DATE & TIME FUNCTION

NOTE:
SELECT CEIL(48.99) = 49
FLOOR (49.11) = 49

1) SELECT CURRENT_DATE FROM dual;
2) SELECT SYSDATE FROM dual;
3) ADD_MONTHS (date, number_months)
→ Eg: SELECT ADD_MONTHS ('15-FEB-2018', 2) AS MONTHS_ADDED FROM dual;

4) MONTHS_BETWEEN (date1, date2)
→ Eg: SELECT MONTHS_BETWEEN ('02-Aug-2003', '02-Jun-2003') AS RESULT FROM dual;
output: 2

NOTE: output can be in negative, decimal

Eg: SELECT MONTHS_BETWEEN (TO_DATE ('2003/08/02', 'yyyy/mm/dd'), TO_DATE ('2003/06/02', 'yyyy/mm/dd')) AS RESULT FROM DUAL;

5) SELECT EXTRACT (DAY FROM DATE '2017-07-14') AS DAY FROM dual;
→ output:

DAY
14

6) LAST_DAY (date);
NEXT_DAY (date, weekday); ⇒ Eg: SELECT NEXT_DAY ('15-FEB-2018', 'WEDNESDAY') AS RESULT FROM dual;
↓

Result
21-FEB-18

SUBQUERY
→ Rules: It is placed on right side of comparison operator.
ORDER BY CLAUSE cannot be added

Occur in: SELECT, FROM, WHERE, HAVING CLAUSE.
nested inside: SELECT, INSERT, UPDATE, DELETE STATEMENT

Eg:

Product	
Prod_Id	Pdt_Name
300	Toys
301	Rhymes
302	Shirt

Product_Sold	
Prod_Id	Sold out
300	10
301	4
302	5

Query to display productId, Pdt_Name and sold_out of those products which are sold better than the product with Id 302.

Sol^n: 1) Query to return the sold_out of product id 302 from Product_sold table

2) Query to Identity the products which are sold better than result of first query

Select sold_out from Product_sold where Product_Id = 302
↓
output: 5

Select Product_Id, Pdt_Name, sold_out from Product, Product_sold where sold_out > 5
↓
Output:

Prod Id	Pdt_Name	Soldout
300	Toy	10
301	Rhymes	10
302	Shirt	10

✗ WRONG RESULT
(∵ there is no path checking)

★ To clearify which productId you were told. You can Alias, Eg: p.productId

So, SELECT Product.Product_Id, Pdt_Name, Sold_out
FROM Product, Product_Sold
WHERE Product.Product_Id = Product_Sold.Product_Id
and Sold_Out > 5
↓
Output:

Prod Id	Pdt_Name	Sold_out
300	Toys	10

① + ② Select p.product_Id, Pdt_Name, Sold_out from Product p, Product_Sold s where p.product_Id = s.product_Id and sold_out > [(Select Sold_out from Product_sold where Product_Id = 302)]
↓ output = 5

TYPES OF SUBQUERIES

1) Single row subqueries (use: =)   >ANY / <ANY

2) Multiple row subqueries (use: IN , ANY or ALL)

3) Multiple column subqueries

4) Correlated subqueries (USE: =) [The sub query depends on the outer query for its values]

↳ Eg: SELECT * FROM EMP
WHERE EXISTS (SELECT *
FROM DEPT where dept.Eid = Emp.Eid) ;

#STRING FUNCTION

1) CONCAT (string1, string2) - Eg: SELECT CONCAT('Hello', 'world') = HelloWorld

2) SELECT 'Hello' || 'world' → HelloWorld   same function

3) SELECT INITCAP ('work work') → Work Work

4) SELECT LENGTH('Hello World') → 11

5) LTRIM   RTRIM

left से remove होगा   right se

6) LPAD('Hello', 8, 'H')   RPAD('Hello', 8, '0')

HHHHello

7) LOWER ('ansh') → ansh

UPPER → ANSH

8) INSTR ('Welcome to', 'e') → 2

(" " , 'e', 1, 2) → 7

starting   appearance

9) REPLACE ('HelloWorld', 'Hello') → World

10) REPLACE (" " , 'Hi') → HiWorld

11) SUBSTR('Hello World', 7) → World

" ('Hello World', 1, 4) → Hello

VIEWS → provide security
simplify complex queries
limit data access

Syntax: CREATE VIEW [view_name]
AS
[SELECT statement]

NORMAL Eg: CREATE VIEW CustomersInChennai
AS
SELECT Customer_Id, FirstName, Phone
FROM customer WHERE City =
'Chennai';

GROUP + ORDER BY Eg:

```
CREATE VIEW salesperorder
AS
SELECT Order_Id, SUM(Quantity) total_quantity
FROM PRODUCT_ORDERS
GROUP BY ORDER_ID
ORDER BY totalquantity DESC;
```

JOINS have changes

UPDATE VIEW — Syntax: CREATE OR REPLACE [view_name]
AS
[SELECT statement]

DROP VIEW — DROP VIEW view_name

SET OPERATORS

UNION	UNION ALL	INTERSECT	MINUS
SELECT Col_name FROM table1	〃 〃 〃	〃 〃 〃	〃 〃 〃
UNION	UNION ALL	INTERSECT	MINUS
〃 〃 〃 〃 2	〃 〃 〃	〃 〃 〃	〃 〃 〃