# Institute of Computer Technology

# B. Tech Computer Science and Engineering

Sub: Database Management System (2CSE301)

Practical 3: **Performing queries with various operators and functions.**

Scenario : Mohan was worried about total income to getting raised every month for which he has to query differently in sales table. Thus he suggested IT Company to have multiple options to see the count of sales happen every day or weekly or monthly.

## COMPUTATION ON TABLE DATA

None of the techniques used till now allows displays of some data from a table after some arithmetic has been done with it.

Arithmetic and logical operators give a new dimension to SQL sentences.

### Arithmetic Operators:

Oracle allows arithmetic operator to be used while viewing records from a table or while performing Data Manipulation operations such as Insert, Update and Delete.

| | | | |
|---|---|---|---|
| + | Addition | * | Multiplication |
| - | Subtraction | ** | Exponentiation |
| / | Division | ( ) | Enclosed operation |

For example:
Retrieve the content of the column p_no, description and compute 5% of the values contained in the column sell_price for each row from the table product_master.

```
Sql>SELECT p_no, description, sell_price*0.05
        FROM product_master;
```

### Renaming Columns used with Expression Lists:

When displaying the result of a query, SQL *PLUS normally uses the selected column's name as the column heading.

These column names may however be short and cryptic; they can be changed for better understanding of the query result by entering an alias, or substitute name, after the column name in the select clause.

```
Sql>SELECT columnname result_columnname, columnname result_columnname
            FROM table name For
example:
```

Retrieve the content of the column p_no, description and compute 5% of the values contained in the column sell_price for each row from the table product_master. Rename sell_price * 0.05 as **Increase.**

```
Sql>SELECT p_no, description, sell_price*0.05  Increase
      FROM product_master;
```

## Logical Operators:

Logical operators that can be used in SQL sentence are:

**1. AND operator:**

The Oracle engine will process all rows in a table and display the result only when all of the conditions specified using the AND operator are satisfied.

```
sql >SELECT column list
      FROM tablename
          WHERE columnname AND columnname;
```

```
Sql>SELECT p_no, desc, p_percent
       FROM product_master
         WHERE p_percent>=10 AND p_percent<=20;
```

**2. OR operator:**

The Oracle engine will process all rows in a table and display the result only when any of the conditions specified using the OR operators are satisfied.

```
sql >SELECT column list
        FROM tablename
           WHERE columnname OR columnname;
```

```
sql >SELECT c_no, name, address, pincode
      FROM client_master
        WHERE (pincode=400125 OR pincode=400126);
```

### 3. NOT operator:

The Oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.

```
Sql> SELECT c_no, name, address, pincode
        FROM client_master WHERE NOT (city='Bombay' or city='Delhi');
```

**Range Searching:**

In order to select data that is within a range of values, the **BETWEEN** operator is used. This operator allows the selection of rows that contain values within a specified lower and upper limit.

```
sql >SELECT column list from tablename
       WHERE column BETWEEN min _value AND max_value;

sql >SELECT c_no, name, address, pincode
        FROM client_master
          WHERE  bal_due  BETWEEN 100  AND 500;
```

**Note:**

.BETWEEN is an inclusive operator i.e. if either the min value or the max value is found, as well as any in between, the row is returned.

### 4. NOT BETWEEN

Rows not having value in the range specified, and also not having value equal; to min or the max value is returned.

```
sql >SELECT column list from tablename
         WHERE  column  NOT  BETWEEN  min  _value  AND  max_value;
sql >SELECT c_no, name, address, pincode
        FROM client_master
          WHERE  bal_due NOT  BETWEEN 100  AND 500;
```

## Pattern Matching:

### 1. LIKE

Allows comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are available are:

- The percent sign **(%)** that matches any string.
- The Underscore **( _ )** that matches any single character.

```
sql >SELECT column list FROM tablename
```

```
        WHERE column LIKE 'pattern';
```

OR

```
         WHERE column NOT LIKE' pattern'; For
```

example:

Retrieve all information about suppliers whose name begin with the letter ‘ja’ fro supplier_master.

```
sql >SELECT * FROM supplier_master
          WHERE s_name LIKE ‘ja%’;
```

**2. <u>IN</u>**

This operator can be used to select rows that match one of the values included in the list.

```
sql>SELECT columnlist FROM tablename
              WHERE columnlist IN (list of values);
```

For example:
Retrieve the details from supplier table where supplier name is either Aman or Vimal or Ajay.

```
sql>SELECT s_no, name, city, address, pincode
        FROM supplier_master
          WHERE name IN (‘Aman’, ‘Vimal’, ‘ Ajay’ );
```

**3. <u>NOT IN</u>**

The **NOT IN** predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

```
sql>SELECT columnlist FROM tablename
              WHERE columnlist NOT IN (list of values);
```

**4. <u>IS NULL</u>**

This operator is used to compare the value in the column with NULL and return the row accordingly.

```
sql >SELECT column list FROM tablename
               WHERE column is NULL;
```

OR

```
               WHERE column is not NULL;
```

**ORACLE FUNCTIONS:**

Oracle functions serve the purpose of manipulating data items and returning result. Functions are also capable of accepting user-supplied variables or constants and operating on them. Such variables or constants are called as argument. Any number of arguments can be passed to a function in the following format:

**Function_name (argument1, argument2, …).**

Oracle functions can be clubbed together depending upon whether they operate on a single row or a group of rows retrived from a table. Accordingly, functions can be classified as follows:

**Group Functions (Aggregate Function):**
Functions that act on a set of values are called as group functions. For example, SUM, is a function which calculates the total of a set of numbers. A group function returns a single result row a group of queried rows.

**Scalar Function (Single Row Function):**
Functions that act on only one value at a time are called as scalar functions. For example, LENGTH, is a function, which calculates the length of one particular string value. A single row function returns one result for every row of a queried table or view.

Single row function can be further grouped together by the data type of their arguments and return values. For example, LENGTH, relates to the string Data type. Functions can be classified corresponding to different data types as:

| | |
|---|---|
| String functions | : Work for String Data type |
| Numeric functions | : Work for Number Data type |
| Conversion functions | : Work for Conversion of one type to another. |
| Date functions | : Work for Date Data type |

## Aggregate Functions:

| | | |
|---|---|---|
| **AVG** | Syntax | AVG([DISTINCT\|ALL]n) |
| | Purpose | Return average value of n ignoring null values. |
| | Example | Select AVG(sell_price) "Average" from p_master; |
| | Output | Average<br>2012.3654 |
| **MIN** | Syntax | MIN([DISTINCT\|ALL]expr) |
| | Purpose | Return minimum value of 'expr'. |
| | Example | Select MIN(bal_due) "Min_bal" from c_master; |
| | Output | Min_bal<br>0 |
| **COUNT** | Syntax | MIN([DISTINCT\|ALL]expr) |
| | Purpose | Return the number of rows WHERE 'expr' is not null . |

| | | |
|---|---|---|
| | Example | Select COUNT(p_no) “Products” from P_master; |
| | Output | <u>Products</u><br>9 |
| **COUNT(*)** | Syntax | COUNT(*) |
| | Purpose | Return the number of rows in the table, including duplicates and those with nulls.. |
| | Example | Select COUNT(*) “Total” from C_master; |
| | Output | <u>Total</u><br>9 |
| **MAX** | Syntax | MAX([DISTINCT\|ALL]expr) |
| | Purpose | Return maximum value of ‘expr’. |
| | Example | Select MAX(bal_due) “Maximum” from c_master; |
| | Output | <u>Maximum</u><br>15000 |
| **SUM** | Syntax | SUM([DISTINCT\|ALL]n) |
| | Purpose | Return Sum of values of ‘n’. |
| | Example | Select SUM(bal_due) “Balance” from c_master; |
| | Output | <u>Balance</u><br>22000 |

## Numeric Functions:

| | | |
|---|---|---|
| **ABS** | Syntax | ABS(n) |
| | Purpose | Return the absolute values of ‘n’. |
| | Example | Select ABS(-15) “Absolute” from dual; |
| | Output | <u>Absolute</u><br>15 |
| **POWER** | Syntax | POWER(m,n) |
| | Purpose | Returns m raised to nth power. N must be an integer, else an error is returned. |
| | Example | Select POWER(3,2) “Raised” from dual; |
| | Output | <u>Raised</u><br>9 |
| **ROUND** | Syntax | ABS(n[,M]) |

| | | |
|---|---|---|
| | Purpose | Returns 'n' rounded to 'm' places right the decimal point. If 'm' is omitted 'n' is rounded to 0 places. 'm' can be negative to round off digit left of the decimal point 'm' must be an integer. |
| | Example | Select ROUND(15.19,1) "Round" from dual; |
| | Output | <u>Round</u><br>15.2 |
| **SQRT** | Syntax | SQRT(n) |
| | Purpose | Returns square root of 'n'. if n<0, NULL. SQRT returns a real result. |
| | Example | Select SQRT(25) "Square root" from dual; |
| | Output | <u>Square root</u><br>5 |

## String Functions:

| | | |
|---|---|---|
| **LOWER** | Syntax | LOWER(char) |
| | Purpose | Return char, with all letters in lowercase. |
| | Example | Select LOWER('XYZ') "Lower" from dual; |
| | Output | <u>Lower</u><br>xyz |
| **INITCAP** | Syntax | INITCAP(char) |
| | Purpose | Return STRING with first letter in upper case. |
| | Example | Select INITCAP('COMP DEPT') "Title Case" from dual; |
| | Output | <u>Title Case</u><br>Comp Dept |
| **UPPER** | Syntax | UPPER(char) |
| | Purpose | Return char, with all letters in uppercase. |
| | Example | Select UPPER('xyz') "Upper" from dual; |
| | Output | <u>Upper</u><br>XYZ |
| **SUBSTR** | Syntax | UPPER(char, M[,n]) |
| | Purpose | Return a portion of char, beginning at character 'm' exceeding up to 'n' characters. If 'n' is omitted, result is returned up to the end char. The first position of char is 1. |
| | Example | Select SUBSTR('SECURE',3,4) "Substring" from dual; |
| | Output | <u>Substring</u><br>CURE |

| | | |
|---|---|---|
| **LENGTH** | Syntax | LENGTH(char) |
| | Purpose | Return the length of character. |
| | Example | Select LENGTH(‘xyz’) “Length” from dual; |
| | Output | Length<br>3 |
| **LTRIM** | Syntax | LTRIM(char[,Set]) |
| | Purpose | Return characters from the left of char with initial. |
| | Example | Select LTRIM(‘College’,’C’) “Left” from dual; |
| | Output | Left<br>ollege |
| | | |
| **RTRIM** | Syntax | RTRIM(char[,Set]) |
| | Purpose | Return char, with final characters removed after the last character not I the set. ‘set’ is optional, it defaults to spaces. |
| | Example | Select RTRIM(‘College’,’e’) “Right” from dual; |
| | Output | Right<br>Colleg |
| **LPAD** | Syntax | LPAD(char1,n,[,char2]) |
| | Purpose | Return ‘char1’, left padded to length ‘n’ with the sequence of characters in ‘char2’, ‘char2, defaults to blanks. |
| | Example | Select LPAD(‘Page 1’,10,’*’) “Lpad” from dual; |
| | Output | Lpad<br>****Page 1 |
| **RPAD** | Syntax | RPAD(char1,n,[,char2]) |
| | Purpose | Return ‘char1’, right- padded to length ‘n’ with the characters in ‘char2’, replicated as many times as necessary. If ‘char2’ is omitted, right-pad is with blanks. |
| | Example | Select RPAD(‘page’,10,’x’) “Rpad” from dual; |
| | Output | Rpad<br>Pagexxxxxx |

## Conversion Functions:

| | | |
|---|---|---|
| **TO_NUMBER** | Syntax | TO_NUMBER(char) |
| | Purpose | Converts ‘char’ , a character value containing a number to a value of number datatype. |

|  |  |  |
|---|---|---|
|  | Example | Update P_master set sell_price= sell_price + TO_NUMBER(SUBSTR('$100',2,3));<br><br>Here the value 100 will be added to every products selling price in the product_master table. |
| **TO_CHAR** | Syntax | TO_CHAR(n[,fmt]) |
|  | Purpose | Converts a value of number data type to a value of char data type, using the optional format string. It accepts a number (n) and a numeric format (fmt) in which the number has to appear. If 'fmt' is omitted, 'n' is converted to a char value exactly long enough to hold significant digits. |
|  | Example | Select TO_CHAR(17145,'$099,999')"char" from dual; |
|  | Output | Char<br>$017,145 |
| **TO_CHAR** | Syntax | TO_CHAR(date[,fmt]) |
|  | Purpose | Converts a value of DATE data type to a value of char data type, using the optional format string. It accepts a date (date), as well as format (fmt) in which the date has to appear. 'fmt' must be a date format.. If 'fmt' is omitted, 'date' is converted to a char value in the default date format, i.e. "DD-MON-YY". |
|  | Example | Select TO_CHAR(O_DATE,'Month DD, YYYY) "Format" from s_order where o_no='o42541'; |
|  | Output | Format<br>January 26, 20006 |

## Date Conversion Functions:

|  |  |  |
|---|---|---|
| **TO_DATE** | Syntax | TO_DATE(char [,fmt]) |
|  | Purpose | Converts a character field to a date field. |
|  |  |  |
| **ADD_MONTHS** | Syntax | ADD_MONTHS(D,N) |
|  | Purpose | Returns date after adding the number of months specified with the function |
|  | Example | Select ADD_MONTHS(SYSDATE, 4) from dual; |
|  | Output | ADD_MONTHS<br>04-AUG-06 |
|  |  |  |
| **LAST_DAY** | Syntax | LAST_DAY(d) |
|  | Purpose | Returns the last date of the month specified with the function. |

| | | |
|---|---|---|
| | Example | Select SYSDATE,LAST_DAY(SYSDATE)"Last" from dual; |
| | Output | SYSDATE Last<br>04-AUG-06 31-AUG-06 |
| | | |
| **MONTHS_BETWEEN** | Syntax | MONTHS_BETWEEN(d1,d2) |
| | Purpose | Returns number of months between d1 and d2. |
| | Example | Select MONTHS_BETWEEN('04-AUG-06', '04-JUL-06' )"Month" from dual; |
| | Output | Month<br>1 |
| | | |
| **NEXT_DAY** | Syntax | NEXT_DAY(date,char) |
| | Purpose | Returns the date of the first weekday named by 'char' that is after the date named by 'date'. 'Char' must be a day of the week. |
| | Example | select NEXT_DAY('04-feb-06', 'Friday') "Next day" from dual; |
| | Output | Next day 06-feb-06 |

## The Oracle Table 'DUAL':

Dual is a small Oracle worktable, which consists of only one row and and one column, and contains the value x in that column. Besides arithmetic calculation, it also supports date retrieval and it's formatting.

When an arithmetic exercise is to be performed such as 2*2 or 4/3 etc., there really is no table being referenced; only numeric literals are being used.

To facilitate such calculation via a SELECT, Oracle provides a dummy table called DUAL, against which SELECT statements that are required to manipulate numeric literals can be fired, and output obtained.

Sql>**SELECT** 2*2 **FROM** DUAL;

Output:

```
       2*2
    ----------
        4
```

**SYSDATE:**

Sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table DUAL and returns the current date.

```
Sql>SELECT sysdate FROM DUAL;
```

Output:

```
        Sysdate
    _____________
     06-jun-06
```

## Exercise:

Create table sales_order and insert data as given below.

| **Column Name** | **Data Type** | **Size** |
|---|---|---|
| Order_no | Varchar | 6 |
| Order_date | Date | |
| Client_no | Varchar | 6 |
| S_no | Varchar | 6 |
| Dely_type | Char | 1 |
| Billed_yn | Char | 1 |
| Dely_date | Date | |
| Order_status | Varchar | 10 |

| Order_no | Order_date | Client_no | S_no | Dely_ type | Billed_yn | Dely_date | Order_status |
|---|---|---|---|---|---|---|---|
| O1901 | 06/12/2015 | C001 | S001 | F | N | 06/20/2015 | InProcess |
| O1902 | 01/25/2015 | C002 | S002 | P | N | 06/27/2015 | Cancelled |
| O4665 | 02/18/2015 | C003 | S003 | F | Y | 02/20/2015 | Fullfilled |
| O1903 | 04/03/2015 | C001 | S001 | F | Y | 04/07/2015 | Fullfilled |
| O4666 | 05/20/2015 | C004 | S002 | P | N | 05/22/2015 | Cancelled |
| O1908 | 05/24/2015 | C005 | S003 | F | N | 05/26/2015 | InProcess |

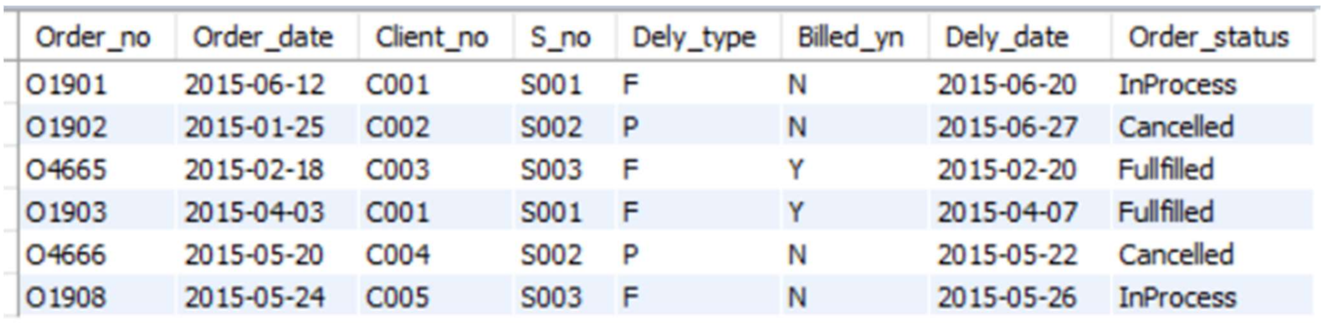
## Solution Query:

- CREATE TABLE sales_order
```
(
        Order_no Varchar (6),
        Order_date Date,
        Client_no Varchar (6),
        S_no Varchar (6),
        Dely_type Char (1),
        Billed_yn Char (1),
        Dely_date Date,
        Order_status Varchar (10)
);
```

- INSERT INTO sales_order
```
Values
   ('O1901', '2015-06-12', 'C001', 'S001', 'F', 'N', '2015-06-20', 'InProcess'),
   ('O1902', '2015-01-25', 'C002', 'S002', 'P', 'N', '2015-06-27', 'Cancelled'),
   ('O4665', '2015-02-18', 'C003', 'S003', 'F', 'Y', '2015-02-20', 'Fullfilled'),
   ('O1903', '2015-04-03', 'C001', 'S001', 'F', 'Y', '2015-04-07', 'Fullfilled'),
   ('O4666', '2015-05-20', 'C004', 'S002', 'P', 'N', '2015-05-22', 'Cancelled'),
   ('O1908', '2015-05-24', 'C005', 'S003', 'F', 'N', '2015-05-26', 'InProcess');
```

- SELECT * FROM sales_order;

## Output:



| Order_no | Order_date | Client_no | S_no | Dely_type | Billed_yn | Dely_date | Order_status |
|---|---|---|---|---|---|---|---|
| O1901 | 2015-06-12 | C001 | S001 | F | N | 2015-06-20 | InProcess |
| O1902 | 2015-01-25 | C002 | S002 | P | N | 2015-06-27 | Cancelled |
| O4665 | 2015-02-18 | C003 | S003 | F | Y | 2015-02-20 | Fullfilled |
| O1903 | 2015-04-03 | C001 | S001 | F | Y | 2015-04-07 | Fullfilled |
| O4666 | 2015-05-20 | C004 | S002 | P | N | 2015-05-22 | Cancelled |
| O1908 | 2015-05-24 | C005 | S003 | F | N | 2015-05-26 | InProcess |

Solve the following queries using the database given in practical 1 and above table.

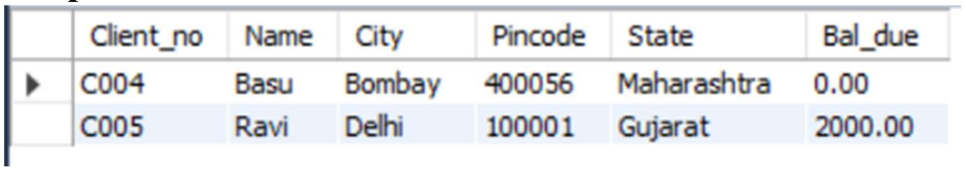
## Queries on computation on table data:

1. Find the name of all clients having 'a ' as the second letter in their names

### Solution Query:

- SELECT * FROM client_master where Name Like '_a%';

### Output:



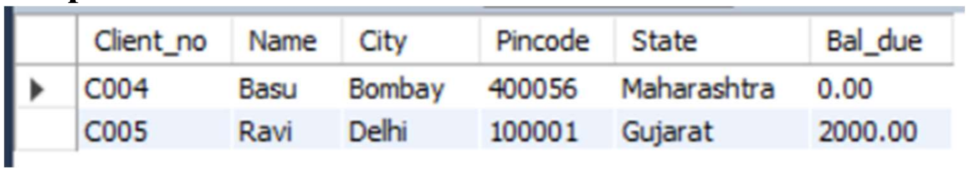
| | Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|---|
| ▶ | C004 | Basu | Bombay | 400056 | Maharashtra | 0.00 |
| | C005 | Ravi | Delhi | 100001 | Gujarat | 2000.00 |

2. Find out the clients whose name is four character ling and second letter is 'a'.

### Solution Query:

- SELECT * FROM client_master where Name Like '_a__';

### Output:



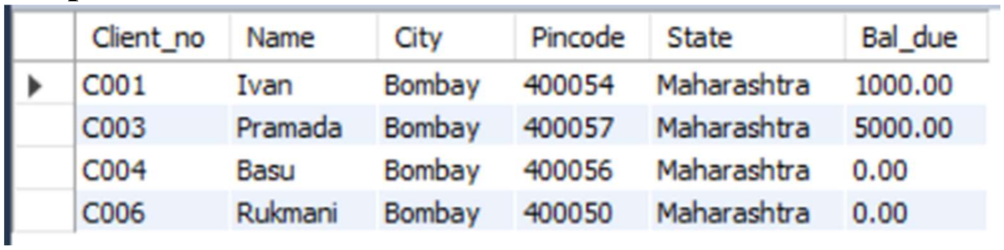
| | Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|---|
| ▶ | C004 | Basu | Bombay | 400056 | Maharashtra | 0.00 |
| | C005 | Ravi | Delhi | 100001 | Gujarat | 2000.00 |

3. Find out the name of city whose second last character is 'a'.

### Solution Query:

- SELECT * FROM client_master where City Like '%a_';

### Output:



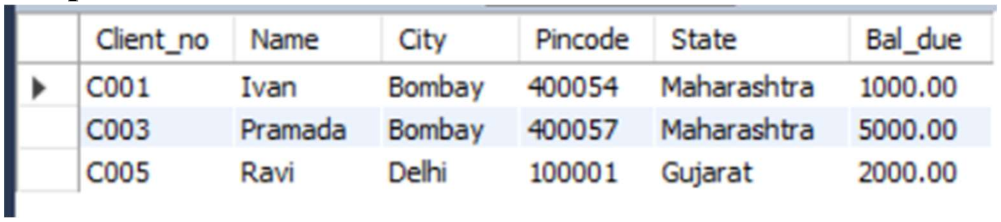
| | Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|---|
| ▶ | C001 | Ivan | Bombay | 400054 | Maharashtra | 1000.00 |
| | C003 | Pramada | Bombay | 400057 | Maharashtra | 5000.00 |
| | C004 | Basu | Bombay | 400056 | Maharashtra | 0.00 |
| | C006 | Rukmani | Bombay | 400050 | Maharashtra | 0.00 |

4. Print the list of clients whose bal_due is greater than or equal to 10000.

**Solution Query:**

- SELECT * FROM client_master where Bal_due >= 1000;

**Output:**



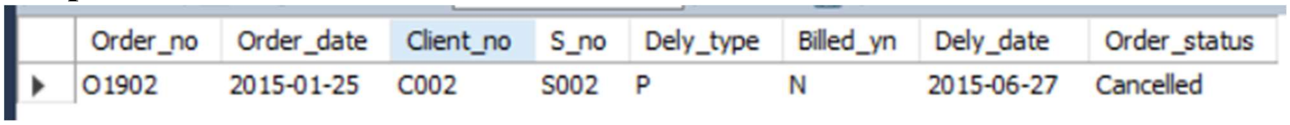
| | Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|---|
| ▶ | C001 | Ivan | Bombay | 400054 | Maharashtra | 1000.00 |
| | C003 | Pramada | Bombay | 400057 | Maharashtra | 5000.00 |
| | C005 | Ravi | Delhi | 100001 | Gujarat | 2000.00 |

5. Print the information from sales_order table for orders placed in the month of January.

**Solution Query:**

- d

**Output:**



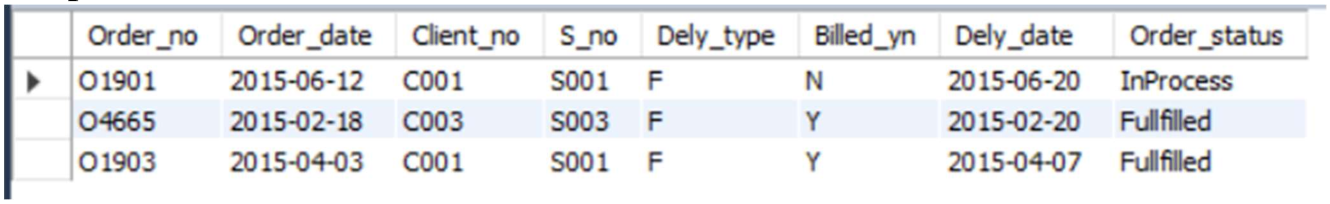
| | Order_no | Order_date | Client_no | S_no | Dely_type | Billed_yn | Dely_date | Order_status |
|---|---|---|---|---|---|---|---|---|
| ▶ | O1902 | 2015-01-25 | C002 | S002 | P | N | 2015-06-27 | Cancelled |

6. Display the order information for client_no ‘C003’ and ‘C001’.

**Solution Query:**

- SELECT * FROM sales_order where (Client_no = 'C003' OR Client_no = 'C001');

**Output:**



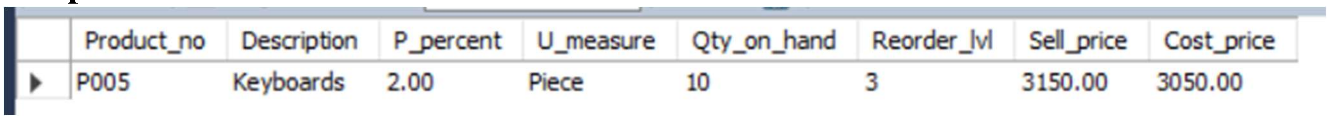
| | Order_no | Order_date | Client_no | S_no | Dely_type | Billed_yn | Dely_date | Order_status |
|---|---|---|---|---|---|---|---|---|
| ▶ | O1901 | 2015-06-12 | C001 | S001 | F | N | 2015-06-20 | InProcess |
| | O4665 | 2015-02-18 | C003 | S003 | F | Y | 2015-02-20 | Fullfilled |
| | O1903 | 2015-04-03 | C001 | S001 | F | Y | 2015-04-07 | Fullfilled |

7. Find products whose selling price is greater than 2000 and less than or equal to 5000.

**Solution Query:**

➢ SELECT * FROM product_master where (sell_price > 2000 AND sell_price <= 5000);

**Output:**



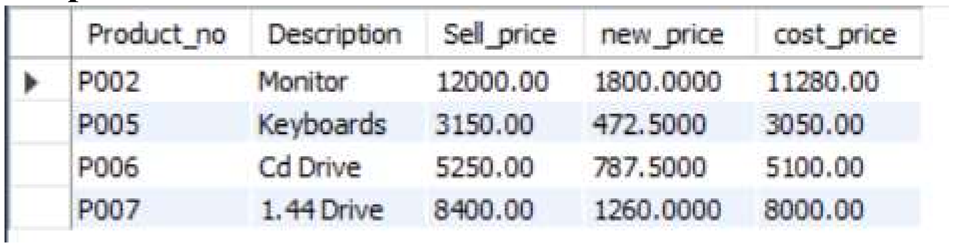
| Product_no | Description | P_percent | U_measure | Qty_on_hand | Reorder_lvl | Sell_price | Cost_price |
|---|---|---|---|---|---|---|---|
| P005 | Keyboards | 2.00 | Piece | 10 | 3 | 3150.00 | 3050.00 |

8. Find products whose selling price is more than 1500. Calculate a new selling price as, original selling price * .15. Rename the new column in the above query as new_price.

**Solution Query:**

➢ SELECT Product_no, Description, Sell_price, Sell_price * 0.15 AS new_price, cost_price FROM product_master where sell_price > 1500;

**Output:**



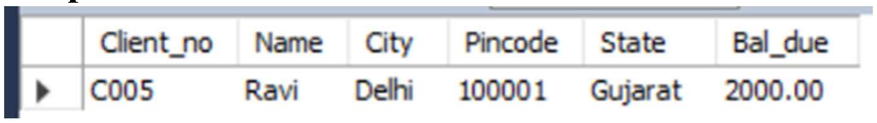
| Product_no | Description | Sell_price | new_price | cost_price |
|---|---|---|---|---|
| P002 | Monitor | 12000.00 | 1800.0000 | 11280.00 |
| P005 | Keyboards | 3150.00 | 472.5000 | 3050.00 |
| P006 | Cd Drive | 5250.00 | 787.5000 | 5100.00 |
| P007 | 1.44 Drive | 8400.00 | 1260.0000 | 8000.00 |

9. List the names, city and state of clients who are not in the state of 'Maharashtra'.

**Solution Query:**

➢ SELECT * FROM client_master WHERE NOT ( state = 'Maharashtra' );

**Output:**



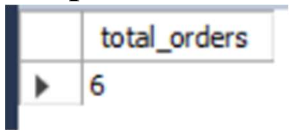
| Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|
| C005 | Ravi | Delhi | 100001 | Gujarat | 2000.00 |

10. Count the total number of orders.

## Solution Query:

- SELECT COUNT( Order_no ) AS total_orders FROM sales_order;

## Output:



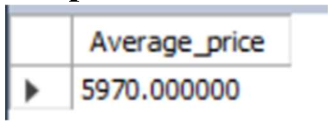
| | total_orders |
|---|---|
| ▶ | 6 |

11. Calculate the average price of all products.

## Solution Query:

- SELECT AVG( sell_price ) As Average_price from product_master;

## Output:



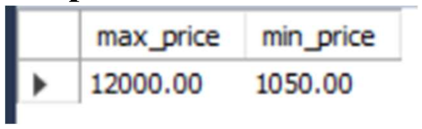
| | Average_price |
|---|---|
| ▶ | 5970.000000 |

12. Determine the maximum and minimum product prices. Rename the output as max_price and min_price respectively.

## Solution Query:

- SELECT MAX( sell_price ) As max_price, MIN( sell_price ) As min_price from product_master;

## Output:



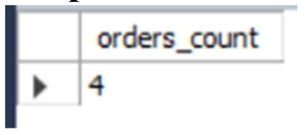
| | max_price | min_price |
|---|---|---|
| ▶ | 12000.00 | 1050.00 |

13. Count the number of products having price greater than or equal to 1500.

## Solution Query:

- ➢ SELECT COUNT( * ) AS orders_count FROM product_master where sell_price >= 1500;

## Output:



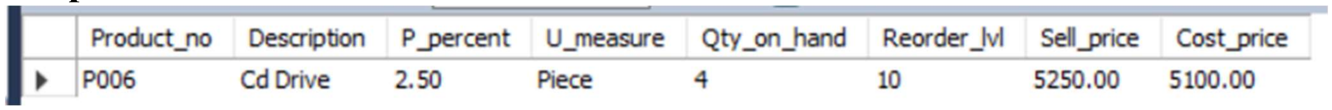
| orders_count |
|---|
| 4 |

14. Find all the products whose qty_on_hand is less than reorder level.

## Solution Query:

- ➢ SELECT * FROM Product_master WHERE Qty_on_hand < Reorder_lvl;

## Output:



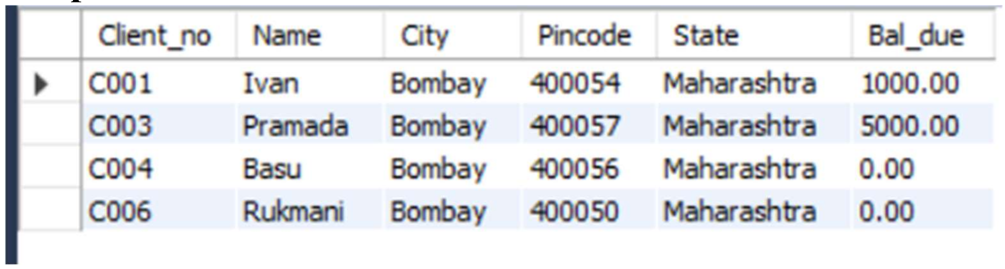
| Product_no | Description | P_percent | U_measure | Qty_on_hand | Reorder_lvl | Sell_price | Cost_price |
|---|---|---|---|---|---|---|---|
| P006 | Cd Drive | 2.50 | Piece | 4 | 10 | 5250.00 | 5100.00 |

15. Create table cmaster from client_master table.
16. Insert data in cmaster from client_master where city='bombay'

## Solution Query:

- ➢ CREATE TABLE cmaster AS SELECT * FROM Client_master WHERE City = 'Bombay';
- ➢ SELECT * FROM cmaster;

## Output:



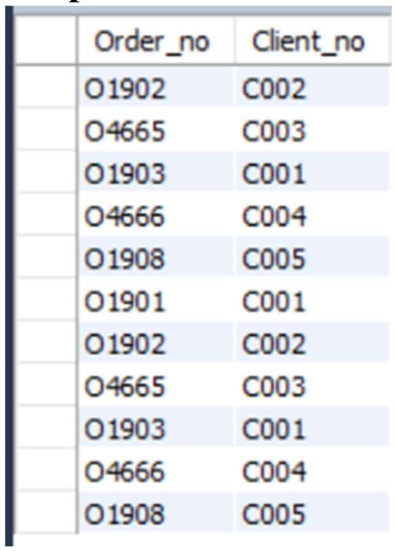
| Client_no | Name | City | Pincode | State | Bal_due |
|---|---|---|---|---|---|
| C001 | Ivan | Bombay | 400054 | Maharashtra | 1000.00 |
| C003 | Pramada | Bombay | 400057 | Maharashtra | 5000.00 |
| C004 | Basu | Bombay | 400056 | Maharashtra | 0.00 |
| C006 | Rukmani | Bombay | 400050 | Maharashtra | 0.00 |

17. Create table sales from sales_order with order_no and client_no columns.
18. Insert data in sales from sales_order table.

## Solution Query:

- CREATE TABLE sales AS SELECT Order_no, Client_no FROM Sales_order;
- SELECT * FROM sales;

## Output:



| Order_no | Client_no |
|---|---|
| O1902 | C002 |
| O4665 | C003 |
| O1903 | C001 |
| O4666 | C004 |
| O1908 | C005 |
| O1901 | C001 |
| O1902 | C002 |
| O4665 | C003 |
| O1903 | C001 |
| O4666 | C004 |
| O1908 | C005 |

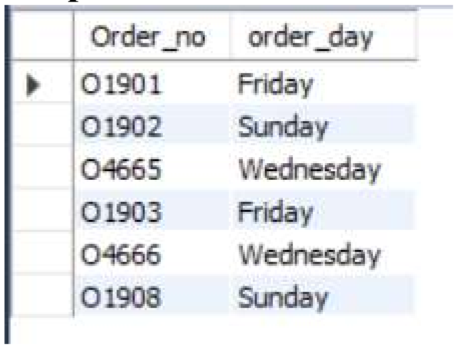
**Queries on Date manipulation:**

1) Display the order number and day on which clients placed their order.

## Solution Query:

➢ SELECT Order_no, DAYNAME(Order_date) AS order_day FROM sales_order;

## Output:



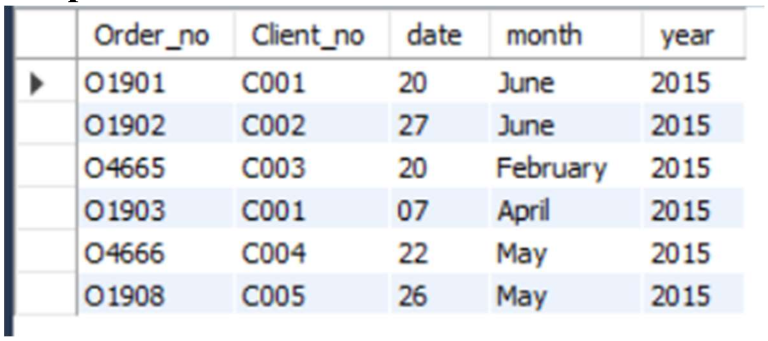
| Order_no | order_day |
|---|---|
| O1901 | Friday |
| O1902 | Sunday |
| O4665 | Wednesday |
| O1903 | Friday |
| O4666 | Wednesday |
| O1908 | Sunday |

2) Display the month (in alphabets) and date when the order must be delivered.

## Solution Query:

➢ SELECT Order_no, Client_no, DATE_FORMAT(Dely_date, '%d') AS date, DATE_FORMAT(Dely_date, '%M') AS month, DATE_FORMAT(Dely_date, '%Y') AS year FROM sales_order;

## Output:



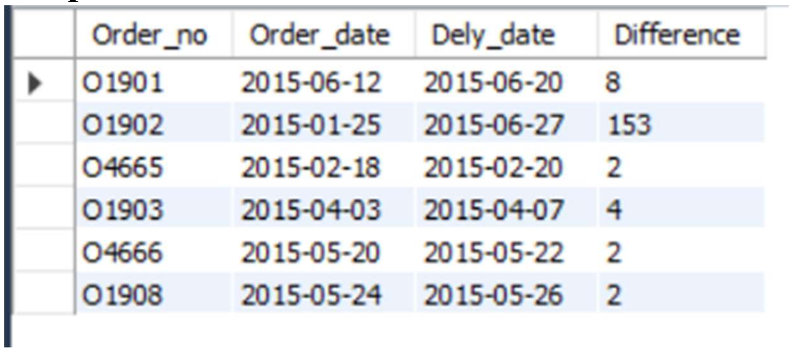
| Order_no | Client_no | date | month | year |
|---|---|---|---|---|
| O1901 | C001 | 20 | June | 2015 |
| O1902 | C002 | 27 | June | 2015 |
| O4665 | C003 | 20 | February | 2015 |
| O1903 | C001 | 07 | April | 2015 |
| O4666 | C004 | 22 | May | 2015 |
| O1908 | C005 | 26 | May | 2015 |

3) Find the number of days elapsed between delivery date and order date from sales_order table.

## Solution Query:

- SELECT Order_no, Order_date, Dely_date, DATEDIFF(Dely_date, Order_date) AS Difference FROM sales_order;

## Output:



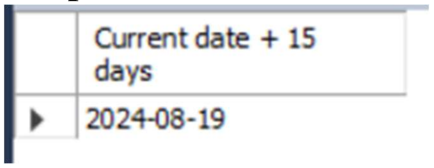
| Order_no | Order_date | Dely_date | Difference |
|---|---|---|---|
| O1901 | 2015-06-12 | 2015-06-20 | 8 |
| O1902 | 2015-01-25 | 2015-06-27 | 153 |
| O4665 | 2015-02-18 | 2015-02-20 | 2 |
| O1903 | 2015-04-03 | 2015-04-07 | 4 |
| O4666 | 2015-05-20 | 2015-05-22 | 2 |
| O1908 | 2015-05-24 | 2015-05-26 | 2 |

4) Find the date, 15 days after today's date.

## Solution Query:

- SELECT DATE_ADD(CURRENT_DATE(), interval 15 DAY) AS 'Current date + 15 days';

## Output:



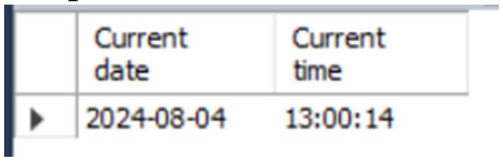
| Current date + 15 days |
|---|
| 2024-08-19 |

5) Display current date and time.

## Solution Query:

- SELECT CURRENT_DATE() AS 'Current date', CURRENT_TIME() AS 'Current time';

## Output:



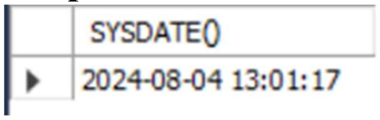
| Current date | Current time |
|---|---|
| 2024-08-04 | 13:00:14 |

6) Display system time.

## Solution Query:

- SELECT SYSDATE();

## Output:



```
SYSDATE()
2024-08-04 13:01:17
```