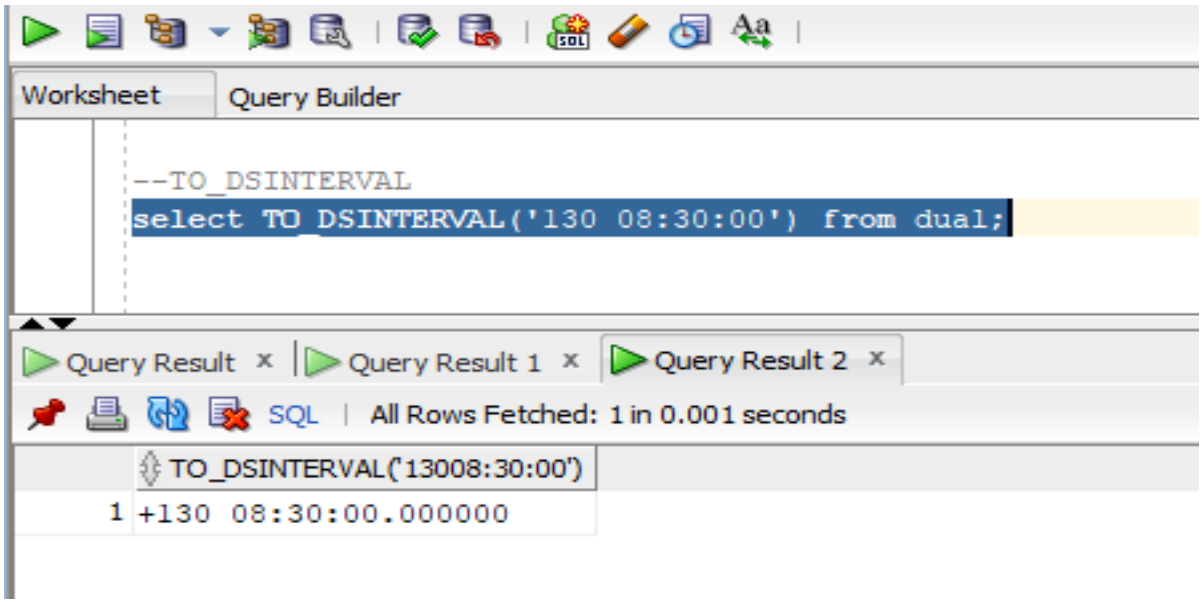
TO_DSINTERVAL (char, nls_language)

Parameters:

char: It is used to specify the string to be converted.

nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

Example1:



Explanation:

Here, a string is converted to an INTERVAL DAY TO SECOND value.

TO_DATE Function In Oracle:

TO_DATE is one of the vital Conversion functions of Oracle. It is used to convert a string to a date. The TO_DATE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

TO_DATE (string, format_mask, nls_language)

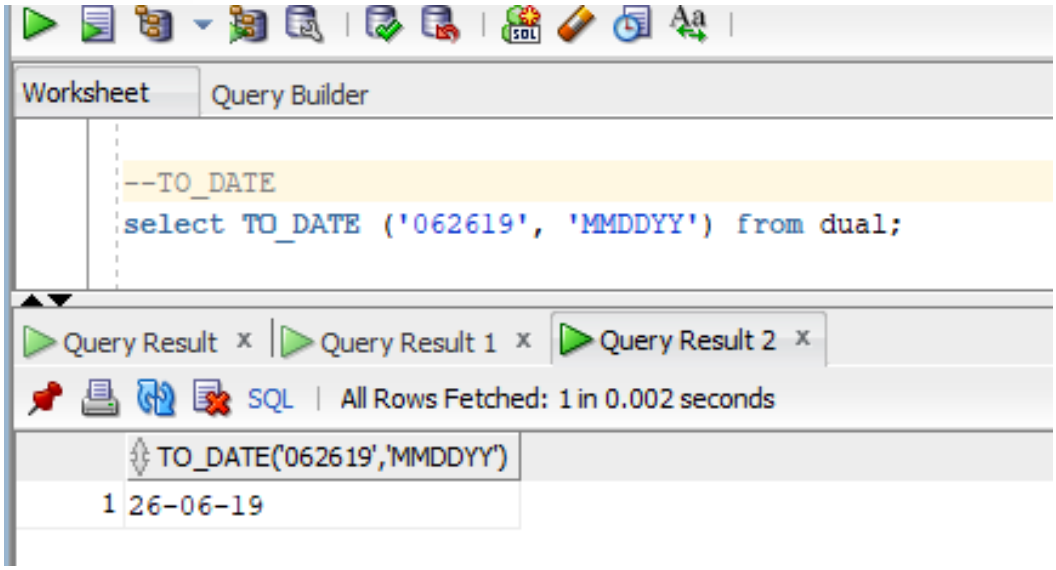
Parameters:

string: It is used to specify the string to be converted.

format_mask: It is an optional parameter which is used to specify the format to be used for conversion.

nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

Example:



Explanation:

Here, a string is converted to a date in the desired format.

TO_CHAR Function In Oracle:

TO_CHAR is one of the vital Conversion functions of Oracle. It is used to convert a number or date to a string. The TO_CHAR function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax: TO_CHAR (value, format_mask, nls_language)

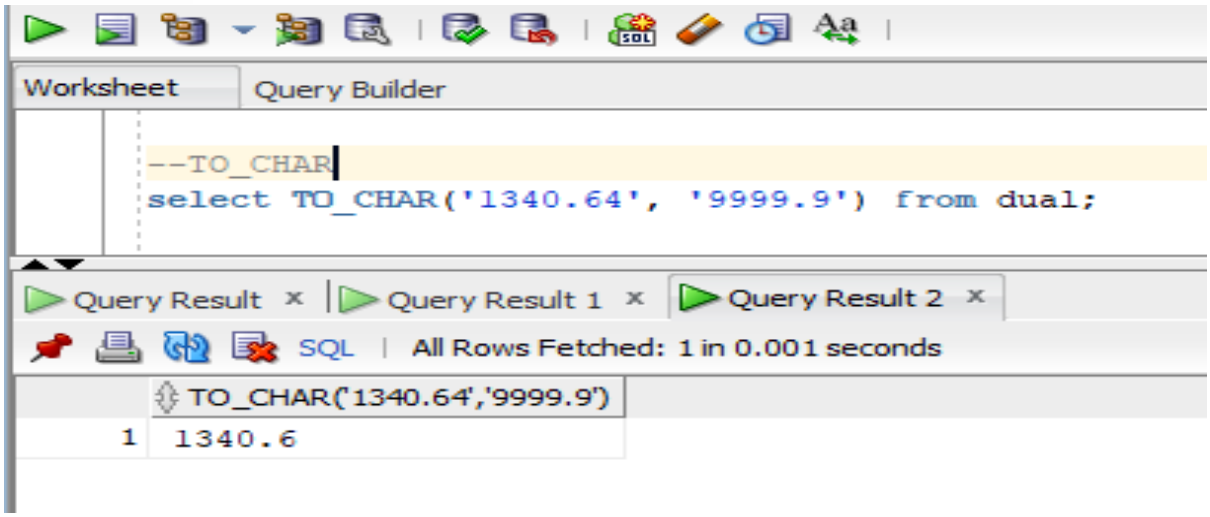
Parameters:

value: It is used to specify the number or date to be converted.

format_mask: It is an optional parameter which is used to specify the format to be used for conversion.

nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

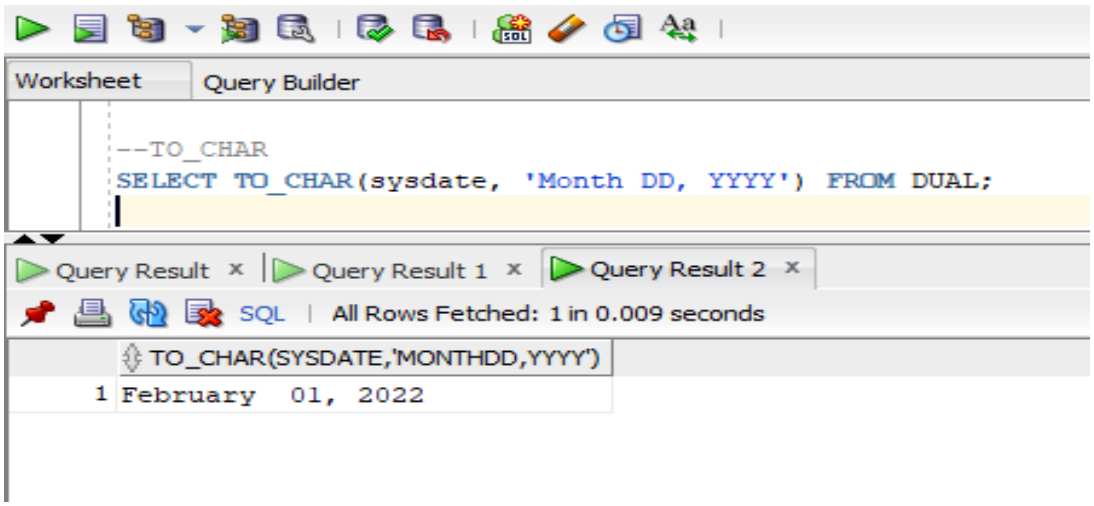
Example 1:



Explanation:

Here, a number is converted to a string in the desired format.

Example 2:



Explanation:

Here, a date value is converted to a string in the desired format.

TANH Function In Oracle:

TANH is one of the vital Numeric/Math functions of Oracle. It is used to get the hyperbolic tangent of a number. The TANH function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

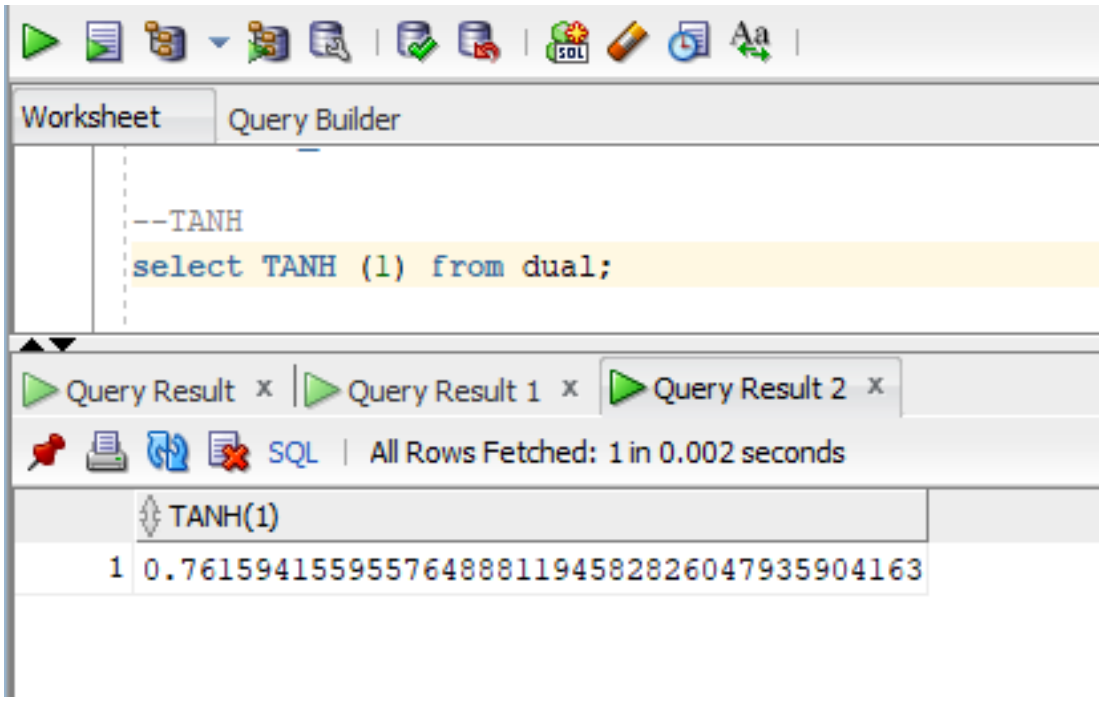
Syntax:

TANH (number)

Parameters:

number: It is used to specify the number to get the hyperbolic tangent value of.

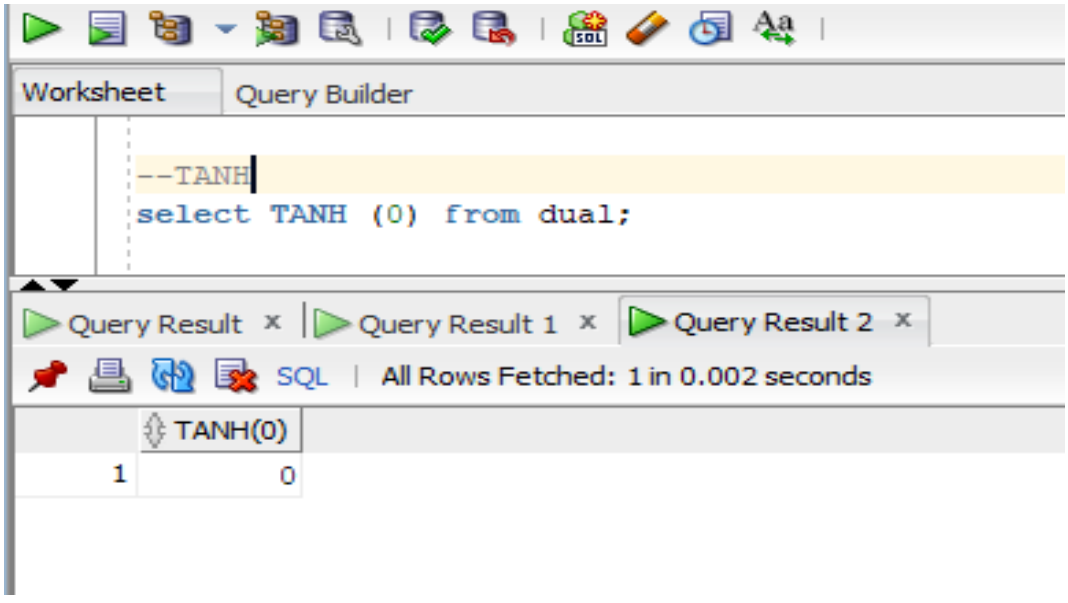
Example 1:



Explanation:

The hyperbolic tangent value of 1 is 0.76159415595576, and so is the result.

Example 2:



Explanation:

The hyperbolic tangent value of 0 is 0, and so is the result.

TAN Function In Oracle:

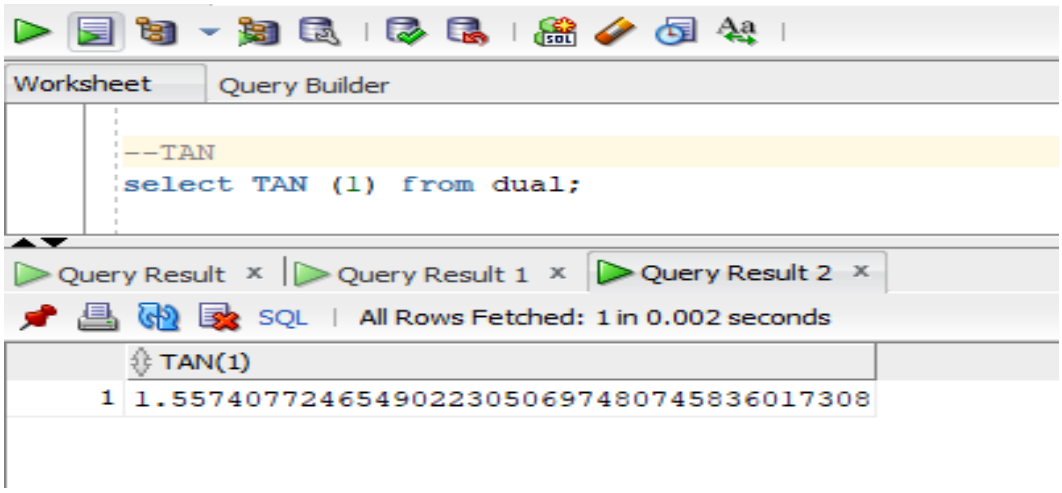
TAN is one of the vital Numeric/Math functions of Oracle. It is used to get the tangent of a number. The TAN function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax: TAN (number)

Parameters:

number: It is used to specify the number in radians to get the tangent value of.

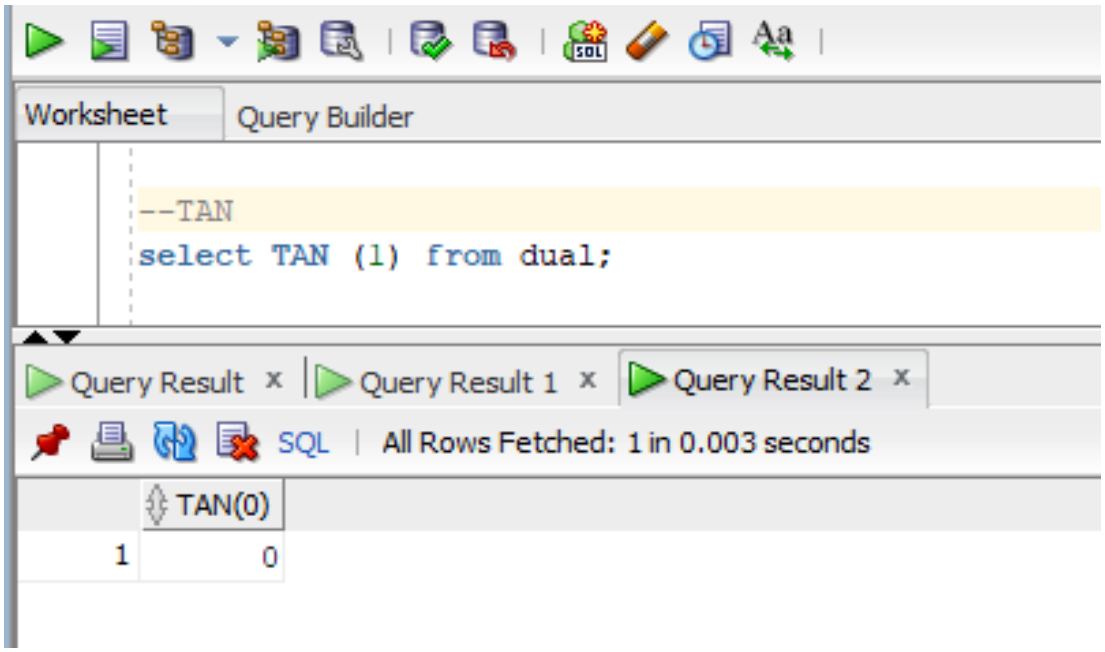
Example 1:



Explanation:

The tangent value of 1 is 1.55740772465, and so is the result.

Example 2:



Explanation:

The tangent value of 0 is 0, and so is the result.

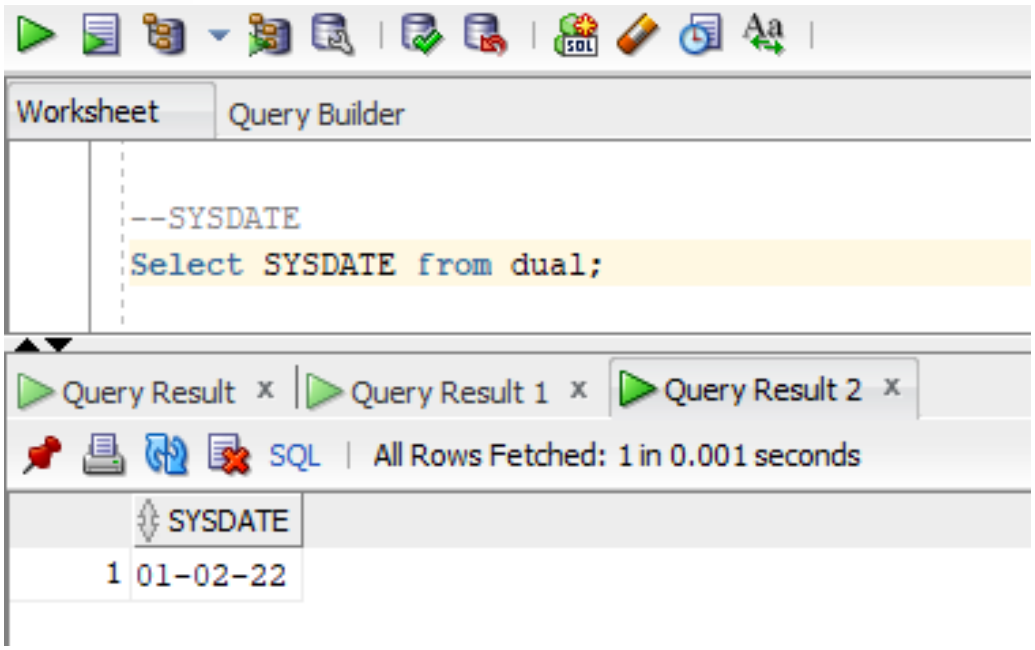
SYSDATE Function In Oracle:

SYSDATE is one of the vital Date/Time functions of Oracle. It is used to get the current system date and time on the local database. The result generated is either as a time zone offset or a time zone region name. The SYSDATE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

SYSDATE

Example:



Explanation:

The var is a variable that will contain the system's date and time at the moment of execution.

SQRT Function In Oracle:

SQRT is one of the vital Numeric/Math functions of Oracle. It is used to get the square root of a number. The SQRT function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

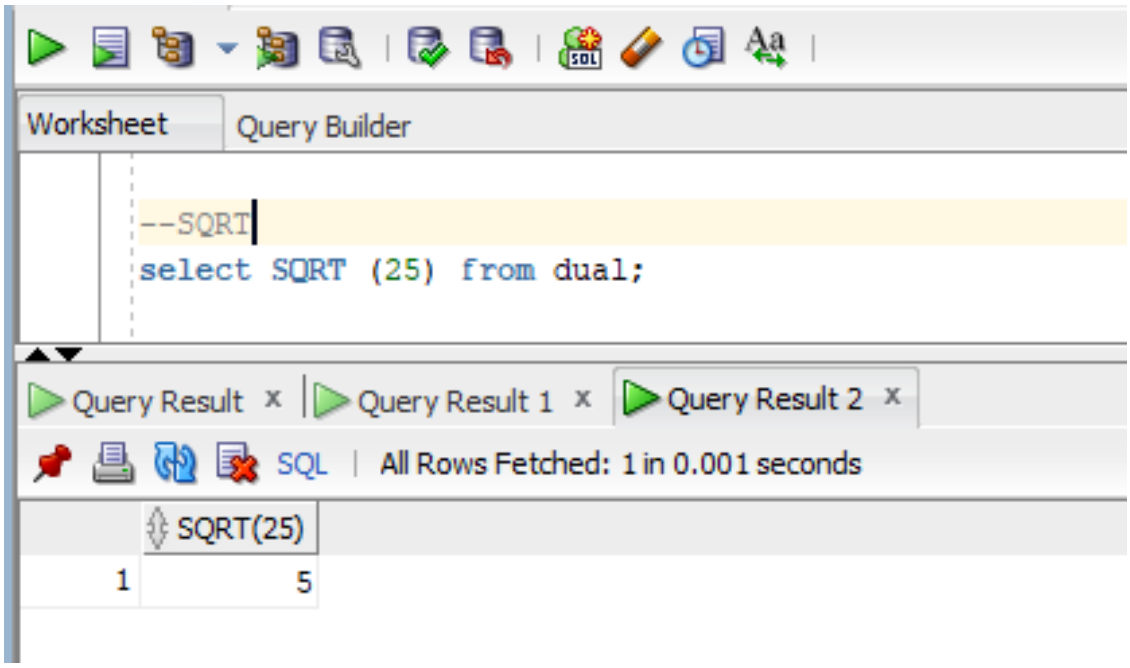
SQRT (number)

Parameters:

number: It is used to specify the number to get the square root value of.

Example:

11-Feb-2022



Explanation:

The square root of 25 is 5, and so is the result.

LOG Function In Oracle:

LOG is one of the vital Numeric/Math functions of Oracle. It is used to get the logarithm of x base y. The LOG function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

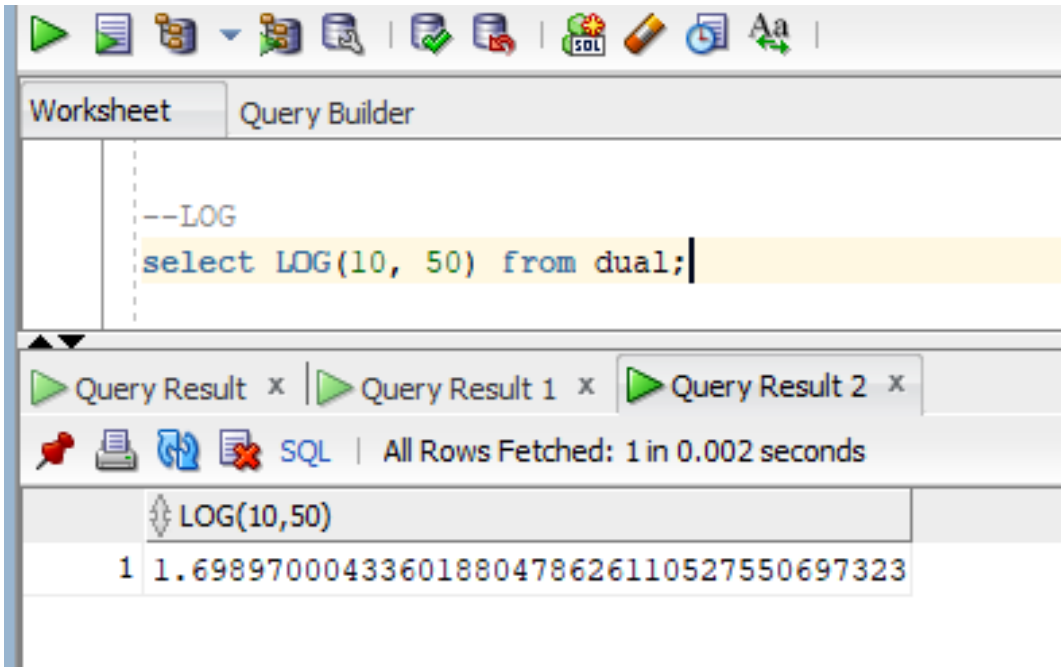
Syntax:

LOG(y, x)

Parameters:

y: It must be a positive number except 1 or 0.
x: It must be a positive number.

Example 1:



Explanation:

The value of base 10 logarithm of 50 is 1.6989700043 and so is the result.

LAST_DAY Function In Oracle:

LAST_DAY is one of the vital Date/Time functions of Oracle. It is used to get the last day of the month. The LAST_DAY function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

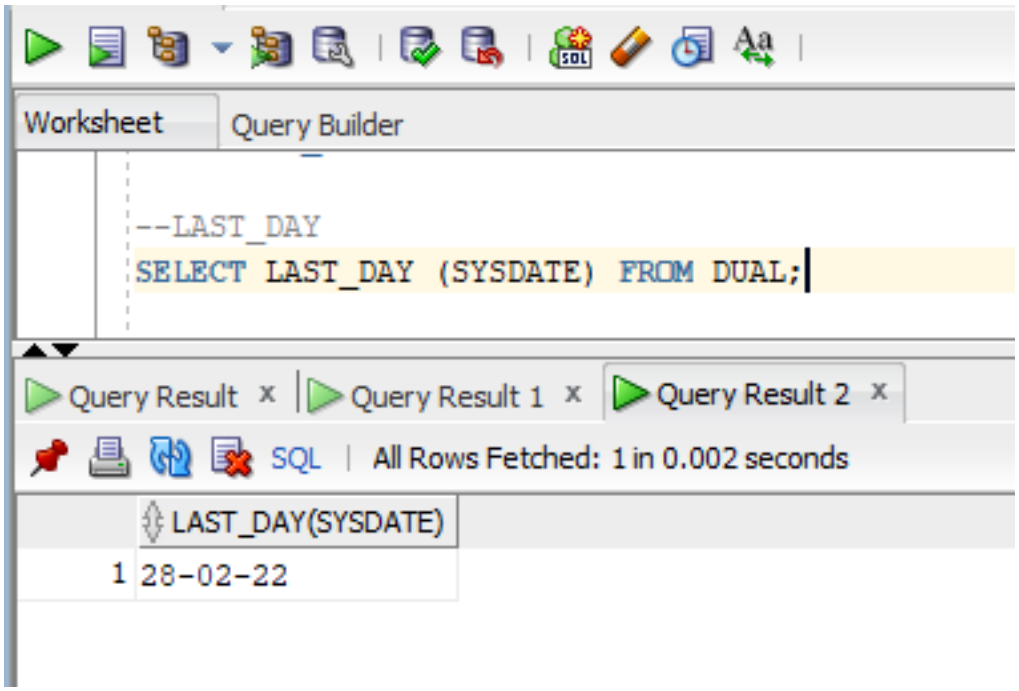
Syntax:

LAST_DAY (date)

Parameters:

date: It is used to specify the date to be used for evaluation.

Example:



Explanation:

The last day for the month of June in the year 2019 is the 30th of June.

LEAST Function In Oracle:

LEAST is one of the vital Numeric/Math functions of Oracle. It is used to get the least value from a list of expressions. The LEAST function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

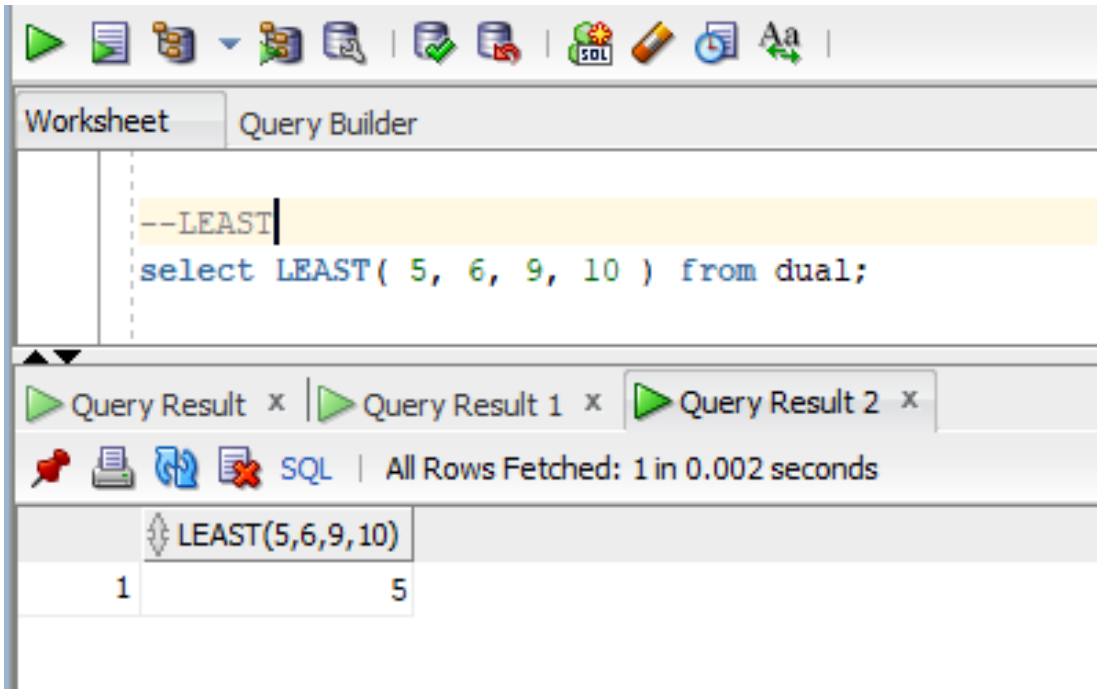
Syntax:

LEAST(expr_1, expr_2, ... expr_n)

Parameters:

expr_1, expr_2, … expr_n: It is used to specify the expression to be evaluated.

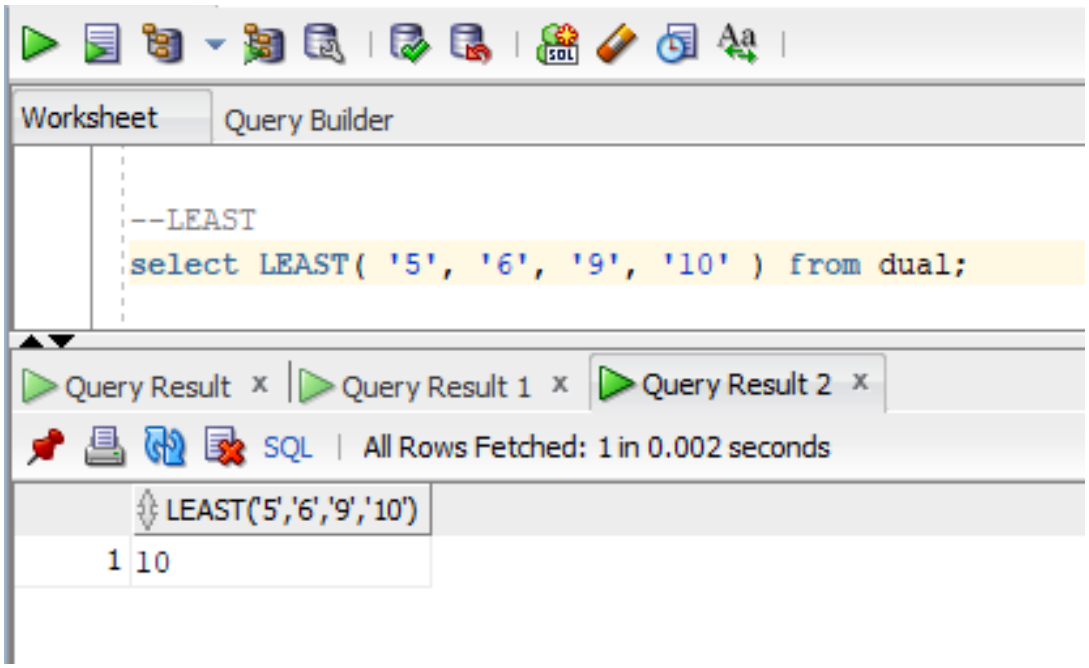
Example 1:



Explanation:

The LEAST value is 5 in the above expression.

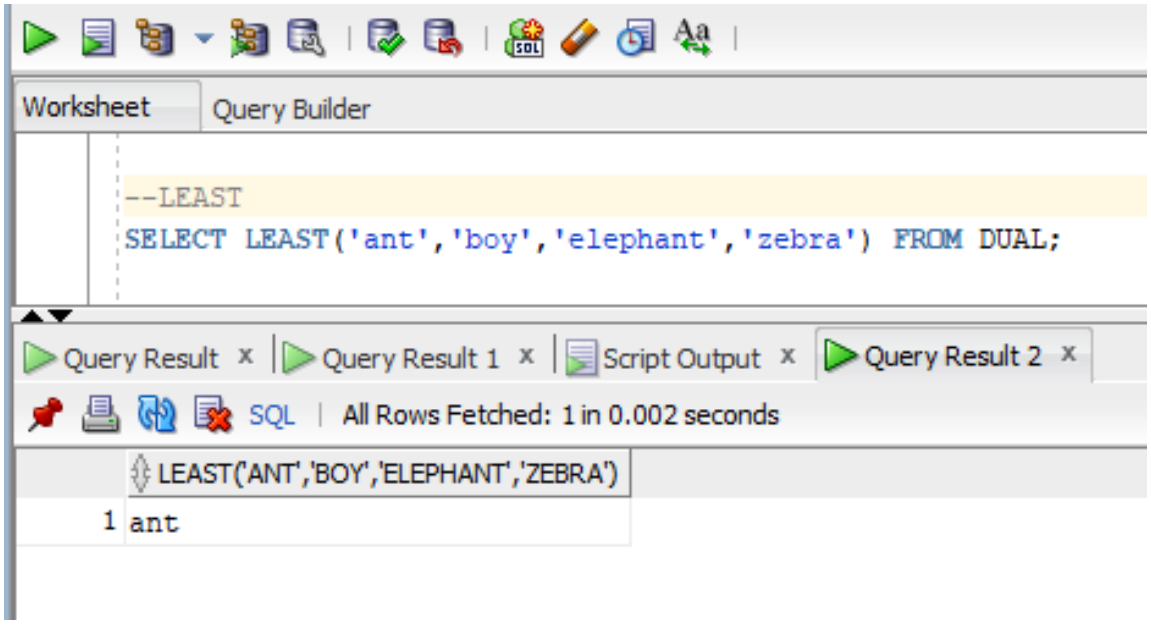
Example 2:



Explanation:

Since it is a character comparison, hence '10' here is LEAST since it has a lower character set value.

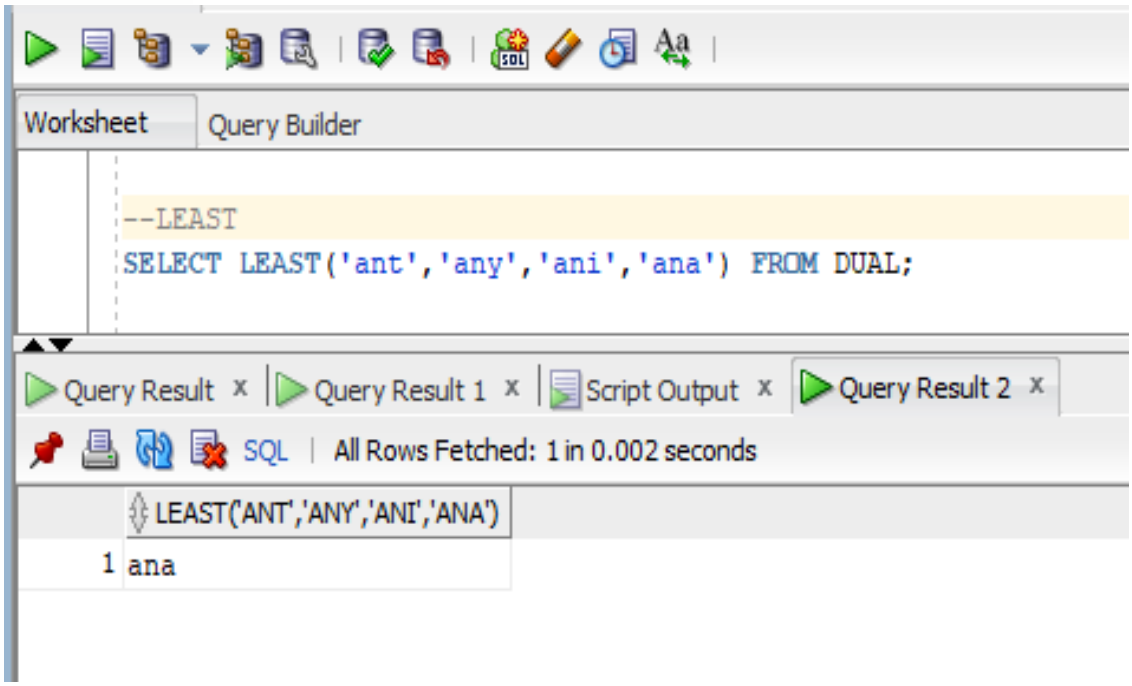
Example 3:



Explanation:

Since it is a character comparison, hence 'ant' here is LEAST since it has a lower character set value.

Example 4:



Explanation:

Since it is a character comparison, hence 'ana' here is LEAST since it has a lower character set value.

LENGTH Function In Oracle:

LENGTH is one of the vital string/char functions of Oracle. It is used to get the length of a string. The LENGTH function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

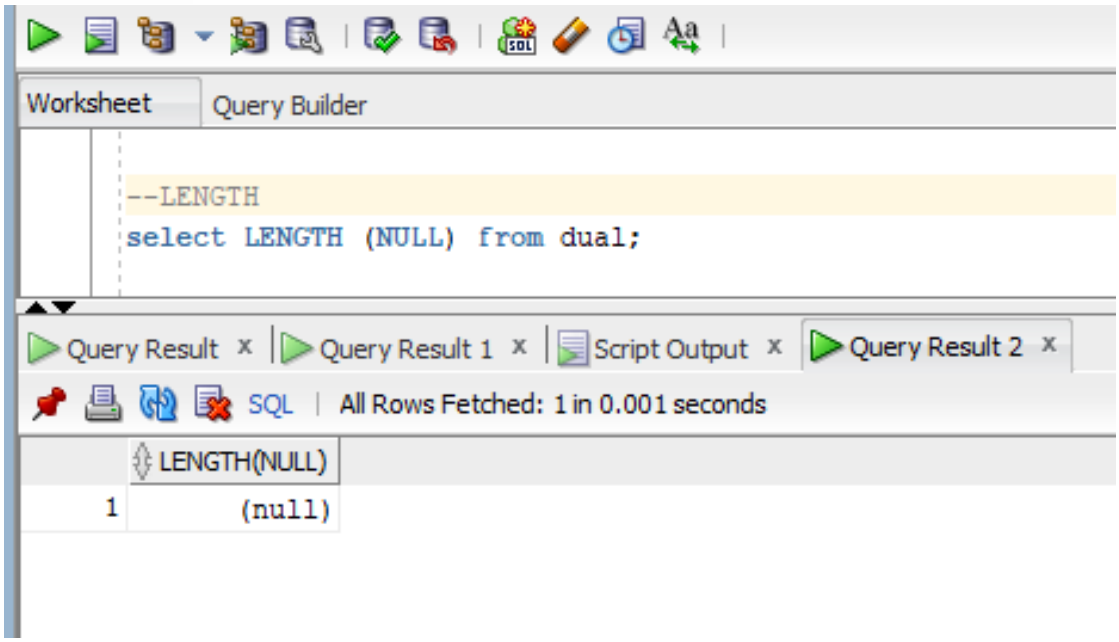
Syntax:

LENGTH (string)

Parameters:

string: It is used to specify the string whose length needs to get.

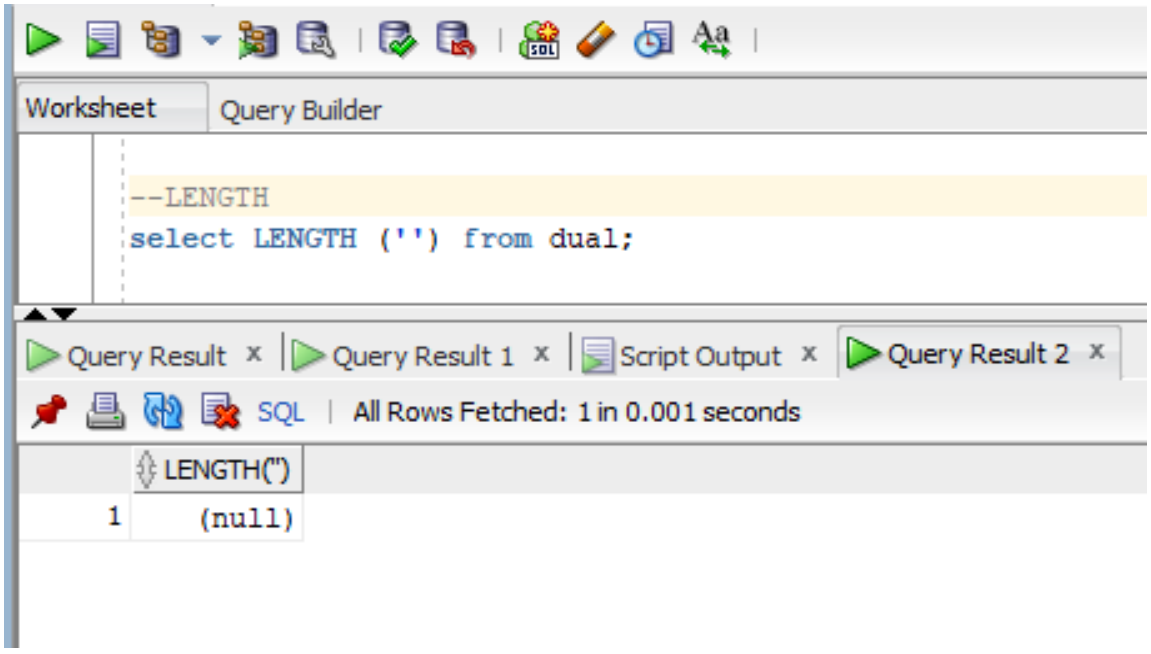
Example 1:



Explanation:

There is no string passed hence the result is NULL.

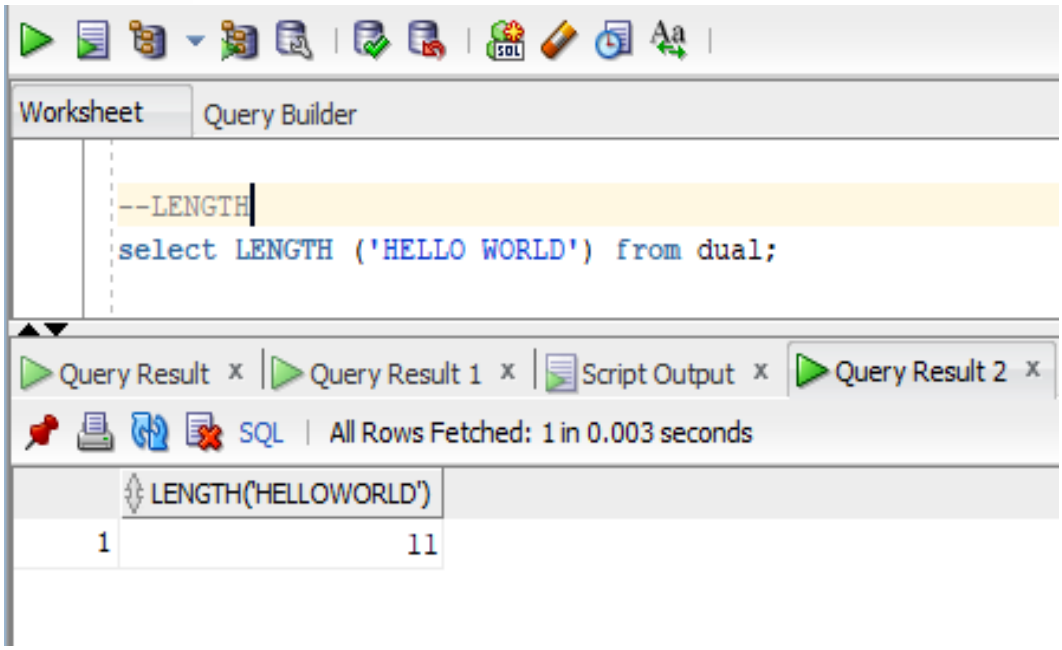
Example 2:



Explanation:

There is no string passed hence the result is NULL.

Example 3:



Explanation:

The total number of characters including the SPACE is 11 and so is the result..

INSTR4 Function In Oracle:

INSTR4 is one of the vital string/char functions of Oracle. It is used to get the location of a substring, where a substring is a part of a string. It works same as INSTR and the INSTR2 function with the only difference that it uses UCS4 code points. The INSTR4 function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

INSTR4(string, substring, start_position, nth_appearance)

Parameters:

string: It is used to specify the string to set.

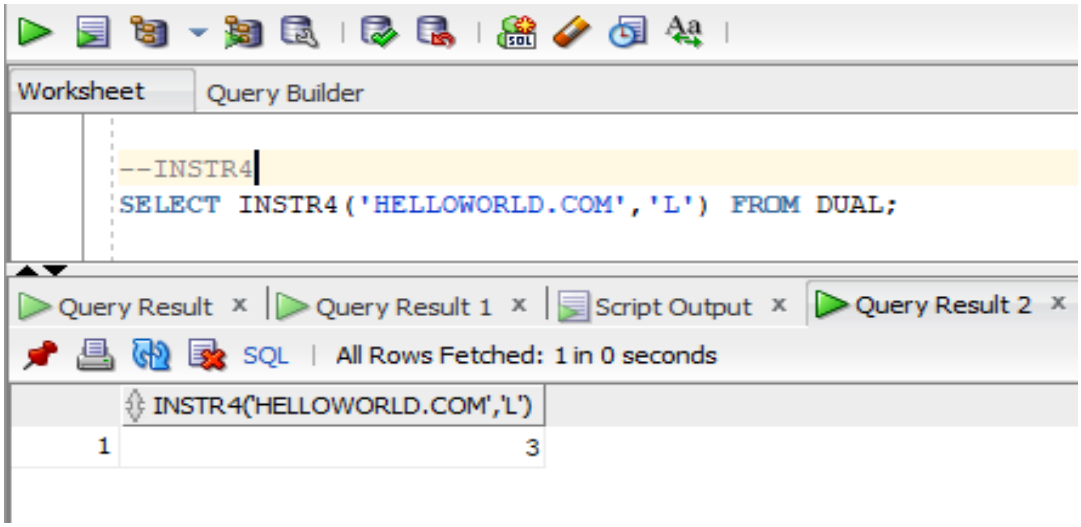
substring: It is used to specify the substring to search for.

start_position: It is an optional parameter which is used to specify the position in string where the search will start. Its default value is 1, i.e, the first position. It also accepts negative value, and in that case it counts back from the end of string and then starts the search backwards towards the beginning of the string. However,

whatever be the case the value of the position will be same as counted from the start, i.e, the first position belongs to the first character of the string only the start position is from the beginning or the end.

nth_appearance: It is also an optional parameter which is used to specify the nth appearance of the substring. Its default value is 1.

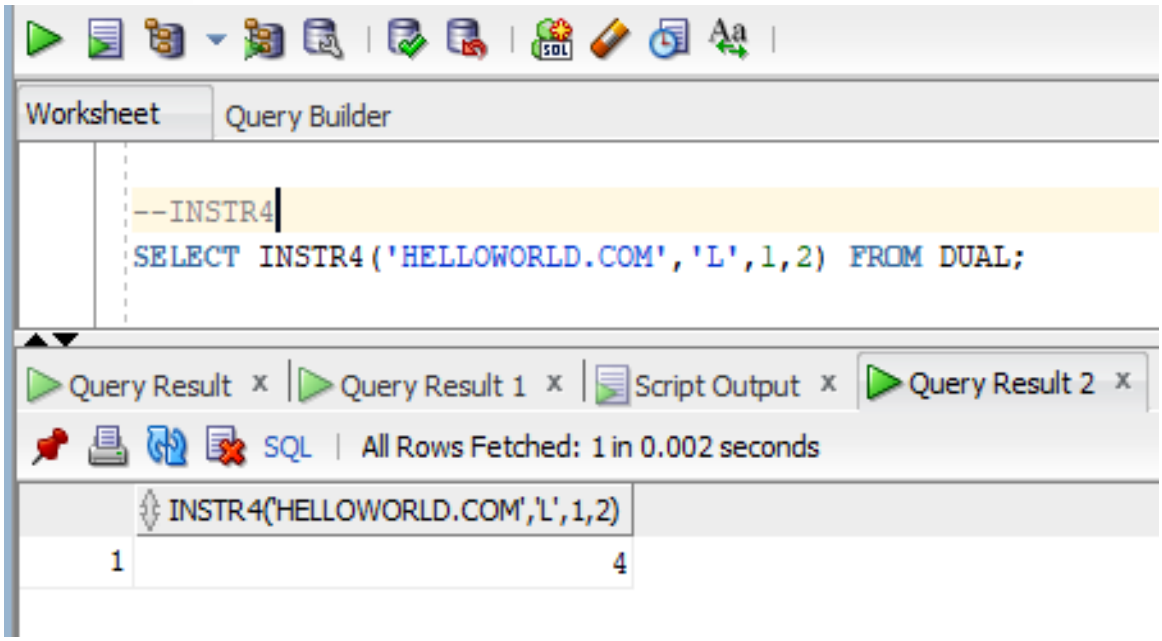
Example 1:



Explanation:

The first occurence of 'L' from the beginning of the given string is at the third position.

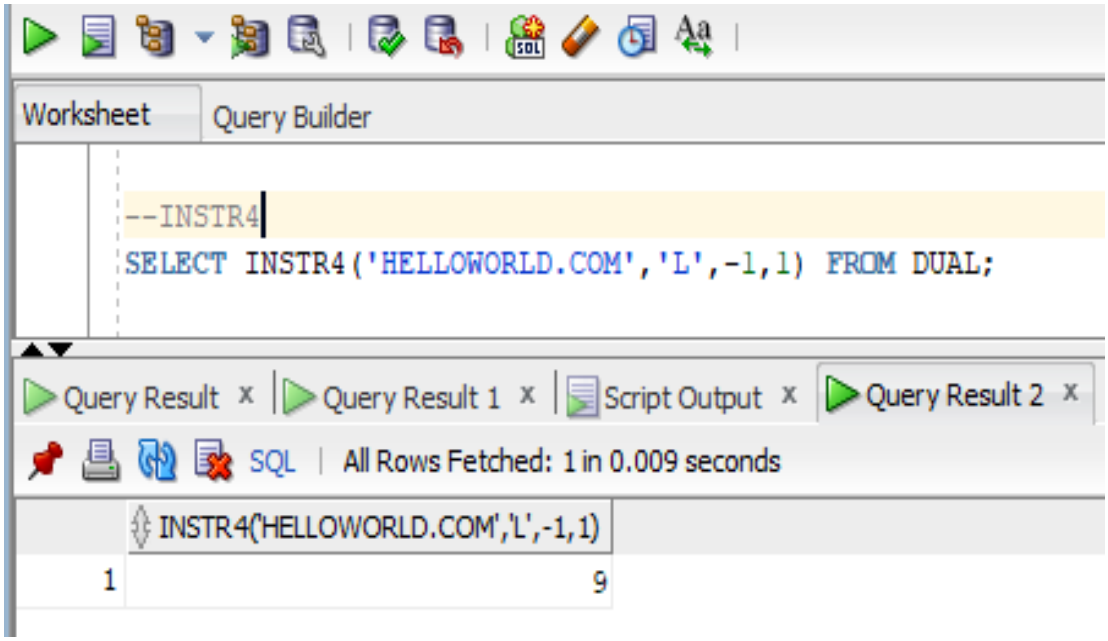
Example 2:



Explanation:

The second occurence of 'L' from the beginning of the given string is at the fourth position.

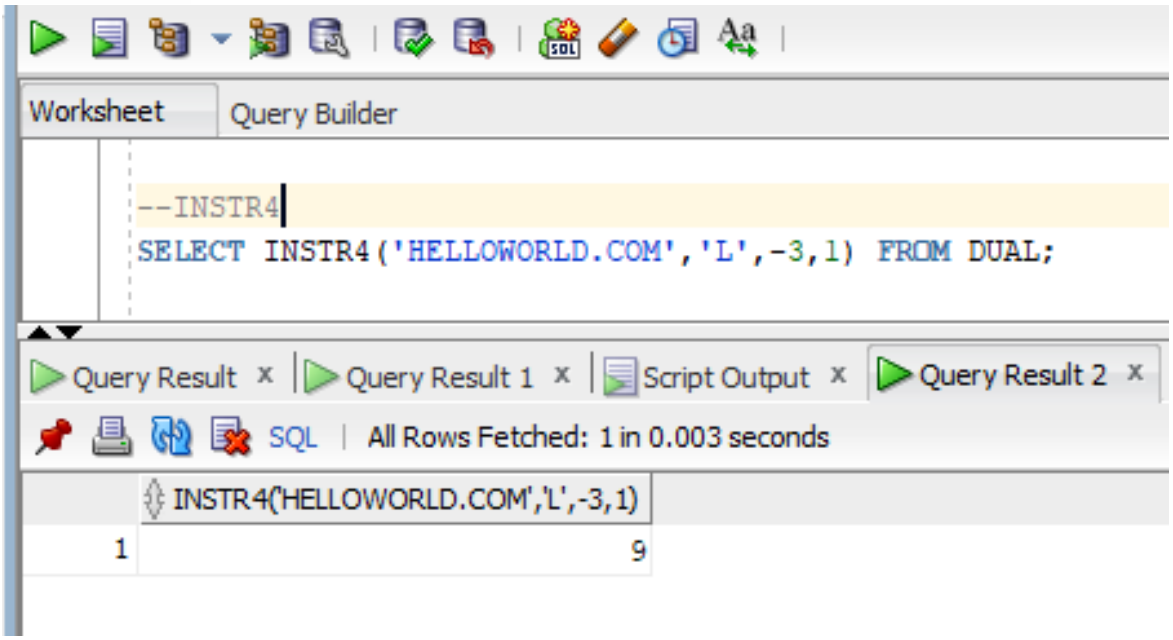
Example 3:



Explanation:

The first occurence of 'L' from the end of the given string is at the ninth position.

Example 4:



Explanation:

The first occurence of 'L' from the third position from the end of the given string is at the ninth position. Here the counting started from the complete end of the string.

TZ_OFFSET function in Oracle

TZ_OFFSET is one of the vital Date/Time functions of Oracle. It is used to get the time zone offset of a value. The TZ_OFFSET function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TZ_OFFSET (timezone)

Parameters:

timezone: It is used to specify a valid time zone name.

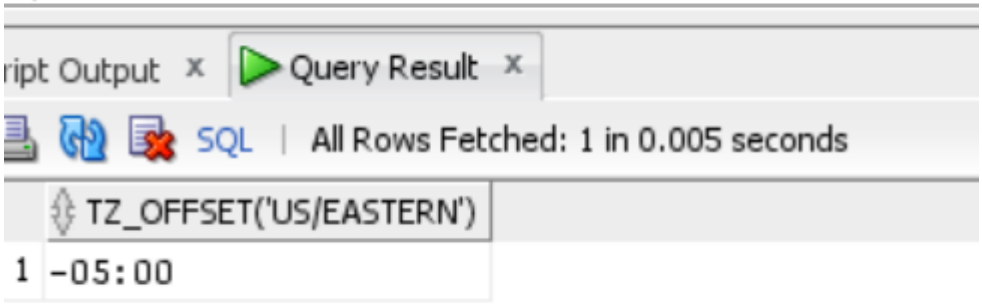
Example 1:

TZ_OFFSET('US/Michigan')

Output:

'-05:00'

```
SELECT TZ_OFFSET('US/Eastern') FROM DUAL;
```



TRUNC (numbers) function in Oracle

TRUNC (numbers) is one of the vital Numeric/Math functions of Oracle. It is used to get a number truncated to a certain number of decimal places. The TRUNC (numbers) function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

TRUNC (number, decimal_places)

Parameters:

number: It is used to specify the numbers to be truncated.

decimal_places: It is an optional parameter which is used to specify the number of decimal places to be truncated to.

Example 1:

TRUNC (10.789)

Output:

10

```
SELECT TRUNC(15.79,1) "Truncate" FROM DUAL;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.003 seconds

	Truncate
1	15.7

Explanation:

By default the TRUNC (numbers) function returns the nearest integer value if the decimal_places parameter is not specified.

TRUNC (dates) function in Oracle

TRUNC (dates) is one of the vital Date/Time functions of Oracle. It is used to get a date truncated to a specific unit of measure. The TRUNC (dates) function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

TRUNC (date, format)

Parameters:

date: It is used to specify the date to be truncated.

format: It is an optional parameter which is used to specify the unit of measure to be used for truncating.

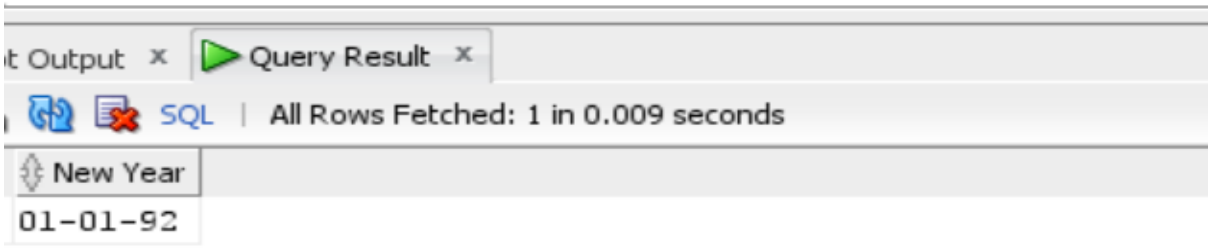
Example 1:

TRUNC(TO_DATE ('22-AUG-19'),'YEAR')

Output:

'01-JAN-19'

```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'), 'YEAR')
  "New Year" FROM DUAL;
```



Explanation:

The result is the truncated date as per the YEAR format.

TO_YMINTERVAL function in Oracle

TO_YMINTERVAL is one of the vital Conversion functions of Oracle. It is used to convert a string to an INTERVAL YEAR TO MONTH value. The TO_YMINTERVAL function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TO_YMINTERVAL (char)

Parameters:

char: It is used to specify the string to be converted.

Example:

TO_YMINTERVAL('09-12')

Output:

9 years 12 months

Explanation:

Here, a string is converted to an INTERVAL YEAR TO MONTH value.

TO_TIMESTAMP_TZ function in Oracle

TO_TIMESTAMP_TZ is one of the vital Conversion functions of Oracle. It is used to convert a string to a timestamp with time zone. The TO_TIMESTAMP_TZ function is

supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TO_TIMESTAMP_TZ (char, format_mask, nls_language)

Parameters:

char: It is used to specify the string to be converted.

format_mask: It is an optional parameter which is used to specify the format to be used for conversion.

nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

Example:

TO_TIMESTAMP_TZ('2019/06/28 10:16:28 -7:00', 'YYYY/MM/DD HH:MI:SS TZH:TZM')

Output:

'28-JUN-19 10.16.28.000000000 AM -7:00'

```
SELECT TO_TIMESTAMP_TZ('1999-12-01 11:00:00 -8:00',
'YYYY-MM-DD HH:MI:SS TZH:TZM') FROM DUAL;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.002 seconds

	TO_TIMESTAMP_TZ('1999-12-0111:00:00-8:00','YYYY-MM-DDHH:MI:SSTZH:TZM')
1	01-12-99 11:00:00.000000000 AM -08:00

Explanation:

Here, a string is converted to a timestamp with time zone value.

TO_TIMESTAMP function in Oracle

TO_TIMESTAMP is one of the vital Conversion functions of Oracle. It is used to convert a string to a timestamp. The TO_TIMESTAMP function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TO_TIMESTAMP (char, format_mask, nls_language)

Parameters:

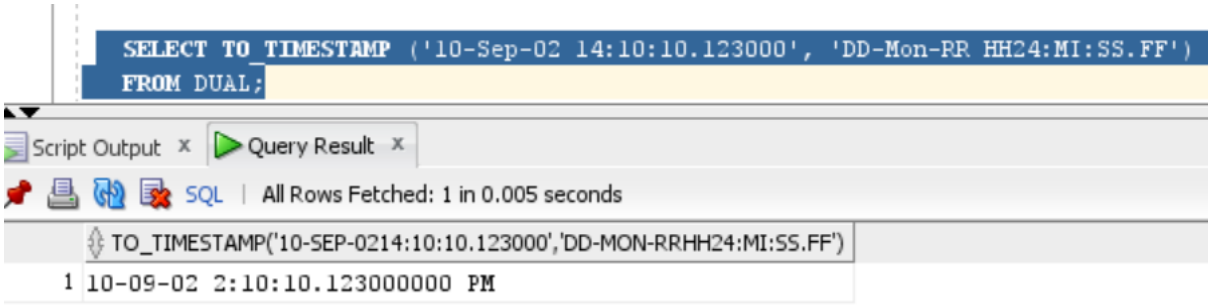
char: It is used to specify the string to be converted.
format_mask: It is an optional parameter which is used to specify the format to be used for conversion.
nls_language: It is an optional parameter which is used to specify the nls language to be used for conversion.

Example:

TO_TIMESTAMP('2019/06/28 10:16:28', 'YYYY/MM/DD HH:MI:SS')

Output:

'28-JUN-19 10.16.28.000000000 AM'



Explanation:

Here, a string is converted to a timestamp value.

TO_SINGLE_BYTE function in Oracle

TO_SINGLE_BYTE is one of the vital Conversion functions of Oracle. It is used to convert a character value with all of the multi-byte characters to single-byte

characters. The TO_SINGLE_BYTE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

TO_SINGLE_BYTE (char)

Parameters:

char: It is used to specify the character value to be converted.

Example:

```
select TO_SINGLE_BYTE( 'Hello World' )
from dual
```

```
SELECT TO_SINGLE_BYTE( CHR(15711393)) FROM DUAL;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.002 seconds

	TO_SINGLE_BYTE(CHR(15711393))
1	A

Explanation:

Here, the result will be a single-byte character value.

TO_NCLOB Function In Oracle:

TO_NCLOB is one of the vital Conversion functions of Oracle. It is used to convert a LOB value to a NCLOB value. The TO_NCLOB function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TO_NCLOB (expression)

Parameters:

***expression*:** It is used to specify the expression to be converted.

Example:

```
select TO_NCLOB( lob_column)
from students;
```

Explanation:

Here, the value in the field called lob_column will be converted to a NCLOB value.

TO_LOB Function In Oracle:

The TO_LOB is one of the vital Conversion functions of Oracle. It is used to convert a LONG or LONG RAW values to LOB values. The TO_LOB function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

```
TO_LOB ( long_column )
```

Parameters:

***long_column*:** It is used to specify the LONG or LONG RAW value to be converted.

Example:

```
insert into class (lob_column)
select TO_LOB(long_column)
from students;
```

Explanation:

Here, the value in the field called long_column will be converted to a LOB value.

TO_CLOB Function In Oracle:

TO_CLOB is one of the vital Conversion functions of Oracle. It is used to convert a LOB value to the database character set from the national character set. The TO_CLOB function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

TO_CLOB (**expression**)

Parameters: expression: It is used to specify the expression to be converted.

Example:

```
select TO_CLOB (clob_column)
from students;
```

Explanation:

Here, the value in the field called clob_column will be converted to a CLOB value.

SYSTIMESTAMP Function In Oracle:

SYSTIMESTAMP is one of the vital Date/Time functions of Oracle. It is used to get the current system timestamp on the local database. The SYSTIMESTAMP function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

SYSTIMESTAMP

Example:

```
Select SYSTIMESTAMP
INTO var
from dual;
```

```
Select SYSTIMESTAMP
from dual;
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.003 seconds

	SYSTIMESTAMP
1	01-02-22 11:09:08.454000000 AM +05:30

Explanation:

The var is a variable that will contain the system's timestamp at the moment of execution.

SYS_CONTEXT Function In Oracle:

The SYS_CONTEXT is an advanced function that the Oracle database supports. It is used to get information about the Oracle environment. The SYS_CONTEXT function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

```
SYS_CONTEXT( namespace, parameter, length )
```

Parameters:

namespace: It is used to specify the already created namespace.

parameter: It is used to specify a valid attribute (set using the DBMS_SESSION.set_context procedure).

length: It is an optional parameter which is used to specify the length of the return value in bytes. It has a default value of 256 bytes.

Example:

```
SYS_CONTEXT('USERENV', 'NLS_DATE_FORMAT')
```

Output:

'RR-MM-DD'

SESSIONTIMEZONE Function In Oracle:

SESSIONTIMEZONE is one of the vital Date/Time functions of Oracle. It is used to get the current session's time zone. The result generated is either as a time zone offset or a time zone region name. The SESSIONTIMEZONE function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g and Oracle 9i.

Syntax:

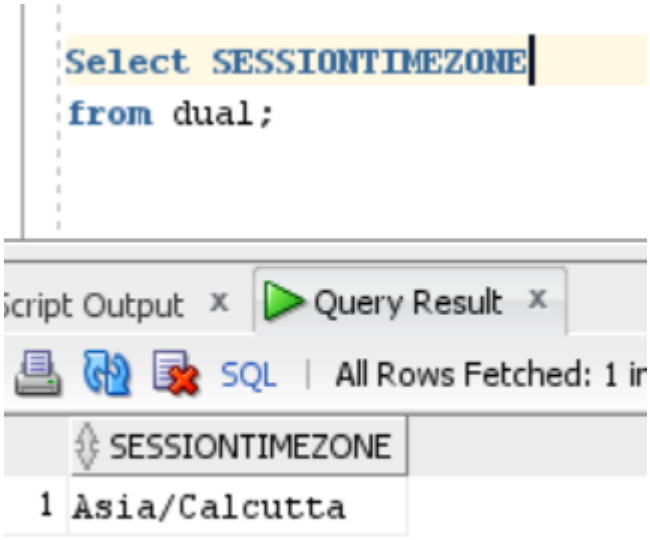
SESSIONTIMEZONE

Example:

```
Select SESSIONTIMEZONE
from dual;
```

Output:

-02:00



Explanation:

The result depends on the timezone value of the Oracle database.

LENGTH2 Function In Oracle:

LENGTH2 is one of the vital string/char functions of Oracle. It is used to get the length of a string, same as the LENGTH function with the only difference that it uses the UCS2 code points. The LENGTH2 function is supported in the various versions of the Oracle/PLSQL, including, Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i and Oracle 8i.

Syntax:

```
LENGTH2 ( string )
```

Parameters:

string: It is used to specify the string whose length needs to get.

Example 1:

```
LENGTH2 (NULL)
```

Output:

NULL

Explanation:

There is no string passed hence the result is NULL.

Example 2:

```
LENGTH2 ('')
```

Output:

NULL

Explanation:

There is no string passed hence the result is NULL.

Example 3:

```
LENGTH2 (' ')
```

Output:

1

Explanation:

The SPACE is passed as a string, hence the result is 1.

Example 3:

```
LENGTH2 ('HELLOWORLD.COM')
```

Output:

14

Explanation:

The total number of characters including the '.' is 14 and so is the result..

INSTRB function -

Syntax- INSTRB (string, substring, start_position, nth_appearance)

Start_position: It is an optional parameter which is used to specify the position in string where the search will start. Its default value is 1, i.e, the first position. It also accepts negative value, and in that case, it counts back from the end of string and then starts the search backwards towards the beginning of the string. However, whatever be the case the value of the position will be same as counted from the start, i.e, the first position belongs to the first character of the string only the start position is from the beginning or the end.

nth appearance : It is also an optional parameter which is used to specify the nth appearance of the substring. Its default value is 1.

Query and output as below-

```
select category_name, INSTRB('Electric Bikes','i',1,2) as match from bicycle
where category_name='Electric Bikes'
```

ript Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.002 seconds

	CATEGORY_NAME	MATCH
1	Electric Bikes	11
2	Electric Bikes	11
3	Electric Bikes	11

INSTRC function-

Syntax- INSTRC(string, substring, start_position, nth_appearance)

It works same as INSTRB function with the only difference that it uses UNICODE complete characters. There are lot of Unicode Character are present.

Query and output as below-

```
select category_name, INSTRC('Electric Bikes','i',1,2) as match from bicycle
where category_name='Electric Bikes'
```

cript Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.003 seconds

	CATEGORY_NAME	MATCH
1	Electric Bikes	11
2	Electric Bikes	11
3	Electric Bikes	11

Difference between INSTRB and INSTRC

INSTRB

Strings consist of bytes. The return value indicates the byte position at which the substring is found.

INSTRC

Strings consist of Unicode characters. Decomposed Unicode characters are recognized (e.g., a\0303 is recognized as being the same as \00E3 or ã). Eg., https://unicode-table.com/en/ ````` @ #

LAG function-

Syntax- SELECT id, marks,
LAG (marks,1) OVER (ORDER BY marks) AS prev_marks
FROM stu;

Query and output as below-

```
SELECT id, marks,
LAG (marks,1) OVER (ORDER BY marks) AS prev_marks
FROM stu;
```

ript Output x | Query Result x | Query Result 1 x | Query Resu

SQL | All Rows Fetched: 6 in 0.003 seconds

	ID	MARKS	PREV_MARKS
1	4	75	(null)
2	1	80	75
3	6	84	80
4	5	85	84
5	3	95	85
6	2	100	95

LAST_VALUE function-

Syntax-

SELECT DISTINCT LAST_VALUE (marks)
OVER (ORDER BY marks ASC
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
AS "HIGHEST"
FROM stu;

RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING : It changes the First row in the windows with a change in the current row.

Query and output as below-

```
 SELECT
  category_name,
  yr,
  quantity,
  LAST_VALUE(category_name) OVER(
            PARTITION BY yr  ORDER BY quantity
       RANGE BETWEEN
          UNBOUNDED PRECEDING AND
          UNBOUNDED FOLLOWING
    ) highest_sales_volume
FROM
   bicycle
WHERE
   yr IN (2016,2017,2018);
```

```
SELECT
category_name,
yr,
quantity,
LAST_VALUE(category_name) OVER(
        PARTITION BY yr
    ORDER BY quantity
    RANGE BETWEEN
        UNBOUNDED PRECEDING AND
        UNBOUNDED FOLLOWING
) highest_sales_volume
```

cript Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 18 in 0.003 seconds

	CATEGORY_NAME	YR	QUANTITY	HIGHEST_SALES_VOLUME
1	Electric Bikes	2016	93	Road Bikes
2	Comfort Bicycles	2016	94	Road Bikes
3	Cruisers Bicycles	2016	104	Road Bikes
4	Cyclocross Bicycles	2016	111	Road Bikes
5	Mountain Bikes	2016	116	Road Bikes
6	Road Bikes	2016	116	Road Bikes
7	Comfort Bicycles	2017	19	Mountain Bikes
8	Cruisers Bicycles	2017	20	Mountain Bikes
9	Cyclocross Bicycles	2017	29	Mountain Bikes

LEAD function-

Syntax- SELECT id, marks,
LEAD (marks,1) OVER (ORDER BY marks) AS next_marks
FROM stu;

Query and output as below-

```
SELECT extra_marks, marks,
LEAD (marks,1) OVER (PARTITION BY extra_marks ORDER BY marks) AS next_marks
FROM stu;
```

ipt Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 6 in 0.002 seconds

	EXTRA_MARKS	MARKS	NEXT_MARKS
1	0	84	(null)
2	10	75	80
3	10	80	(null)
4	15	85	95
5	15	95	(null)
6	20	100	(null)

LENGTH4 function-

Query and output as below-

```
select category_name, LENGTH4 (category_name) from bicycle
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 18 in 0.001 seconds

	CATEGORY_NAME	LENGTH4(CATEGORY_NAME)
1	Comfort Bicycles	16
2	Cruisers Bicycles	17
3	Cyclocross Bicycles	19
4	Electric Bikes	14
5	Mountain Bikes	14
6	Road Bikes	10
7	Comfort Bicycles	16
8	Cruisers Bicycles	17
9	Cyclocross Bicycles	19

LENGTHB function-

Query and output as below-

```
select category_name, LENGTHB (category_name) from bicycle
```

ript Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 18 in 0.002 seconds

	CATEGORY_NAME	LENGTHB(CATEGORY_NAME)
1	Comfort Bicycles	16
2	Cruisers Bicycles	17
3	Cyclocross Bicycles	19
4	Electric Bikes	14
5	Mountain Bikes	14
6	Road Bikes	10
7	Comfort Bicycles	16
8	Cruisers Bicycles	17
9	Cyclocross Bicycles	19

LENGTHC function-

Query and output as below-

```
select category_name, LENGTHC (category_name) from bicycle
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 18 in 0.002 seconds

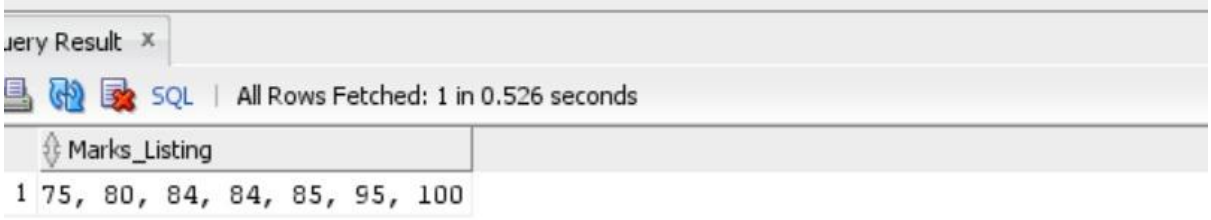
	CATEGORY_NAME	LENGTHC(CATEGORY_NAME)
1	Comfort Bicycles	16
2	Cruisers Bicycles	17
3	Cyclocross Bicycles	19
4	Electric Bikes	14
5	Mountain Bikes	14
6	Road Bikes	10
7	Comfort Bicycles	16
8	Cruisers Bicycles	17
9	Cyclocross Bicycles	19

AL32UTF8 is a multibyte character set: a character can take 1,2,3 or 4 bytes for storage. LENGTHB returns the length in number of bytes and LENGTHC returns the length in number of characters.

LISTAGG function-

Query and output as below-

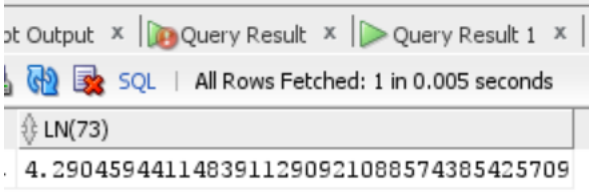
```
SELECT LISTAGG( marks, ', ') WITHIN GROUP (ORDER BY marks) "Marks_Listing"
FROM stu;
```



LN function-

Query and output as below-

```
SELECT LN(73)
FROM DUAL;
```



LNNVL function-

The LNNVL function will return to the following:

The condition is assessed as	LNNVL will return the value
TRUE	FALSE
FALSE	TRUE
UNKNOWN	TRUE

Query and output as below-

```
select * from stu
where lnnvl(prev_marks >= marks);
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 5 in 0.001 seconds

	ID	MARKS	EXTRA_MARKS	PREV_MARKS
1	2	100	20	97
2	3	95	15	80
3	4	75	10	70
4	5	85	15	75.55
5	6	84	0	60

LOCALTIMESTAMP function-

Query and output as below-

```
Select LOCALTIMESTAMP
from dual;
```

ript Output | Query Result | Quer

SQL | All Rows Fetched: 1 in 0.0

	LOCALTIMESTAMP
1	01-02-22 3:37:49.826000000 PM

MAX function-

Query and output as below-

```
SELECT MAX(marks)
FROM stu;
```

ript Output | Query Re

SQL | All Rows

	MAX(MARKS)
1	100

MEDIAN function-

Query and output as below-

```
select median(marks)
from stu;
```

Script Output | Query Result

SQL | All Rows Fet

	MEDIAN(MARKS)
1	84.5

MIN function-

Query and output as below-

```
SELECT MIN(marks)
FROM stu;
```

ot Output | Query Re

SQL | All Row

MIN(MARKS)
75

MOD function-

Query and output as below-

```
SELECT Mod(marks,prev_marks)
FROM stu where id =1
```

pt Output | Query Result | Query R

SQL | All Rows Fetched: 1 in 0.001

	MOD(MARKS,PREV_MARKS)
1	80

MONTHS_BETWEEN function-

Query and output as below-

```
SELECT MONTHS_BETWEEN(DATE '2031-06-10', DATE '2030-06-10')
FROM DUAL;
```

Script Output | Query Result | Query Result 1 | Query Result 2 | Qu

SQL | All Rows Fetched: 1 in 0.001 seconds

	MONTHS_BETWEEN(DATE'2031-06-10',DATE'2030-06-10')
1	12

NANVL function-

Query and output as below-

```
select NANVL(prev_marks, 45)
from stu;
```

ript Output | Query Result | Que

SQL | All Rows Fetched: 6 in 0

	NANVL(PREV_MARKS,45)
1	90
2	97
3	80
4	70
5	75.55
6	60

NEW_TIME function-

Query and output as below-

```
select NEW_TIME (TO_DATE ('2022/02/01 12:30', 'yyyy/mm/dd HH24:MI'), 'GMT', 'PST')
from dual;
```

ript Output | Query Result | Query Result 1 | Query Result 2 | Query Result 3 | Query Result 4

SQL | All Rows Fetched: 1 in 0 seconds

	NEW_TIME(TO_DATE('2022/02/0112:30','YYYY/MM/DDHH24:MI'),'GMT','PST')
1	01-FEB-2022 04:30:00

NEXT_DAY function-

Query and output as below-

```
select NEXT_DAY ( '26-JUN-19', 'SUNDAY')
from dual;
```

ript Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.001 seconds

	NEXT_DAY('26-JUN-19','SUNDAY')
1	02-JUL-0019 00:00:00

NULLIF function in Oracle

NULLIF function is used to compare two values and return null if the values are same, else return the first value.

Syntax: nullif(expr1,expr2)

Parameters:

expr1, expr2: They are used to specify the expressions to compare.

Worksheet | Query Builder

```
select nullif(1,1) from dual;
```

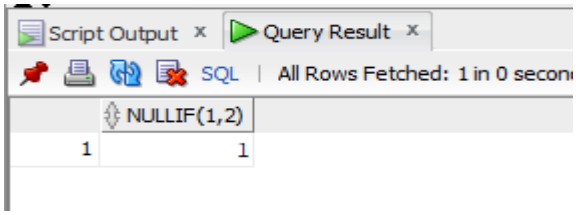
Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.001

	NULLIF(1,1)
1	(null)

Worksheet | Query Builder

```
select nullif(1,2) from dual;
```

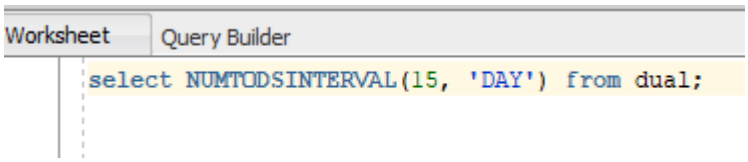


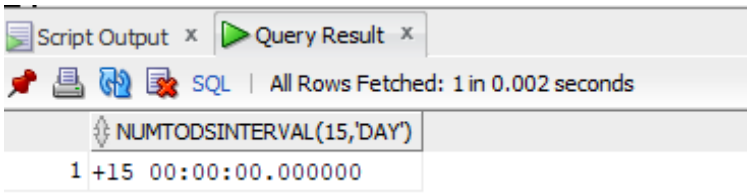
NUMTODSINTERVAL function in Oracle

NUMTODSINTERVAL function converts a number to an INTERVAL DAY TO SECOND literal**.**

Syntax: NUMTODSINTERVAL (number, expression)

Parameters:
number: It is used to specify the number to be converted.
expression: It is used to specify the unit to be converted to.



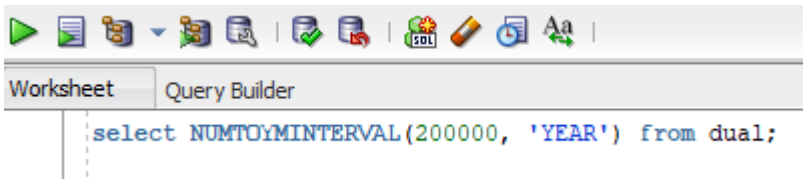


NUMTOYMINTERVAL function in Oracle

NUMTOYMINTERVAL is one of the vital Conversion functions of Oracle. It is used to convert a number to an INTERVAL YEAR TO MONTH literal.

Syntax: NUMTOYMINTERVAL (number, expression)

Parameters:
number: It is used to specify the number to be converted.
expression: It is used to specify the unit to be converted to.



Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.003 seconds

	NUMTOYMINTERVAL(200000,'YEAR')
1	+200000-00

NVL function in Oracle

NVL function is used to substitute a value, when a NULL value is encountered.

Syntax: NVL (value, replace_with)
Parameters:
value: It is used to specify the value to be tested for a NULL value.
replace_with: It is used to specify the value to return if value is NULL.

**Null Values in the COMM Column*

Script Output | Query Result

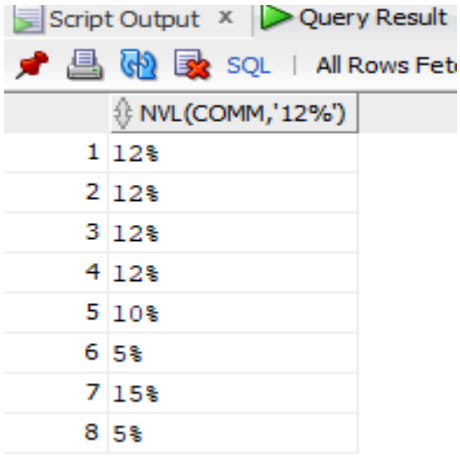
SQL | All Rows Fetched: 8 in 0.01 seconds

	EMP_ID	EMP_NAME	CITY	DEPT_NO	EMP_SAL	COMM
1	1	emp1	Ranchi	1	20000	(null)
2	2	emp2	Delhi	10	30000	(null)
3	3	emp3	Pune	10	25000	(null)
4	4	emp4	Bangalore	10	22000	(null)
5	5	emp5	Ranchi	1	26000	10%
6	6	Vignesh	Hyd	9	29000	5%
7	7	Yesser	Hyd	9	27000	15%
8	8	Aravinth	Goa	9	29000	5%

Worksheet | Query Builder

```
SELECT NVL (COMM, '12%')
FROM emp_records;
```

**Null Fields are replaced by values*

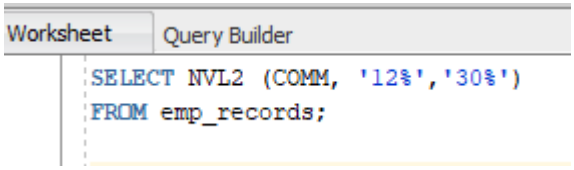


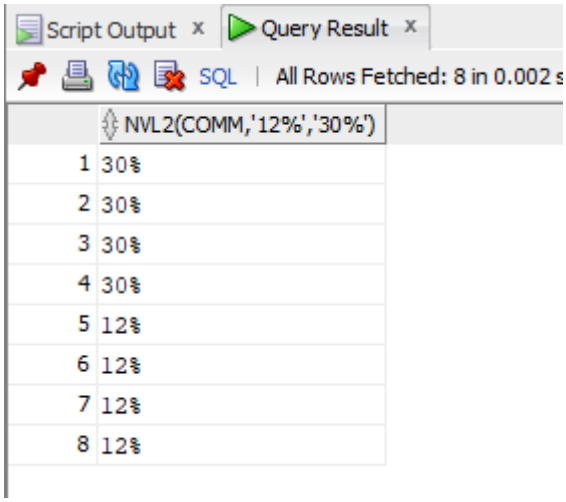
	NVL(COMM,'12%')
1	12%
2	12%
3	12%
4	12%
5	10%
6	5%
7	15%
8	5%

NVL2 function in Oracle

NVL2 function is used to substitute a value, if a NULL value is encountered and to substitute another value, if a non-NULL value is encountered.

Syntax: NVL2 (value, replace_with_if_not_null, replace_with_if_null)
Parameters:
value: It is used to specify the value to be tested for a NULL value.
replace_with_if_not_null: It is used to specify the value to return if value is not NULL.
replace_with_if_null: It is used to specify the value to return if value is NULL.



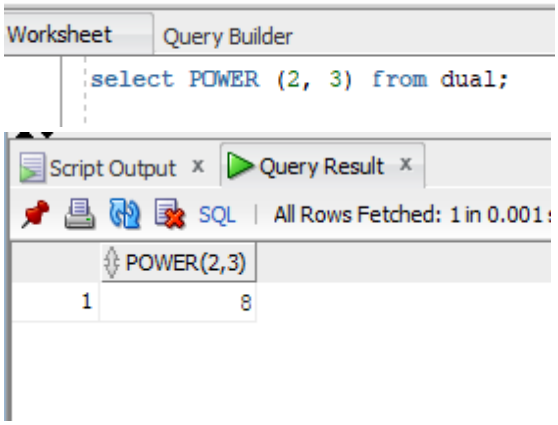


	NVL2(COMM,'12%','30%')
1	30%
2	30%
3	30%
4	30%
5	12%
6	12%
7	12%
8	12%

POWER function in Oracle

POWER is one of the vital Numeric/Math functions of Oracle. It is used to get the value of x raised to yth power.

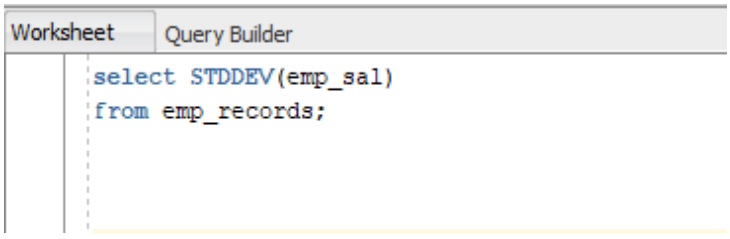
Syntax: POWER (x, y)
Parameters:
x, y: They are used to specify the numbers to be used for calculation.

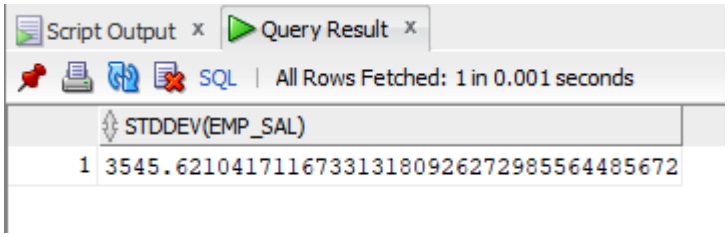


STDDEV function in Oracle

STDDEV functions is used to get the standard deviation of a set of numbers.

Syntax: stddev([DISTINCT | ALL] expression)
Parameters:
expression: It is used to specify a numeric value or a formula.





31. NTH_VALUE function in Oracle

NTH_VALUE function is used to get a specific value in an ordered set of values from an analytic window.

Syntax: NTH_VALUE (measure_column, n)
[FROM FIRST | FROM LAST]
[RESPECT NULLS | IGNORE NULLS]
OVER ([query_partition_clause] [order_by_clause [windowing_clause]])

Parameters:
measure_column: It is used to specify the expressions or columns to be returned.
n: It is used to specify the nth value of measure_column in the analytic window which needs to be retrieved.
FROM FIRST | FROM LAST: It is an optional parameter which is used to specify whether to start the calculation at the first row or the last row of the analytic window. The default value is FROM FIRST.
RESPECT NULLS | IGNORE NULLS: It is an optional parameter which is used to specify whether to include or ignore the NULL values in the calculation. The default value is RESPECT NULLS.
query_partition_clause: It is also an optional parameter which is used to partition the results into groups.
order_by_clause: It is also an optional parameter which is used to order the data within each partition.
windowing_clause: It is also an optional parameter which is used to specify the rows in the analytic window to be evaluated.

Values of windowing_clause:
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW: It is the default value that changes the Last row in the window with a change in the current row.
RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING: It changes the First row in the windows with a change in the current row.
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING: It includes all the rows in the window, regardless of the current row.

```
Worksheet  Query Builder
SELECT DISTINCT NTH_VALUE (emp_sal,1)
 OVER (ORDER BY emp_sal DESC
       RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
       AS HIGHEST FROM emp_records;
```

Query Result

SQL | All Rows Fetched: 1 in 0.001 seconds

	HIGHEST
1	30000

SINH function in Oracle

SINH is used to get the hyperbolic sine of a number.

Syntax: SINH (number)

Parameters:
number: It is used to specify the number to get the hyperbolic sine value of.

```
Worksheet  Query Builder
select SINH (1) from dual;
```

Query Result

SQL | All Rows Fetched: 1 in 0.008 seconds

	SINH(1)
1	1.175201193643801456882381850595600815

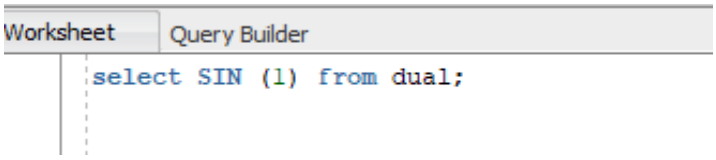
SIN function in Oracle

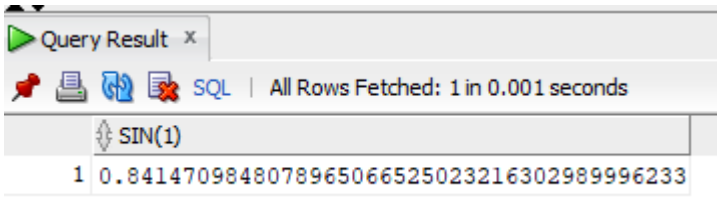
SIN function is used to get the Sine of a number.

Syntax: SIN (number)

Parameters:

number: It is used to specify the number in radians to get the Sine value of.





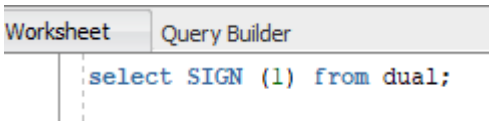
SIGN function in Oracle

SIGN function is used to get a value indicating the sign of a number.

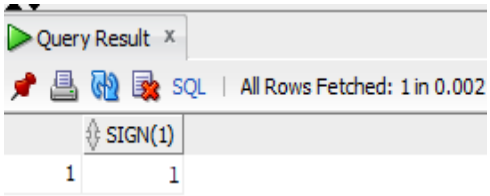
Syntax: SIGN (number)

Parameters:

number: It is used to specify the numbers to be tested for its sign.

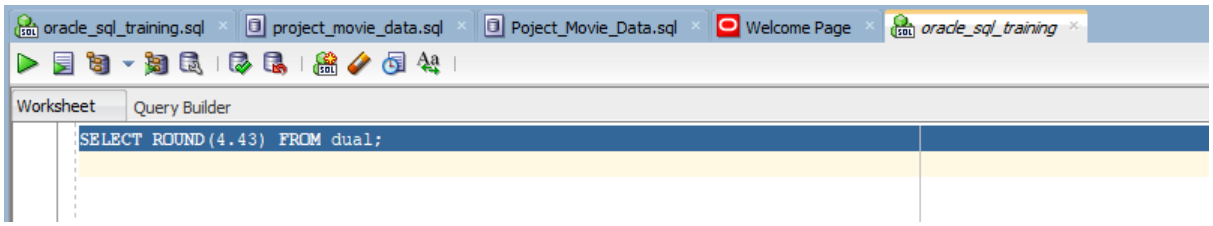


**This returns a positive or a negative value*

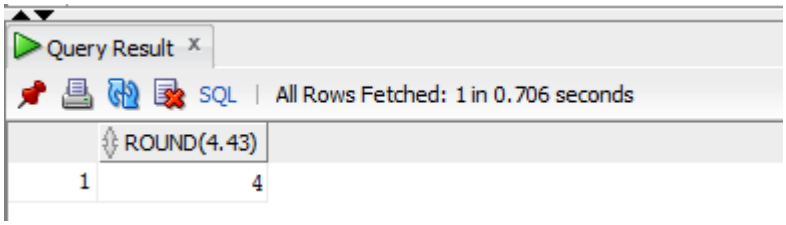


ROUND (numbers) function in Oracle:

This function is used to return n rounded to integer places to the right of the decimal point. If no integer is defined, then n is rounded to zero places. If the integer specified is negative, then n is rounded off to the left of the decimal point.

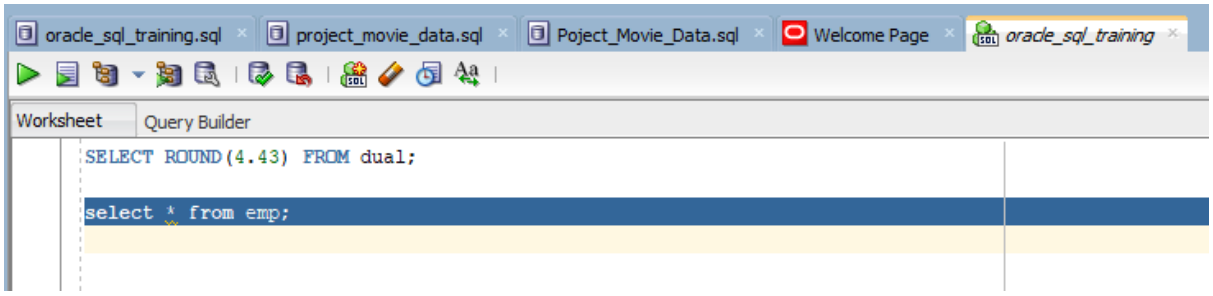


OUTPUT:



FROM clause in Oracle:

The FROM clause is used to retrieve rows from the referenced table(s). As shown in the syntax, the FROM clause can either reference a single table or include a nested tables clause. For nested tables, see the Using NESTED TABLES clause to query multiple tables in the same hierarchy section.



OUTPUT:

Query Result

SQL | All Rows Fetched: 7 in 0.016 seconds

	EMP_ID	EMP_NAME	MANAGER_ID	DEPT
1	1	Minato	A6	Astrology
2	2	Itachi	C6	Chemistry
3	3	Sasuke	M6	Mathematics
4	4	Naruto	H6	History
5	5	Shikamaru	G6	Geography
6	6	Madra	P6	Physics
7	7	Obito	A6	Astrology

SUM function in Oracle

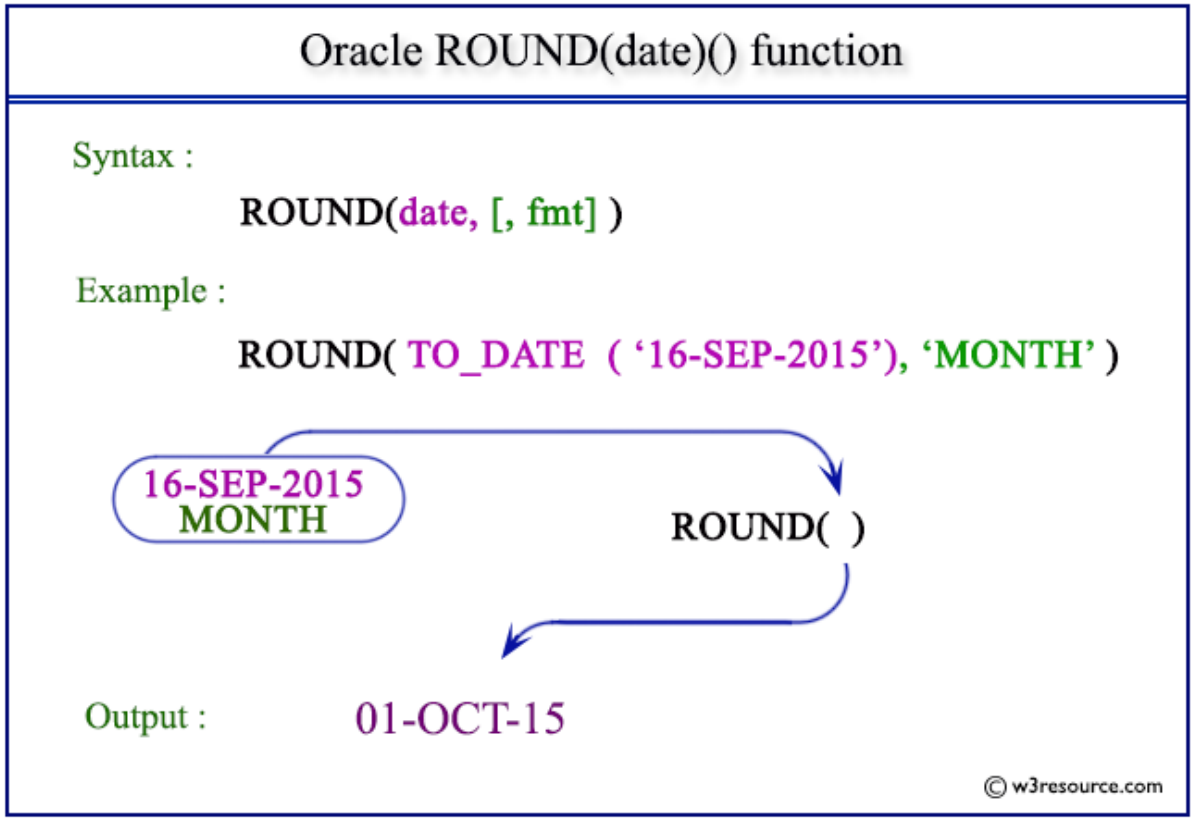
Syntax: SELECT SUM(aggregate_expression) FROM tables [WHERE conditions];

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

OUTPUT:

```
12743
```

ROUND (dates) function in Oracle




```
SELECT ROUND(TO_DATE ('16-SEP-2015'),'MONTH')    "New Month",
ROUND(TO_DATE ('16-SEP-2015'),'YEAR')    "New Year"
FROM DUAL;
```

OUTPUT:

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.006 seconds

	New Month	New Year
1	01-10-15	01-01-16

REMAINDER function in Oracle:

The MOD function is similar to REMAINDER except that the MOD uses FLOOR in its formula, whereas REMAINDER uses ROUND.

```
   DEC_NUM BIN_DOUBLE  BIN_FLOAT
---------- ---------- ----------
   1513.67 1.514E+003 1.514E+003
```

```
SELECT bin_float, bin_double, REMAINDER(bin_float, bin_double)
FROM float_point_test;
```

OUTPUT:

```
 BIN_FLOAT BIN_DOUBLE REMAINDER(BIN_FLOAT,BIN_DOUBLE)
---------- ---------- --------------------------------
1.514E+003 1.514E+003                         1.0E-001
```

RANK function in Oracle:

RANK is one of the vital Analytic functions of Oracle. It is used to get the rank of a value in a group of values. It can also result in the non-consecutive ranking of the rows.

```
SELECT
    col,
    RANK() OVER (ORDER BY col) my_rank
FROM
    rank_demo;
```

OUTPUT:

	COL	MY_RANK
1	A	1
2	A	1
3	B	3
4	C	4
5	C	4
6	C	4
7	D	7

RAWTOHEX function in Oracle:

RAWTOHEX is one of the vital Conversion functions of Oracle. It is used to convert a raw value to a hexadecimal value.

```
RAWTOHEX('AB')
```

OUTPUT:

The result will be ‘AB’ if run as a PLSQL function or the result will be ‘4142’ if run as a SQL function.

```
'4142'
OR
 'AB'
```

REGEXP_COUNT function in Oracle:

The Oracle REGEXP_COUNT function is used to count the number of times that a pattern occurs in a string. It returns an integer indicating the number of occurrences of a pattern. If no match is found, then the function returns 0.

```
1  SELECT REGEXP_COUNT ('The web development Tutorial', 't') FROM dual;
```

OUTPUT:

```
REGEXP_COUNT('THEWEBDEVELOPMENTTUTORIAL','T')
---------------------------------------------
                                            2
```