**Ganpat University**

**Faculty of Engineering & Technology**

**Computer Science & Engineering**

**Practical\_3­**

***Name:- Dwij Vatsal Desai***

***Sem:- 3***

***Sub: - DBMS­­***

***Enrollment No.:- 23162121027***

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:

#### 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;

#### 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);

#### 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.

### 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:

### 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%’;

### IN

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’ );

### NOT IN

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);

### IS NULL

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 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 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 | Products 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 | Total 9 |
| **MAX** | Syntax | MAX([DISTINCT|ALL]expr) |
| Purpose | Return maximum value of ‘expr’. |
| Example | Select MAX(bal\_due) “Maximum” from c\_master; |
| Output | Maximum 15000 |
| **SUM** | Syntax | SUM([DISTINCT|ALL]n) |
| Purpose | Return Sum of values of ‘n’. |
| Example | Select SUM(bal\_due) “Balance” from c\_master; |
| Output | Balance  22000 |

**Numeric Functions:**

|  |  |  |
| --- | --- | --- |
| **ABS** | Syntax | ABS(n) |
| Purpose | Return the absolute values of ‘n’. |
| Example | Select ABS(-15) “Absolute” from dual; |
| Output | Absolute 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 | Raised 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 | Round 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 | Square root  5 |

**String Functions:**

|  |  |  |
| --- | --- | --- |
| **LOWER** | Syntax | LOWER(char) |
| Purpose | Return char, with all letters in lowercase. |
| Example | Select LOWER(‘XYZ’) “Lower” from dual; |
| Output | Lower xyz |
| **INITCAP** | Syntax | INITCAP(char) |
| Purpose | Return STRING with first letter in upper case. |
| Example | Select INITCAP(‘COMP DEPT’) “Title Case” from  dual; |
| Output | Title Case Comp Dept |
| **UPPER** | Syntax | UPPER(char) |
| Purpose | Return char, with all letters in uppercase. |
| Example | Select UPPER(‘xyz’) “Upper” from dual; |
| Output | Upper 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 | Substring CURE |

|  |  |  |
| --- | --- | --- |
| **LENGTH** | Syntax | LENGTH(char) |
| Purpose | Return the length of character. |
| Example | Select LENGTH(‘xyz’) “Length” from dual; |
| Output | Length 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  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 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  \*\*\*\*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  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));  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  $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  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  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  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  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 |

Query for TABLE:

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');

insert into sales\_order values('O1902', '2015-01-25', 'C002', 'S002', 'P', 'N', '2015-06-27', 'Cancelled');

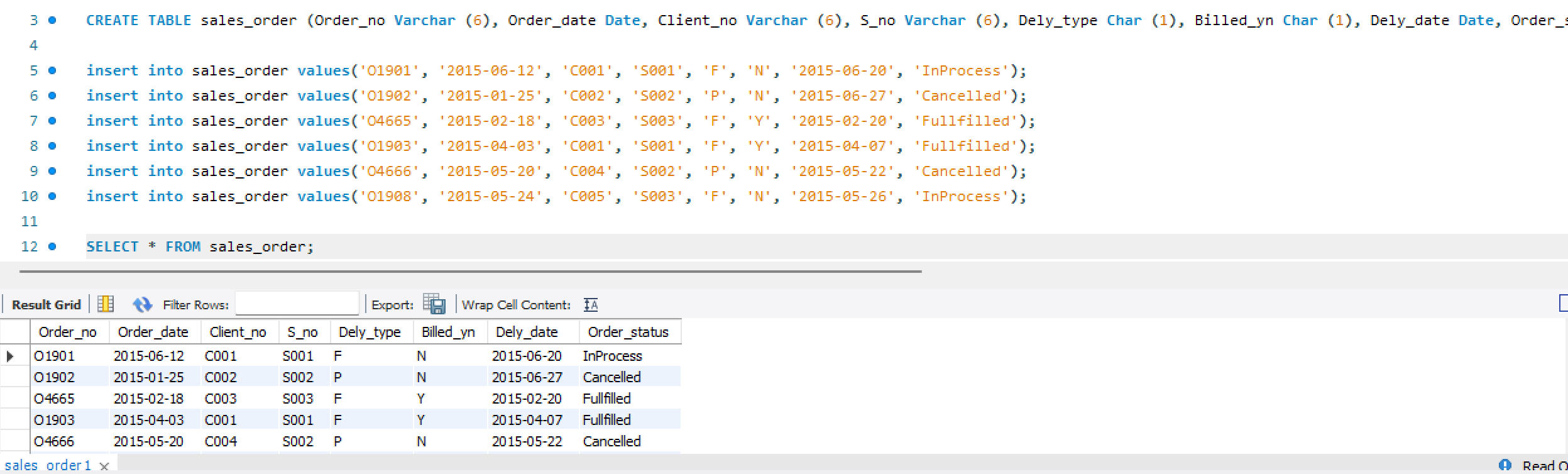
insert into sales\_order values('O4665', '2015-02-18', 'C003', 'S003', 'F', 'Y', '2015-02-20', 'Fullfilled');

insert into sales\_order values('O1903', '2015-04-03', 'C001', 'S001', 'F', 'Y', '2015-04-07', 'Fullfilled');

insert into sales\_order values('O4666', '2015-05-20', 'C004', 'S002', 'P', 'N', '2015-05-22', 'Cancelled');

insert into sales\_order values('O1908', '2015-05-24', 'C005', 'S003', 'F', 'N', '2015-05-26', 'InProcess');

SELECT \* FROM sales\_order;

Image: 

## 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%';

# Image:

# 

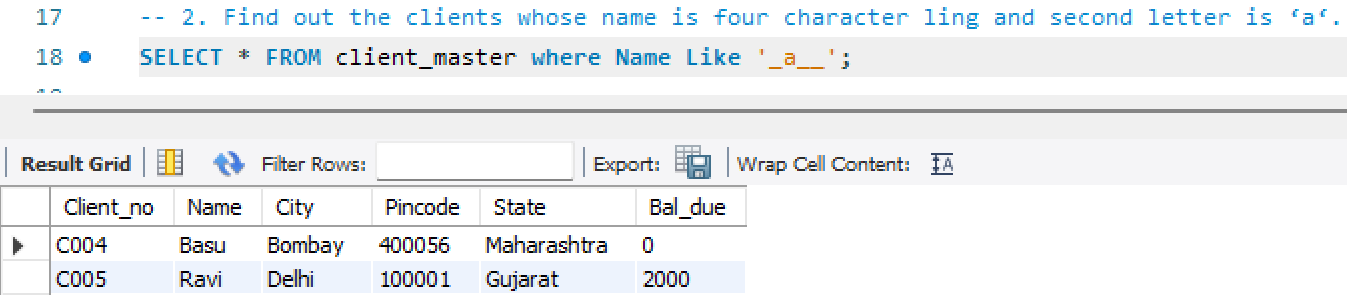
1. **Find out the clients whose name is four character ling and second letter**

**is ‘a‘.**

# Solution Query:

* + SELECT \* FROM client\_master where Name Like '\_a ';

# Image:

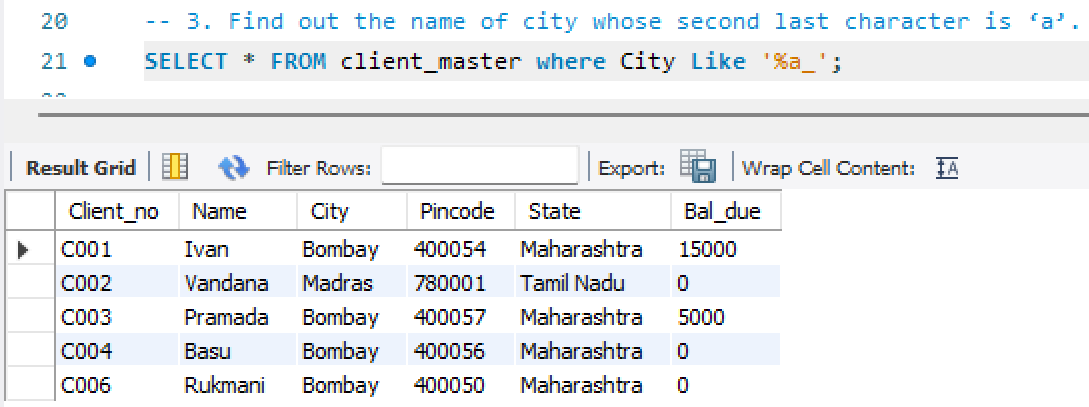


1. **Find out the name of city whose second last character is ‘a’.**

# Solution Query:

* + SELECT \* FROM client\_master where City Like '%a\_';

# Image:



1. **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;

# Image:

# 

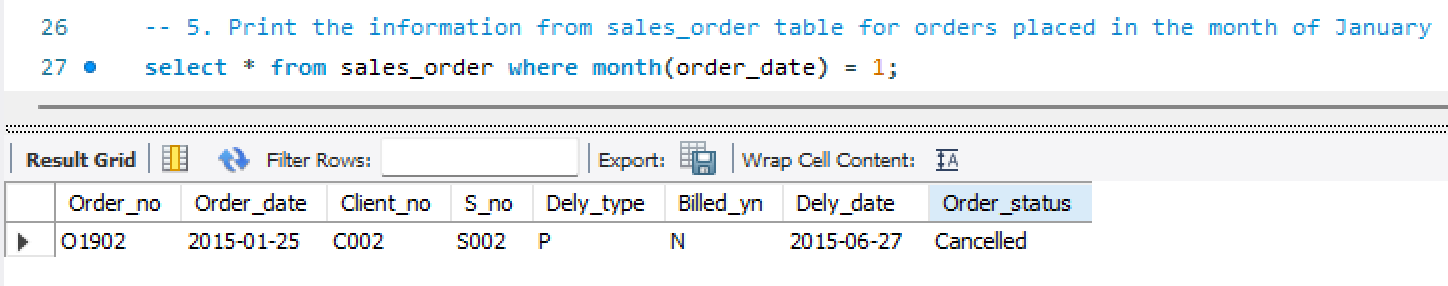
1. **Print the information from sales\_order table for orders placed in the**

**month of January.**

# Solution Query:

* + select \* from sales\_order where month(order\_date) = 1;

# Image:

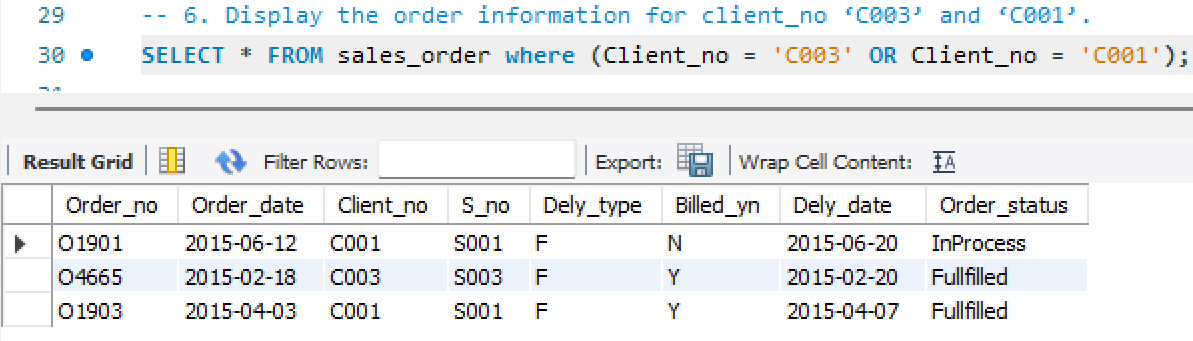
****

1. **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:



1. **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);

# Image:

1. **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;

# Image:

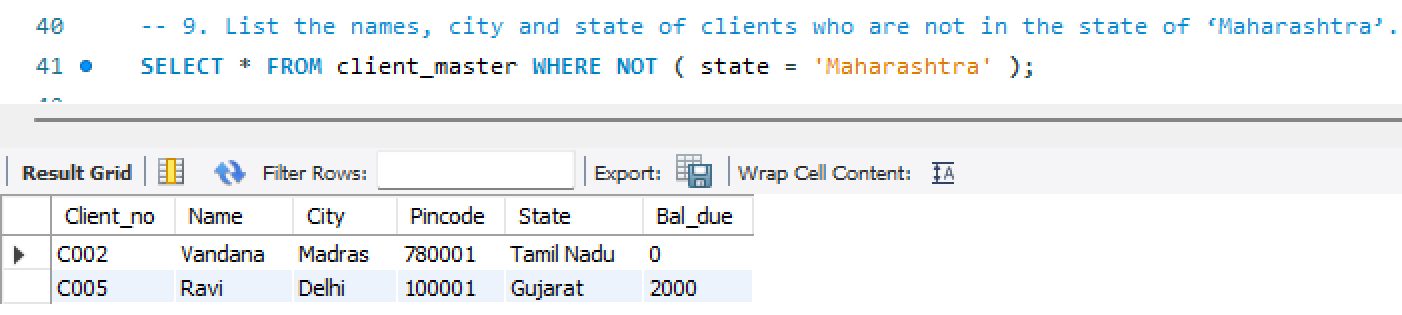
# 

1. **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' );

# Image:



1. **Count the total number of orders.**

# Solution Query:

* + SELECT COUNT( Order\_no ) AS total\_orders FROM sales\_order;

# Image:

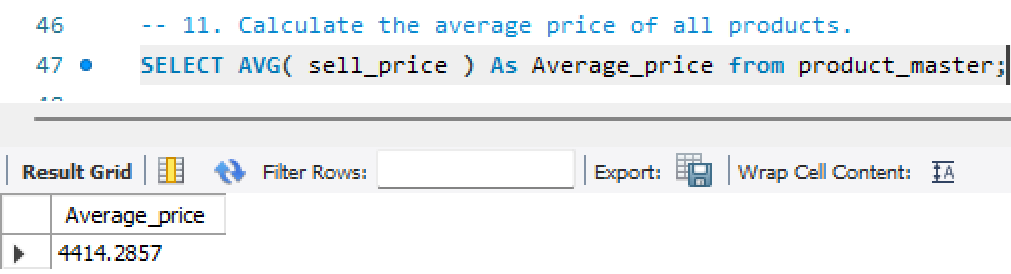
# 

1. **Calculate the average price of all products.**

# Solution Query:

* + SELECT AVG( sell\_price ) As Average\_price from product\_master;

# Image:



1. **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;

# Image:

# 

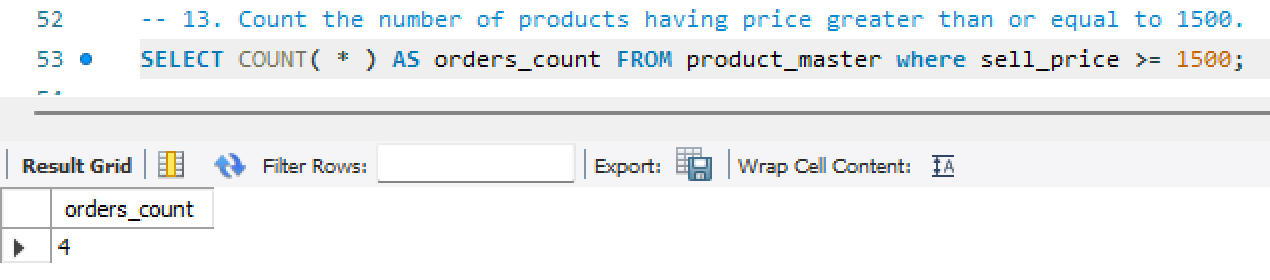
1. **Count the number of products having price greater than or equal to**

**1500.**

# Solution Query:

* + SELECT COUNT( \* ) AS orders\_count FROM product\_master where sell\_price >= 1500;

# Image:

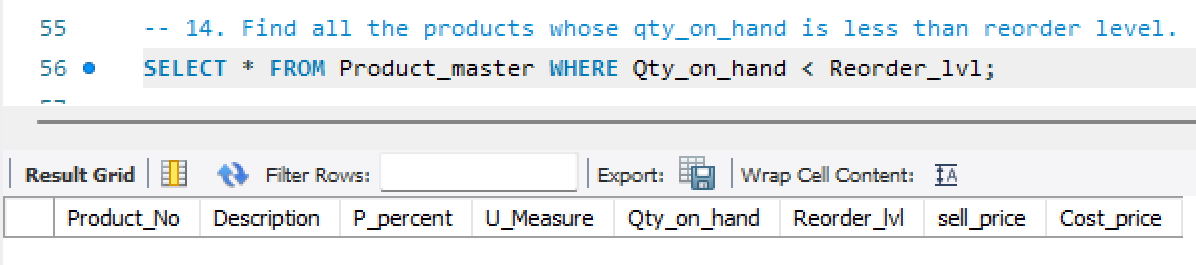
****

1. **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;

# Image:

****

1. **Create table cmaster from client\_master table.**

# Solution Query:

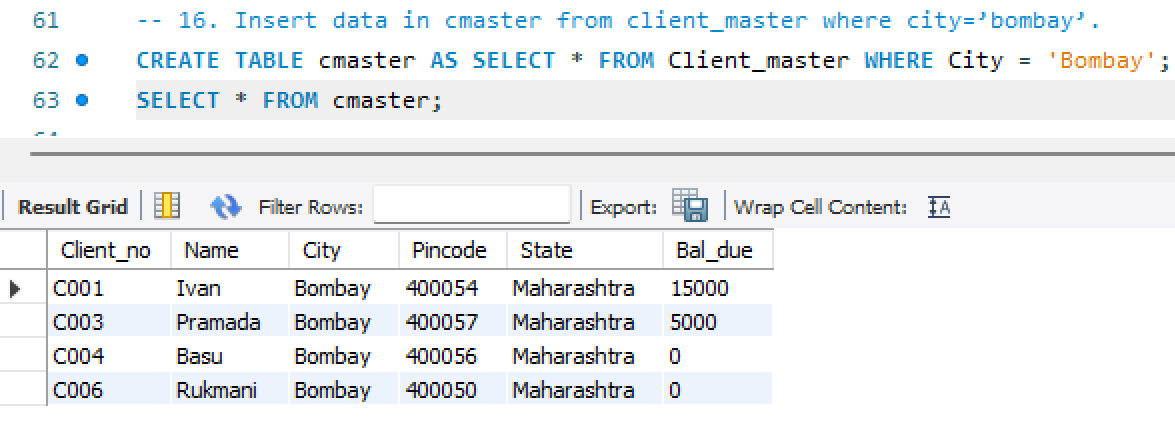
* + CREATE TABLE cmaster AS SELECT \* FROM Client\_master WHERE City = 'Bombay';

1. 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;

# Image:



1. Create table sales from sales\_order with order\_no and client\_no columns.

# Solution Query:

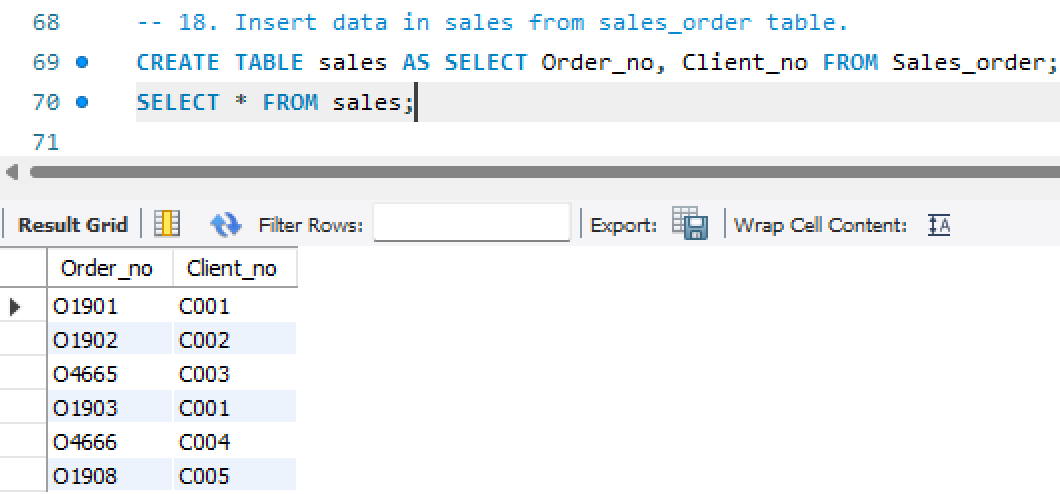
* + CREATE TABLE sales AS SELECT Order\_no, Client\_no FROM Sales\_order;

1. 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;

# Image:



## 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;

# Image:

# 

1. **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;

# Image:

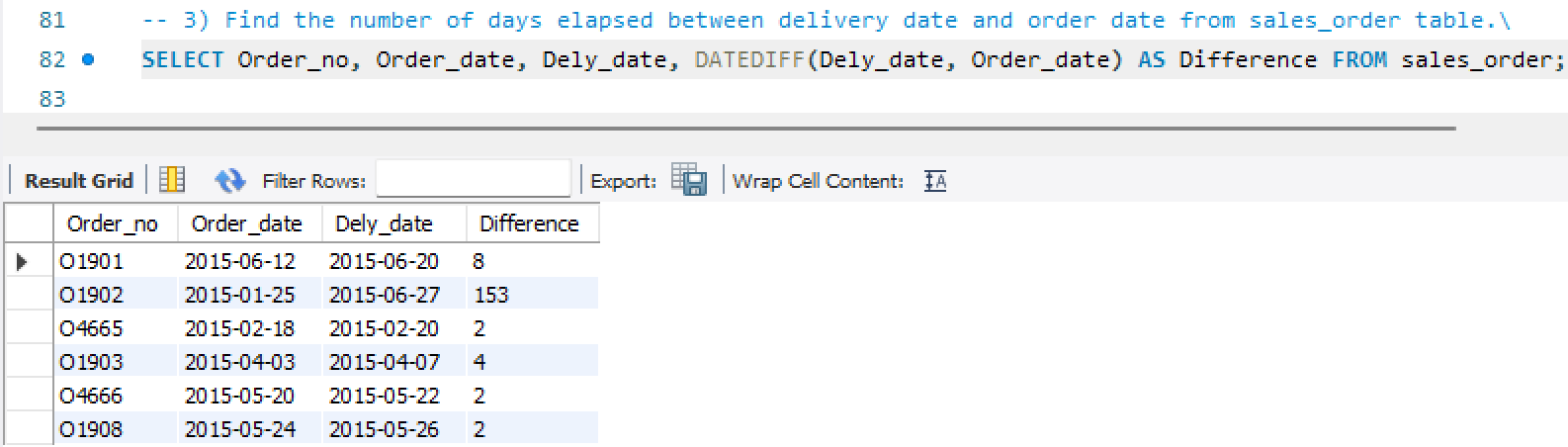
# 

1. **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;

# Image:

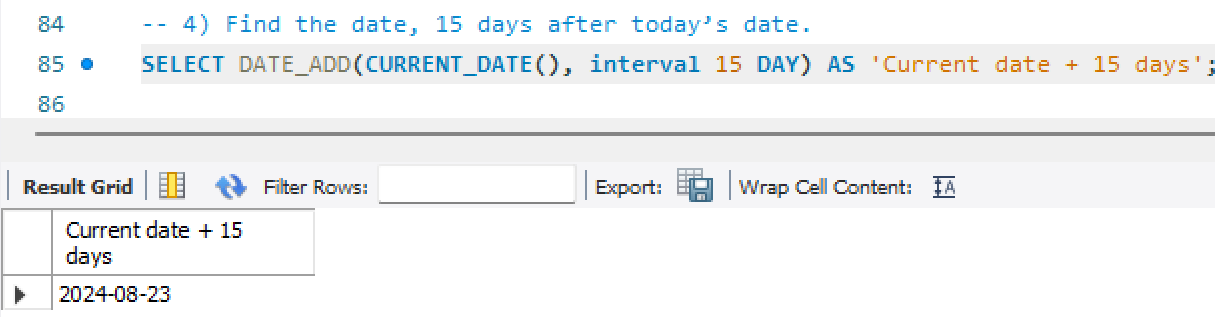
****

1. **Find the date, 15 days after today’s date.**

# Solution Query:

* + SELECT DATE\_ADD(CURRENT\_DATE(), interval 15 DAY) AS 'Current date + 15 days';

# Image:

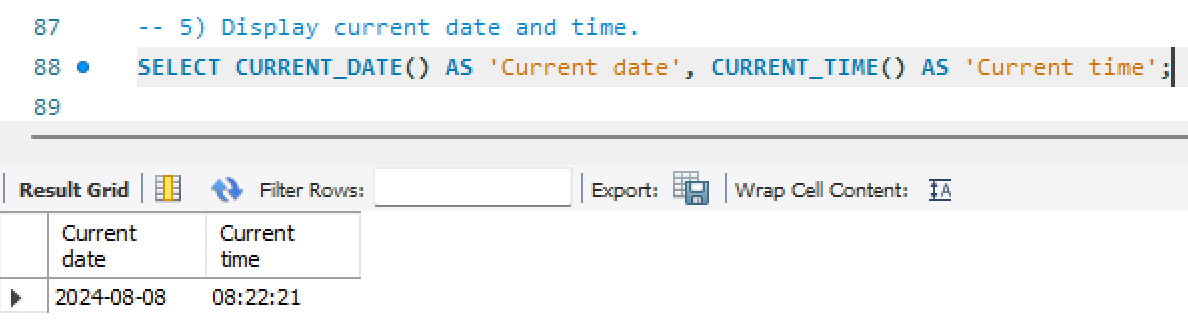
****

1. **Display current date and time.**

# Solution Query:

* + SELECT CURRENT\_DATE() AS 'Current date', CURRENT\_TIME() AS 'Current time';

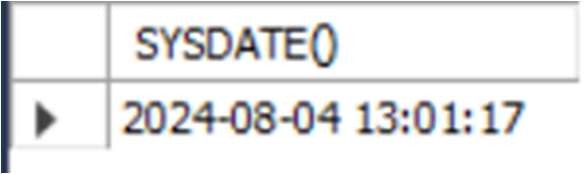
# Image:



1. Display system time.

# Solution Query:

* + SELECT SYSDATE();

**Output:**