**MySQL clauses, Function, Operators**

♣ MySQL clause-where distinct, from, order by, group by, having, rollup

♣ MySQL Function-Aggregate functions, Math functions, String functions, Date and Time

functions, control flow functions and expressions,

comparison functions, Window functions

♣ MySQL operators- and, or, and or, like, in, not, is null, is not null, between, comparison operator

------------------------------------------------------------------------------------------------------------------------

* **MySQL operators**

The symbols which are used to perform logical and mathematical operations in MySQL are called MySQL operators. MySQL reserved words or characters are known as operators.

Three types of operators in MySQL:

1. MySQL Arithmetic Operators(+,-,/,\*)
2. MySQL Comparison Operators (<,>,=,!=,<=,>=,==)
3. MySQL Logical Operators ( and, or)

* **MySQL Arithmetic Operators:**

Arithmetic operators in SQL are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in statements.

**DUAL table**: It is a system defined table which contains only one column to perform calculations on users own data.

|  |  |
| --- | --- |
| **Arithmetic Operators** | **Example** |
| + (Addition) | Select 10+2 from dual; |
| – (Subtraction) | Select 10-2 from dual; |
| \* (multiplication) | Select 10\*2 from dual; |
| / (Division) | Select 10/2 from dual; |
| % (Modulus) | Select 10%2 from dual; |

select (90000\*10)/100 "10% of 90000" from dual;

select sal " Basic Sal", (0.02\*sal) " TA" from emp;

* **MySQL Comparison Operators:** used to compare values by specifying conditions on the columns.

|  |  |
| --- | --- |
| **Operators** | **Example/Description** |
| > | x > y (x is greater than y) |
| < | x < y (x is less than y) |
| >= | x >= y (x is greater than or equal to y) |
| <= | x <= y (x is less than or equal to y) |
| = | x == y (x is equal to y) |
| != or < > | x != y or x <> y (x is not equal to y) |
| !< | x !< y (x is not less than y) |
| !> | x !> y (x is not greater than y) |

* **MySQL Logical Operators:**

Logical operators in SQL are used to perform logical operations on the given expressions in SQL statements. There are many operators in SQL which are used in SQL statements in the WHERE clause. They are

|  |  |
| --- | --- |
| **Operator** | **Description** |
| AND | Display output if all conditions are true. If any one condition was failed then it will not display output. |
| OR | Display output if anyone condition is true. If all conditions are false then it will not display output. |
| BETWEEN | The **BETWEEN** operator is used to get values within range. |
| IN | The **IN** operator is used to search for specified value matches any value in set of multiple values. |
| ALL | The **ALL** operator returns true when value matches with all values in single column set of values. It’s like **AND operator** it will compare value against with all values in column. |
| ANY | The **Any** operator in SQL returns true when value matches with any value in single column set of values. It’s like **OR operator** it will compare value against with any value in column. |
| LIKE | The **LIKE** operator is used to search for character string with specified pattern using wildcards in column. |
| EXISTS | The **EXISTS** operator is used to show result if subquery returns data. |
| NOT | The **NOT** operator is a negate operator that means it will show data for opposite of conditions what we mentioned in SQL statement. |
| SOME | The **SOME** operator is used to compare value with a single column set of values returned by subquery. **SOME** must match at least one value in subquery and that value must be preceded by comparison operators. |

* **USING Comparison Operators =, >, <, >=, <=, !=, <>**

**Example:** select \* from student where no = 2;

select \* from student where no < 4;

select \* from student where no > 3;

select \* from student where no <= 7;

select \* from student where no >= 3;

select \* from student where no != 2;

select \* from student where no <> 2;

* **AND Operator:** returns output when all the conditions become true.

**Syntax:** select \* from <table\_name> where <condition1> and <condi2>and<condition n>;

**Example:** select \* from student where no=2 **and** marks>=200;

* **OR Operator:** returns output when either one of the conditions becomes true.

**Syntax:** select \* from <table\_name> where <condition1>and<condition2> or <conditionn>;

**Example:** select \* from student where no=2 **or** marks>=200;

* **NOT** will negate the condition. It will select the records where condition is not satisfied.
* **Range Searching:** Oracle provides 3 types for searching data in a range.
  1. BETWEEN
  2. IN
  3. IS

**1) Between Operator**

* BETWEEN operator is used to select data that is within a range of values.
* BETWEEN operator that contain specified lower and upper limit values.
* Two values in the range must be linked with keyword AND.
* BETWEEN operators can be used with numeric and character data types.

**Syntax:** select \* from <table\_name> where <col> between <lower bound> and <upper bound>;

**Example:** select \* from student where marks between 200 **and** 400;

select empno, ename, deptno from emp where deptno>=15 **and** deptno<=25;

* **Not Between Operator:** used for to select data which is not within the range.

**Syntax:** select \* from <table\_name> where <col> not between <lower bound> and <upper bound>;

**Example:** select \* from student where marks not between 200 and 400;

select \* from emp where empno **not between** 10 and 20;

**2) In Operator**

* If a value is required to be compared with list of values then IN predicate is used.
* IN operator reduces the need to use multiple OR conditions.
* Arithmetic operator (=) compares a single value to another single value.

**Syntax:** select \* from <table\_name> where <col> in (value1, value2, value3…valuen);

**Example:** select \* from student where no **in** (1, 2, 3);

select empno, ename, deptno from emp where deptno 10 **or** deptno=20;

select empno, ename, deptno from empwhere deptno **in (20,30);**

* **Not In Operator:** Not In operator is usd to select all the rows where value does not match in the list of values specified

**Syntax:** select \* from <table\_name> where <col> not in(value1, value2,value3…valuen);

**Example:** select \* from student where no **not in** (1, 2, 3);

select empno, ename, deptno from emp where deptno **not in** (20,30);

* **Is Null Operator:** used to check the column value is null or not, if it is null display output

**Syntax:** select \* from <table\_name> where <col> is null;

**Example:** select \* from student where marks **is null**;

1. If expression is a NULL value, the condition evaluates to TRUE.
2. If expression is not a NULL value, the condition evaluates to FALSE.

* **Is Not Null Operator:** used to test for NOT NULL value in the specified column.

**Syntax:** select \* from <table\_name> where <col> is not null;

**Example:** select \* from student where marks **is not null**;

* **Like Operator**

Like operator is used to retrieve data based on specify pattern or characters. Like operator selects those rows only containing fields that match specified portions of character strings. LIKE is used with char, varchar, text, date time and small date time data. A wildcard allows the user to match fields that contain certain letters.

**Syntax:** select \* from <table\_name> where <col> like <pattern>;

|  |  |
| --- | --- |
| **%** | It matches zero or more any characters. |
| **\_** | Any single character search with the specified pattern |
| **[ ]** | Any single character within the specified range e.g.([a-f]) or set [abcdef]) |

|  |  |  |  |
| --- | --- | --- | --- |
| Cno | Cname | Balance | City |
| 1 | Arjun | 12550 | Pandharpur |
| 2 | Ramesh | 16550 | Solapur |
| 3 | Jagan | 18000 | Sangola |
| 4 | Madhu | 19000 | Kolhapur |
| 5 | Varun | 15250 | Sangli |
| 6 | Ritu | 14850 | Satara |

**Example:**

Create table customer(

cno number,

cname varchar(10),

balance number(6)

city varchar(10));

Find customer names whose salary is 15000.

select \* from customer where salary like 15000;

Find customer names whose city name beginning start with ‘Sa’.

select cname from customer where city like 'Sa%';

Find customer names ending with ‘un’.

select \* from customer where cname like '%un';

Find customer name whose name’s second letter start with ‘a’.

select \* from customer where cname like '\_a%';

Display customer details whose balance’s third number start with ‘5’.

select \* from customer where balance like '\_\_5%';

Display customer details whose name’s third letter start with ‘h’ from ending.

select \* from customer where cname like '%\_t%';

Find customer names whose city name start with “S” and end with “a”

Select cname from customer where city like 'S%a';

Find customer names whose name contains 2 a’s.

select \* from customer where cname like '%a%a%';

Find customer name whose city name have “ang” in any position

select cname from customer where city like '%ang%';

* **ORDER BY Clause**

The ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns. Some databases sort query results in ascending order by default.

**Syntax:** select expressions from tables where conditions order by expression [asc | desc];

Sort the result in ascending order by name and salary.

select \* from emp order by ename, salary;

To sort the result in descending order by name.

select \* from emp order by name desc;

* **GROUP BY Clause**
* GROUP BY Statement is used to to organize similar data into groups with the help of some functions. i.e. if a particular column has same values in different rows then it will arrange these rows in a group.
* GROUP BY returns only one result per group of data
* A query containing a group by clause is processed in the following way:

1. Select all rows that satisfy the condition specified in the where clause.
2. Discard all groups that do not satisfy the condition in the having clause.
3. Apply aggregate functions to each group.

**Syntax: select** column1, **aggregate function\_name**(column2) **from** table\_name **where** condition **group by** column1, column2;

1. **Aggregate function\_name**: Name of the function. Example- SUM(),AVG().
2. **table\_name**: Name of the table.
3. **condition**: Condition used.

**Example:** select dno, sum(sal) from emp group by dno;

**Table Store\_Information**

|  |  |  |  |
| --- | --- | --- | --- |
| **Store\_Name** | **Product\_ID** | **Sales** | **Txn\_Date** |
| Sangola | 1 | 1500 | Jan-05-1999 |
| Sangola | 2 | 500 | Jan-05-1999 |
| Pune | 1 | 250 | Jan-07-1999 |
| Sangola | 1 | 300 | Jan-08-1999 |
| Sangli | 1 | 700 | Jan-08-1999 |

* **GROUP BY a single column :** To find total sales for each store:

select store\_name, sum(sales) from store\_information group by store\_name;

|  |  |
| --- | --- |
| **Store\_Name** | **SUM(Sales)** |
| Sangola | 2300 |
| Pune | 250 |
| Sangli | 700 |

* **GROUP BY multiple columns:** To have two or more columns associated with GROUP BY.

To find total sales for each product at each store:

select store\_name, product\_id, sum(sales)from store\_information group by store\_name, product\_id;

|  |  |  |
| --- | --- | --- |
| **Store\_Name** | **Product\_ID** | **SUM(Sales)** |
| Sangola | 1 | 1800 |
| Sangola | 2 | 500 |
| Pune | 1 | 250 |
| Sangli | 1 | 700 |

* **GROUP BY multiple columns and multiple functions**

To find total sales and the average sales for each product at each store:

select store\_name, product\_id, sum(sales), avg(sales) from store\_information group by store\_name, product\_id;

|  |  |  |  |
| --- | --- | --- | --- |
| **Store\_Name** | **Product\_ID** | **SUM(Sales)** | **AVG(Sales)** |
| Sangola | 1 | 1800 | 900 |
| Sangola | 2 | 500 | 500 |
| Pune | 1 | 250 | 250 |
| Sangli | 1 | 700 | 700 |

* **Group by month / date / week:** To find total daily sales from Store\_Information:

select txn\_date, sum(sales)from store\_information group by txn\_date order by txn\_date;

|  |  |
| --- | --- |
| Txn\_Date | SUM(Sales) |
| Jan-05-1999 | 2000 |
| Jan-07-1999 | 250 |
| Jan-08-1999 | 1000 |

Total sales for both Sangola and Sangli stores.

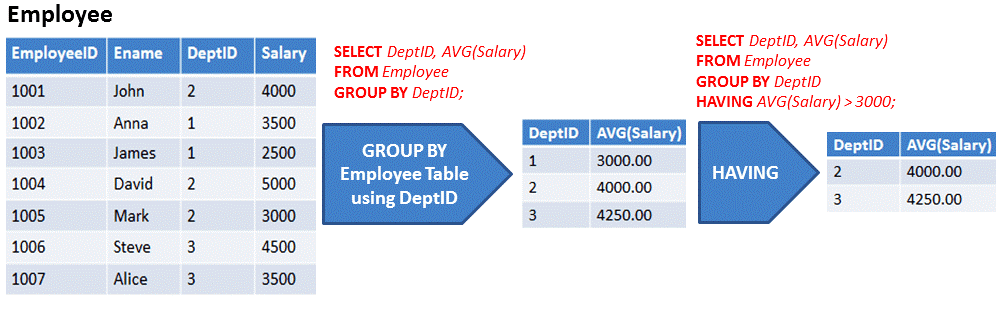
select store\_name,sum(sales) from store\_information where store\_name in('Sangola', 'Sangli') group by store\_name;

* **HAVING Clause**
* We know that WHERE clause is used to place conditions on columns but we want to place conditions on groups then, use HAVING clause.
* The **HAVING** clause is used to filter group of rows based on a specified condition.  It specifies the search condition for the group or aggregate. HAVING caluse is attached with [**GROUP BY**](http://www.1keydata.com/sql/sqlgroupby.html) clause. The HAVING clause works like the [WHERE clause](https://zentut.com/sql-tutorial/sql-where/).
* The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

**Syntax**: **select** column1, function\_name(column2)**from** table\_name **where** condition

**group by** column1, column2 **having** aggregate\_function (<expression>) <operator> <value> **order by** column1, column2;

1. **function\_name**: Name of the function used for example, SUM() , AVG().
2. **table\_name**: Name of the table.
3. **Condition**: Condition used.



Employee table is grouped based on DeptID column and these grouped rows filtered using HAVING Clause with condition AVG(Salary) > 3000.

**Example:** select dno, count(eno) from emp group by dno having count(eno)>2;

To see only the stores with sales over Rs.1500:

select store\_name, sum(sales) from store\_information group by store\_name having sum(sales)>1500;

|  |  |
| --- | --- |
| **Store\_Name** | **SUM(Sales)** |
| **Sangola** | **1800** |

Total sales for both Sangola and Sangli stores.

select store\_name, sum(sales) from store\_information group by store\_name having store\_name in('Sangola', 'Sangli');

|  |  |
| --- | --- |
| **WHERE** | **HAVING** |
| Implemented in row operations. | Implemented in column operations. |
| Applied to Single row | Applied to Summarized row or groups. |
| It only fetches the particular data from particular rows according to the condition. | At first, complete data is fetched then separated according to the condition. |
| Aggregate Functions cannot appear in WHERE clause. | Aggregate Functions can appear in HAVING clause. |
| Used with SELECT and other statements such as UPDATE, DELETE etc. | Used with SELECT statement. Can't be used other statements. |
| Act as a Pre-filter | Act as a Post-filter |
| GROUP BY Comes after WHERE. | GROUP BY Comes before HAVING. |

* **MySQL Functions**

Oracle functions can be used to manipulate data and returning the result. A function is performs a

specific task. A function can accept user-supplied variables or constants and

perform operation on them. Such variables / constants are known as *arguments.*

Function\_Name [ (argument-1, argument-2,.) ]

Functions are

* 1. Scalar Function (Single Row Function)
  2. Aggregate (Group Function)
  3. Miscellaneous Function
* **Single Row Functions**
* Functions that work on only one value at a time are called SCALAR Functions.
* A single row function returns one result for every row of table / view.

Example: LENGTH function calculates length of one string value.

* Single row functions can be grouped together depending on the data type of their arguments and return values. **Ex:** LENGTH function is related to string data type.

**Functions can be classified according to different data types are:**

* + 1. Numeric Functions (For Number data type)
    2. Character / String Functions (For String data type)
    3. Date Function (For Date data type)
    4. Conversion Functions (For Conversion of one data type to another)
* **Aggregate functions**
* Functions that act on a set of values are called Aggregate / Group Functions.

Example: SUM function calculates total of a column.

* Group function returns a single result row for a group of rows.
* Aggregate functions can appear in SELECT lists and in ORDER BY and HAVING

clauses. They are commonly used with the GROUP BY clause in a SELECT statement.

* Aggregate functions perform a calculation on a set of values and return a single, or summary, value.

|  |  |
| --- | --- |
| **Function** | **Description** |
| **AVG** | Returns the average value of the given column or expression  **Ex:** select avg (salary) as ‘average salary’ from emp; |
| **COUNT** | Number of rows where the value of the column is not NULL  **Ex:** select count (dno) as ‘number of deptwise ’ from emp; |
| **COUNT(\*)** | Number of rows returned including duplicates and NULLs  **Ex:** select count (\*) from employees where eno = 5; |
| **MAX** | Return Maximum value of the given column or expression. MAX can be used with numeric, character and date time columns. With character columns, MAX finds the highest value in the collating sequence. MAX ignores any null values.  **Ex:** select max(salary) from emp; or select max(ename) from emp; |
| **MIN** | Return Minimum value of the given column or expression. MIN can be used with numeric, character and date time columns. With character columns, MIN finds the value that is lowest in the sort sequence. MIN ignores any null values.  **Ex:** select min (salary) from emp; or select max(ename) from emp; |
| **SUM** | Returns the sum of all the values given column or expression. SUM can be used with numeric columns only.  **Ex:** select sum (salary) as ‘total salary’ from emp; |

* **Numeric Functions (For Number data type)**
  + **ABS:** Absolute value is the measure of the magnitude of value. Absolute value is always a positive number.

**Syntax:** abs (*value*)

**Ex:** select abs(5), abs(-5), abs(0), abs(null) from dual;

ABS(5) ABS(-5) ABS(0) ABS(NULL)

---------- ---------- ---------- - ------------

5 -5 0

* + **SIGN:** Sign gives the sign of a value.

**Syntax:** sign (*value*)

**Ex:** select sign(5), sign(-5), sign(0), sign(null) from dual;

SIGN(5) SIGN(-5) SIGN(0) SIGN(NULL)

---------- ---------- ---------- --------------

1 -1 0

* + **SQRT:** This will give the square root of the given value.

**Syntax:** sqrt (value) -- here value must be positive.

**Ex:** select sqrt(4), sqrt(0), sqrt(null), sqrt(1) from dual;

SQRT(4) SQRT(0) SQRT(NULL) SQRT(1)

---------- ---------- --------------- ----------

2 0 1

* + **MOD:** Display remainder value after value devides divisor.

**Syntax:** mod(value, divisor)

**Ex:** select mod(7,4), mod(1,5), mod(null,null), mod(0,0), mod(-7,4) from dual;

MOD(7,4) MOD(1,5) MOD(NULL,NULL) MOD(0,0) MOD(-7,4)

------- -------- -------------- -------- ----------

3 1 0 -3

* + **NVL:** This will substitute the specified value in the place of null values.

**Syntax:** nvl (*null\_col, replacement\_value*)

Ex: select \* from student; -- here for 3rd row marks value is null

NO NAME MARKS

--- ------- ---------

1 a 100

2 b 200

3 c

select no, name, nvl(marks,300) from student;

NO NAME NVL(MARKS,300)

--- ------- ---------------------

1 a 100

2 b 200

3 c 300

select nvl(1,2), nvl(2,3), nvl(4,3), nvl(5,4) from dual;

NVL(1,2) NVL(2,3) NVL(4,3) NVL(5,4)

---------- ---------- ---------- ---------

1 2 4 5

select nvl(0,0), nvl(1,1), nvl(null,null), nvl(4,4) from dual;

NVL(0,0) NVL(1,1) NVL(null,null) NVL(4,4)

---------- ---------- ----------------- ----------

0 1 4

* + **POWER:** Display m power nth value.

**Syntax:** power(m,n)

**Ex:** select power(2,5), power(0,0), power(null,null), power(2,-5) from dual;

POWER(2,5) POWER(0,0) POWER(NULL,NULL) POWER(2,-5)

----------- ----------- ----------------- ---------------

32 1 .03125

* + **EXP:** This will raise e value to the give power.

**Syntax:** exp (*value*)

**Ex:** select exp(1), exp(2), exp(0), exp(null), exp(-2) from dual;

EXP(1) EXP(2) EXP(0) EXP(NULL) EXP(-2)

-------- --------- ------- ---------- ----------

2.71828183 7.3890561 1 .135335283

* + **CEIL:** Display lowest integer value which is greater than or equal to given value.

**Syntax:** ceil (*value*)

**Ex:** select ceil(5), ceil(5.1), ceil(-5), ceil( -5.1), ceil(0), ceil(null) from dual;

CEIL(5) CEIL(5.1) CEIL(-5) CEIL(-5.1) CEIL(0) CEIL(NULL)

--------- -------- -------- --------- ------- ----------

5 6 -5 -5 0

* + **FLOOR:** Display highest integer value which is lessthan or equal to given value.

**Syntax:** floor (*value*)

**Ex:** select floor(5), floor(5.1), floor(-5), floor(-5.1), floor(0), floor(null) from dual;

FLOOR(5) FLOOR(5.1) FLOOR(-5) FLOOR(-5.1) FLOOR(0) FLOOR(NULL)

--------- ---------- ---------- ---------- --------- ------------

5 5 -5 -6 0

* + **ROUND:** Display value *“m”* which is rounded to the *“n”* number of decimal places

**Syntax:** round (m,n)

**Ex:** select round(123.2345), round(123.2345,2), round(123.2354,2) from dual;

ROUND(123.2345) ROUND(123.2345,0) ROUND(123.2345,2) ROUND(123.2345,-2)

-------------- ---------------- ---------------- ----------------

123 123 123.23 100

* + **TRUNC:** Display value **m** which is truncated to the **n** number of decimal places.

**Syntax:** trunc(m,n)

**Ex:** select trunc(123.2345), trunc(123.2345,2), trunc(123.2354,2) from dual;

TRUNC(123.2345) TRUNC(123.2345,2) TRUNC(123.2345,-1) TRUNC(123.2345,-2)

-------------- ---------------- ----------------- --------------

123 123.23 120 100

* + **GREATEST:** Display maximum value from the given values or expressions.

**Syntax:** greatest (*value1, value2, value3 … valuen*)

Ex: select greatest (1, 2, 3), greatest (-1, -2, -3) from dual;

GREATEST (1,2,3) GREATEST(-1,-2,-3)

-------------------- -----------------------

3 -1

Ø If all the values are zeros then it will display zero.

Ø If any of one or all the parameters are nulls then it will display nothing.

* + **LEAST:** Display minimum value from the given values or expression results.

**Syntax:** least (*value1, value2, value3 … valuen*)

Ex: select least(1, 2, 3), least(-1, -2, -3) from dual;

LEAST(1,2,3) LEAST(-1,-2,-3)

-------------------- -----------------------

1 -3

Ø If all the values are zeros then it will display zero.

Ø If any of one or all the parameters are nulls then it will display nothing.

* **String Functions**
  + **INITCAP:** Display the given string or column values with begining char as capital.

**Syntax:** initcap (*string*)

**Ex:** select initcap('computer') from dual;

INITCAP

-----------

Computer

* + **UPPER:** Display given string chars or column values in upper case.

**Syntax:** upper (*string*)

**Ex:** select upper('computer') from dual;

UPPER

-----------

COMPUTER

* + **LOWER:** Display the given string chars or column values in lower case.

**Syntax:** lower (*string*)

**Ex:** select lower('COMPUTER') from dual;

LOWER

-----------

computer

* + **LENGTH:** Display number of chars from the given string or column values

**Syntax:** length (*string*)

Ex: select length('computer') from dual; or select ename, length(ename) from emp;

LENGTH

---------

8

* + **RPAD:** Display given string along with specific char in the right of string.

**Syntax:** rpad (*string, length [, padding\_char]*)

**Ex:** select rpad('computer',15,'\*'), rpad('computer',15,'\*#') from dual;

RPAD('COMPUTER' RPAD('COMPUTER'

---------------- ------------------

computer\*\*\*\*\*\*\* computer\*#\*#\*#\*

-- Default padding character was blank space.

* + **LPAD:** Display given string along with the specific char in the left of the string.

**Syntax:** lpad (*string, length [, padding\_char]*)

**Ex:** select lpad('computer',15,'\*'), lpad('computer',15,'\*#') from dual;

LPAD('COMPUTER' LPAD('COMPUTER'

------------------ ------------------

\*\*\*\*\*\*\*computer \*#\*#\*#\*computer

-- Default padding character was blank space.

* + **LTRIM:** Display given string by removing blank spaces from the left of string only.

**Syntax:** ltrim (*string [,unwanted\_chars]*)

**Ex:** select ltrim('computer','co'), ltrim('computer','com') from dual;

LTRIM( LTRIM

-------- ---------

mputer puter

select ltrim('computer','puter'), ltrim('computer','omputer') from dual;

LTRIM('C LTRIM('C

---------- ----------

computer computer

-- If you haven’t specify any unwanted characters it will display entire string.

* + **RTRIM:** Display given string by removing blank spaces from right of string only.

**Syntax:** rtrim (*string [, unwanted\_chars]*)

**Ex:** select rtrim('computer','er'), rtrim('computer','ter') from dual;

RTRIM( RTRIM

-------- ---------

comput compu

select rtrim('computer','comput’), rtrim('computer','compute') from dual;

RTRIM('C RTRIM('C

---------- ----------

computer computer

-- If you haven’t specify any unwanted characters it will display entire string.

* + **TRIM:** Display given string by eleminating blank spaces before and after the string.

**Syntax:** trim (*unwanted\_chars* from *string*)

**Ex:** select trim( 'i' from 'indiani') from dual;

TRIM(

-----

ndian

select trim(' welcome to ') " trim" || initcap('oracle') from dual;

welcome to Oracle

* + **TRANSLATE:** display given string character by translating source character with corresponding target character.

**Syntax:** translate (*string, old\_chars, new\_chars*)

**Ex:** select translate('india','in','xy') from dual;

TRANS

--------

xydxa

* + **REPLACE:** Display given string by replacing source string with target string.

**Syntax:** replace (*string, old\_chars [, new\_chars]*)

**Ex:** select replace('india','in','xy'), replace(‘india’,’in’) from dual;

REPLACE REPLACE

------- -------

Xydia dia

* + **CONCAT:** This will be used to combine two strings only.

**Syntax:** concat (*string1, string2*)

**Ex:** select concat('computer',' operator') from dual;

CONCAT('COMPUTER'

-------------------------

computer operator

To combine more than two strings using concatenation operator (||).

select 'how' || ' are' || ' you' from dual;

'HOW'||'ARE

---------------

how are you

* + **ASCII:** This will return the decimal representation in the database character set of the first character of the string.

**Syntax:** ascii (*string*)

**Ex:** select ascii('a'), ascii('apple') from dual;

ASCII('A') ASCII('APPLE')

-------- -----------

97 97

* + **CHR:** This will return the character having the binary equivalent to the string in either

the database character set or the national character set.

**Syntax:** chr (*number*)

**Ex:** select chr(97) from dual;

CHR

-----

a

* + **SUBSTR:** This will be used to extract substrings.

**Syntax:** substr (string, start\_chr\_no [no\_of\_chars])

**Ex:** select substr('computer',2), substr('computer',2,5), substr('computer',3,7) from dual;

SUBSTR( SUBSTR SUBSTR

---------- ------- --------

omputer omput mputer

Ø If *no\_of\_chars* parameter is negative then it will display nothing.

Ø If both parameters except *string* are null or zeros then it will display nothing.

Ø If *no\_of\_chars* parameter is greater than the length of the string then it ignores and calculates based on the original string length.

Ø If *start\_chr\_count* is negative then it will extract the substring from right end.

1 2 3 4 5 6 7 8

C O M P U T E R

-8 -7 -6 -5 -4 -3 -2 -1

* + **INSTR:** Display the position of char in the given string or column.

Syntax: instr (*string, search\_str, [start\_chr\_count, [occurrence] ]*)

Ex: select instr('information','o',4,1), instr('information','o',4,2) from dual;

INSTR('INFORMATION','O',4,1) INSTR('INFORMATION','O',4,2)

-------------------------- ----------------------------

4 10

Ø If you are not specifying *start\_chr\_count* and *occurrence* then it will start search from the beginning and finds first occurrence only.

Ø If both parameters *start\_chr\_count* and *occurrence* are null, it will display nothing.

* **DATE FUNCTIONS**

Oracle default date format is DD-MON-YY. We can change the default format to our desired format by using following command.

alter session set nls\_date\_format = ‘DD-MONTH-YYYY’;

* + **SYSDATE:** This will give the current date and time.

Ex: select sysdate from dual;

SYSDATE

-----------

24-DEC-06

* + **CURRENT\_DATE:** This will returns the current date in the session’s timezone.

Ex: select current\_date from dual;

CURRENT\_DATE

------------------

24-DEC-06

* + **CURRENT\_TIMESTAMP:** This will return the current timestamp with the active time zone information with system date, including fractional seconds.

Ex: select current\_timestamp from dual; or select systimestamp from dual;

select localtimestamp from dual; // returns local timestamp

CURRENT\_TIMESTAMP

-------------------------------------------------------------

24-DEC-06 03.42.41.383369 AM +05:30

* + **DBTIMEZONE:** This will returns the current database time zone in UTC format. (Coordinated Universal Time)

Ex: select dbtimezone from dual;

DBTIMEZONE

---------------

-07:00

* + **SESSIONTIMEZONE:** This will returns the value of the current session’s time zone.

Ex: select sessiontimezone from dual;

SESSIONTIMEZONE

----------------------------------------------------

+05:30

* + **TO\_CHAR:** This will be used to extract various date formats.

**Syntax:** to\_char (*date*, *format*)

Ex: select to\_char(sysdate,'dd month yyyy hh:mi:ss am dy') from dual;

TO\_CHAR(SYSDATE,'DD MONTH YYYYHH:MI

----------------------------------------------------

24 december 2006 02:03:23 pm sun

select to\_char(sysdate,'dd month year') from dual;

TO\_CHAR(SYSDATE,'DDMONTHYEAR')

-------------------------------------------------------

24 december two thousand six

* + **TO\_DATE:** It will display any non-Oracle date format value in oracle's date format.

Ex: select to\_date ('24/dec/2006', 'dd \* month \* day') from dual;

TO\_DATE('24/DEC/20

--------------------------

24 \* december \* Sunday

-- If you are not using to\_char oracle will display output in default date format.

* + **ADD\_MONTHS:** Display a date value after adding “n” number of months to the specified date.

**Syntax:** add\_months (*date, no\_of\_months*)

**Ex:**select add\_months('11-jan-90',5)from dual; **or** select add\_months(sysdate,5)from emp;

ADD\_MONTHS

----------------

11-JUN-90

select add\_months ('11-jan-90', -5) from dual;

ADD\_MONTH

--------------------- Ø If no\_of\_months is zero then it will display the same date.

11-AUG-89 Ø If no\_of\_months is null then it will display nothing.

* + **MONTHS\_BETWEEN:** it shows number of months between dates.

**Syntax:** months\_between (*date1, date2*)

**Ex:** select months\_between ('11-aug-1990','11-jan-1990') from dual;

MONTHS\_BETWEEN ('11-AUG-1990','11-JAN-1990')

--------------------------------------------------------------

7

select months\_between(sysdate,'21-may-13') from dual;;

* + **NEXT\_DAY:** display the date value of given weekdayname after the specified date.

**Syntax:** next\_day (*date, day*)

Ex: select next\_day('24-dec-2006','sun') from dual;

NEXT\_DAY(

-------------

31-DEC-06 -- If the day parameter is null then it will display nothing.

* + **LAST\_DAY:** Display the date value of last day in the month.

**Syntax:** last\_day(*date*)

Ex: select last\_day('24-dec-2006') from dual;

LAST\_DAY(

-------------

31-DEC-06

* + **EXTRACT:** This is used to extract a portion of the date value.

**Syntax:** extract ((year | month | day | hour | minute | second), *date*)

Ex: select extract (year from sysdate) from dual;

EXTRACT(YEARFROMSYSDATE)

------------------------------------

2006 -- You can extract only one value at a time.

* **CONVERSION FUNCTIONS**
  + **TO\_CHAR (Date conversion):** It accepts Oracle's date type data and convert it into required char format.

**Syntax** to\_char(date\_value, [’format’ ] )

**Example** select to\_char(’24-jun-14’, ‘month dd, yyyy’) “new date format” from dual;

New date format

--------------------

June 24, 2014

* + **TO\_CHAR (Number conversion):** it accepts number type data and convert it into character type data.

**Syntax** to\_char(n,[’format’ ])

**Example** select to\_char (12345, ‘$099,999’) “to char” from dual;

To char

--------------

$012,345

* + **TO\_DATE:** It converts character data type into date data type. The format is a date format specifying the format of char\_value.

**Syntax** to\_date(char\_value,[’format’])

**Example** select to\_date (’05/sep/14’,’dd/mon/yy’) “to date” from dual;

To date

-------------

05-SEP-14

* + **TO\_NUMBER:** It accepts the chardata which contains a sequence of digits and convert it into number type data.

**Syntax** to\_number(char\_value, [’format’] )

**Example** select to\_number(’1234.56’, ’9999.9’) “to number” from dual;

To number

-------------

1234.5