Show all tables cpmmand

select \* from user\_tables;

// to see tables in sql

select TABLE\_NAME from user\_tables ORDER BY TABLE\_NAME;

// to open editor in sql

SQL> SET EDITFILE c:/demo/sqlEditor.txt;

SQL> ed

SQL> SET SERVEROUTPUT ON

To view sql in good format

set lines 256

set trimout on

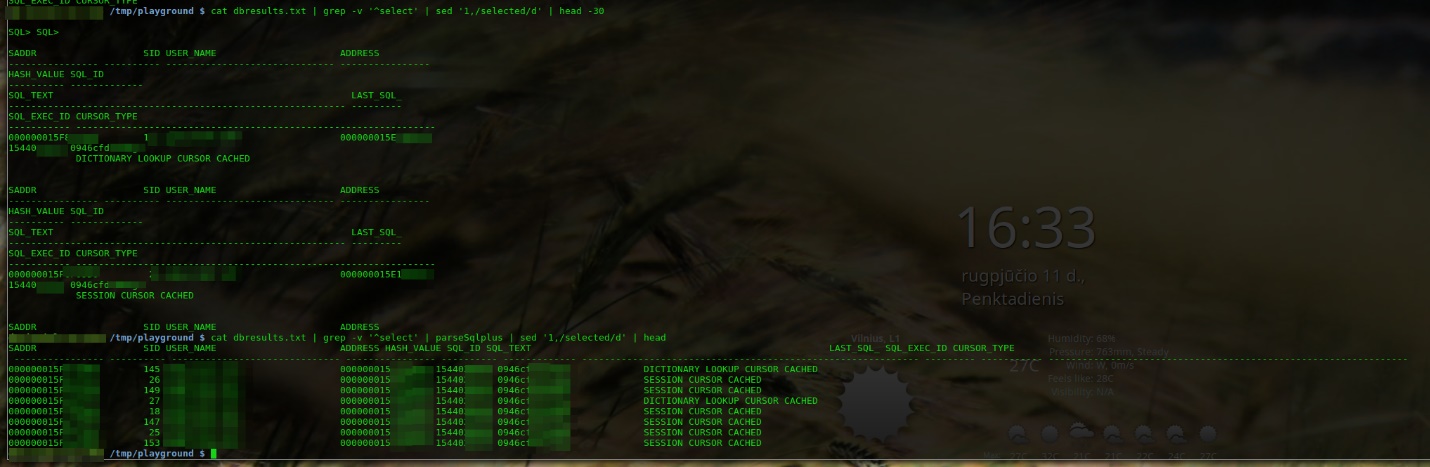
set tab off

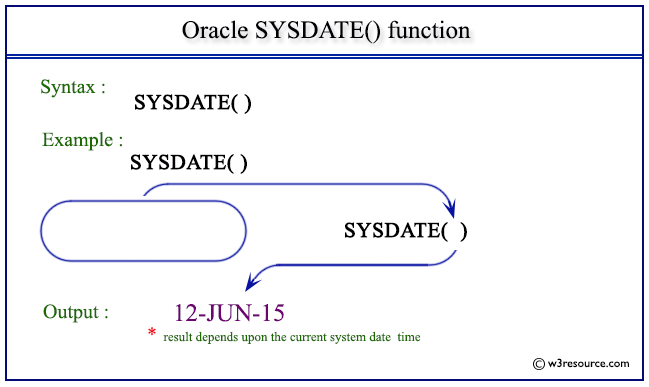
set pagesize 100

set colsep " | "

colsep is optional, but it makes output look like sqlite which is easier to parse using scripts.

EDIT: A little preview of parsed and non-parsed output

[](https://i.stack.imgur.com/P6gs2.png)



# TO\_CHAR - Convert Datetime to String - Oracle to SQL Server Migration

In Oracle, TO\_CHAR function converts a datetime value (DATE, TIMESTAMP data types i.e.) to a string using the specified format.

In SQL Server, you can use CONVERT or CAST functions to convert a datetime value (DATETIME, DATETIME2 data types i.e.) to a string.

**Oracle**:

-- Convert the current date to YYYY-MM-DD format

SELECT TO\_CHAR(SYSDATE, 'YYYY-MM-DD') FROM dual;

*# 2012-07-19*

**SQL Server**:

-- Convert the current date to YYYY-MM-DD format

SELECT CONVERT(VARCHAR(10), GETDATE(), 120);

*# 2012-07-19*

## TO\_CHAR for Datetime Conversion Overview

Summary information:

|  |  |  |
| --- | --- | --- |
|  | **Oracle** | **SQL Server** |
| **Syntax** | TO\_CHAR(datetime, format) | CONVERT(VARCHAR(n), datetime, style) |
| CAST(datetime as VARCHAR(n)) |
| **Default Format and Style** | Specified by NLS\_DATE\_FORMAT | Mon DD YYYY HH12:MI |

**Last Update**: Oracle 11g R2 and Microsoft SQL Server 2012

## TO\_CHAR Format Specifiers

Oracle TO\_CHAR supports the following format specifiers for datetime values:

|  |  |
| --- | --- |
| **Oracle TO\_CHAR** | **Format Specifier** |
| YYYY | 4-digit year |
| YY | 2-digit year |
| MON | Abbreviated month (Jan - Dec) |
| MONTH | Month name (January - December) |
| MM | Month (1 - 12) |
| DY | Abbreviated day (Sun - Sat) |
| DD | Day (1 - 31) |
| HH24 | Hour (0 - 23) |
| HH or HH12 | Hour (1 - 12) |
| MI | Minutes (0 - 59) |
| SS | Seconds (0 - 59) |

## Converting Oracle TO\_CHAR to SQL Server

Unlike Oracle TO\_CHAR function that allows you to build any format string using format specifiers (YYYY and MM i.e.), in SQL Server, you have to use a datetime **style** that defines the format for **the entire** datetime string.

## Default Format

In Oracle, the default format of a datetime string depends on the NLS\_DATE\_FORMAT session variable:

**Oracle**:

-- Convert to string with the default format

SELECT TO\_CHAR(SYSDATE) FROM dual;

*# 20-JUL-12*

-- Change the default format

ALTER SESSION SET NLS\_DATE\_FORMAT = 'Mon DD, YYYY';

-- Convert to string with the default format

SELECT TO\_CHAR(SYSDATE) FROM dual;

*# Jul 20, 2012*

General Functions

General functions are used to handle NULL values in database. The objective of the general NULL handling functions is to replace the NULL values with an alternate value. We shall briefly see through these functions below.

NVL

The NVL function substitutes an alternate value for a NULL value.

**SYNTAX:**

NVL( Arg1, replace\_with )

In the syntax, both the parameters are mandatory. Note that NVL function works with all types of data types. And also that the data type of original string and the replacement must be in compatible state i.e. either same or implicitly convertible by Oracle.

If arg1 is a character value, then oracle converts replacement string to the data type compatible with arg1 before comparing them and returns VARCHAR2 in the character set of expr1. If arg1 is numeric, then Oracle determines the argument with highest numeric precedence, implicitly converts the other argument to that data type, and returns that data type.

The SELECT statement below will display 'n/a' if an employee has been not assigned to any job yet i.e. JOB\_ID is NULL. Otherwise, it would display the actual JOB\_ID value.

SELECT first\_name, NVL(JOB\_ID, 'n/a')

FROM employees;

NVL2

As an enhancement over NVL, Oracle introduced a function to substitute value not only for NULL columns values but also for NOT NULL columns. NVL2 function can be used to substitute an alternate value for NULL as well as non NULL value.

**SYNTAX:**

NVL2( string1, value\_if\_NOT\_null, value\_if\_null )

The SELECT statement below would display 'Bench' if the JOB\_CODE for an employee is NULL. For a definite not null value of JOB CODE, it would show constant value 'Job Assigned'.

SQL> SELECT NVL2(JOB\_CODE, 'Job Assigned', 'Bench')

FROM employees;

NULLIF

The NULLIF function compares two arguments expr1 and expr2. If expr1 and expr2 are equal, it returns NULL; else, it returns expr1. Unlike the other null handling function, first argument can't be NULL.

**SYNTAX:**

NULLIF (expr1, expr2)

Note that first argument can be an expression that evaluates to NULL, but it can't be the literal NULL. Both the parameters are mandatory for the function to execute.

The below query returns NULL since both the input values, 12 are equal.

SELECT NULLIF (12, 12)

FROM DUAL;

Similarly, below query return 'SUN' since both the strings are not equal.

SELECT NULLIF ('SUN', 'MOON')

FROM DUAL;

# **NVL**

**Syntax**

[Description of the illustration nvl.gif](https://docs.oracle.com/cd/B19306_01/server.102/b14200/img_text/nvl.htm)

**Purpose**

NVL lets you replace null (returned as a blank) with a string in the results of a query. If *expr1* is null, then NVL returns *expr2*. If *expr1* is not null, then NVL returns *expr1*.

The arguments *expr1* and *expr2* can have any datatype. If their datatypes are different, then Oracle Database implicitly converts one to the other. If they are cannot be converted implicitly, the database returns an error. The implicit conversion is implemented as follows:

* If *expr1* is character data, then Oracle Database converts *expr2* to the datatype of *expr1* before comparing them and returns VARCHAR2 in the character set of *expr1*.
* If *expr1* is numeric, then Oracle determines which argument has the highest numeric precedence, implicitly converts the other argument to that datatype, and returns that datatype.

**See Also:**

[Table 2-10, "Implicit Type Conversion Matrix"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements002.htm#g195937) for more information on implicit conversion and ["Numeric Precedence"](https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements001.htm#i156865) for information on numeric precedence

**Examples**

The following example returns a list of employee names and commissions, substituting "Not Applicable" if the employee receives no commission:

SELECT last\_name, NVL(TO\_CHAR(commission\_pct), 'Not Applicable')

"COMMISSION" FROM employees

WHERE last\_name LIKE 'B%'

ORDER BY last\_name;

LAST\_NAME COMMISSION

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

Baer Not Applicable

Baida Not Applicable

Banda .1

Bates .15

Bell Not Applicable

Bernstein .25

Bissot Not Applicable

Bloom .2

Bull Not Applicable

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |

# Aggregate Functions

Aggregate Functions are all about

* Performing  calculations on multiple rows
* Of a single column of a table
* And returning a single value.

The ISO standard defines five (5) aggregate functions namely;

1) COUNT  
2) SUM  
3) AVG  
4) MIN  
5) MAX

SELECT COUNT(CustomerID), Country

FROM Customers

### Result:

Number of Records: 1

|  |  |
| --- | --- |
| **COUNT(CustomerID)** | **Country** |
| 91 | Germany |

# Group by

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

## Syntax

The basic syntax of a GROUP BY clause is shown in the following code block. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2

FROM table\_name

WHERE [ conditions ]

GROUP BY column1, column2

ORDER BY column1, column2

## Example

Consider the CUSTOMERS table is having the following records −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to know the total amount of the salary on each customer, then the GROUP BY query would be as follows.

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result −

+----------+-------------+

| NAME | SUM(SALARY) |

+----------+-------------+

| Chaitali | 6500.00 |

| Hardik | 8500.00 |

| kaushik | 2000.00 |

| Khilan | 1500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 2000.00 |

+----------+-------------+

Now, let us look at a table where the CUSTOMERS table has the following records with duplicate names −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Ramesh | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | kaushik | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now again, if you want to know the total amount of salary on each customer, then the GROUP BY query would be as follows −

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result −

+---------+-------------+

| NAME | SUM(SALARY) |

+---------+-------------+

| Hardik | 8500.00 |

| kaushik | 8500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 3500.00 |

+---------+-------------+

## The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

### HAVING Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;

### Example

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;

### Example

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE LastName = 'Davolio' OR LastName = 'Fuller'  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 25;

# ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### ORDER BY Syntax

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

## ORDER BY Example

SELECT \* FROM Customers

ORDER BY Country;

umber of Records: 91

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 |  |

## ORDER BY DESC Example

SELECT \* FROM Customers

ORDER BY Country DESC;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 33 | GROSELLA-Restaurante | Manuel Pereira | 5ª Ave. Los Palos Grandes | Caracas | 1081 | Venezuela |
| 35 | HILARIÓN-Abastos | Carlos Hernández | Carrera 22 con Ave. Carlos Soublette #8-35 | San Cristóbal | 5022 | Venezuela |
| 46 | LILA-Supermercado | Carlos González | Carrera 52 con Ave. Bolívar #65-98 Llano Largo | Barquisimeto | 3508 | Venezuela |
| 47 | LINO-Delicateses | Felipe Izquierdo | Ave. 5 de Mayo Porlamar | I. de Margarita | 4980 | Venezuela |
| 32 | Great Lakes Food Market | Howard Snyder | 2732 Baker Blvd. | Eugene | 97403 | USA |
| 36 | Hungry Coyote Import Store | Yoshi Latimer | City Center Plaza 516 Main St. | Elgin | 97827 | USA |
| 43 | Lazy K Kountry Store | John Steel | 12 Orchestra Terrace | Walla Walla | 99362 | USA |

## ORDER BY Several Columns Example

## The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

SELECT \* FROM Customers

ORDER BY Country, CustomerName;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |

## ORDER BY Several Columns Example 2

he following SQL statement selects all customers from the "Customers" table, sorted ascending by the "Country" and descending by the "CustomerName" column:

SELECT \* FROM Customers

ORDER BY Country ASC, CustomerName DESC;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 64 | Rancho grande | Sergio Gutiérrez | Av. del Libertador 900 | Buenos Aires | 1010 | Argentina |
| 54 | Océano Atlántico Ltda. | Yvonne Moncada | Ing. Gustavo Moncada 8585 Piso 20-A | Buenos Aires | 1010 | Argentina |
| 12 | Cactus Comidas para llevar | Patricio Simpson | Cerrito 333 | Buenos Aires | 1010 | Argentina |
| 59 | Piccolo und mehr | Georg Pipps | Geislweg 14 | Salzburg | 5020 | Austria |
| 20 | Ernst Handel | Roland Mendel | Kirchgasse 6 | Graz | 8010 | Austria |
| 76 | Suprêmes délices | Pascale Cartrain | Boulevard Tirou, 255 | Charleroi | B-6000 | Belgium |
| 50 | Maison Dewey | Catherine Dewey | Rue Joseph-Bens 532 | Bruxelles | B-1180 | Belgium |

# [**Insert Data Into Tables Linked by Foreign Key**](https://stackoverflow.com/questions/1997998/insert-data-into-tables-linked-by-foreign-key)

Customer

==================

Customer\_ID | Name

Order

==============================

Order\_ID | Customer\_ID | Price

To insert an order, here is what I need to do usually,

For example, "John" place "1.34" priced order.

(1) Get Customer\_ID from Customer table, where name is "John"

(2) If there are no Customer\_ID returned (There is no John), insert "John"

(3) Get Customer\_ID from Customer table, where name is "John"

(4) Insert "Customer\_ID" and "1.34" into Order table.

There are 4 SQL communication with database involved for this simple operation!!!

You can do it in one sql statement for existing customers, 3 statements for new ones. All you have to do is be an optimist and act as though the customer already exists:

insert into "order" (customer\_id, price) values \

((select customer\_id from customer where name = 'John'), 12.34);

If the customer does not exist, you'll get an sql exception which text will be something like:

null value in column "customer\_id" violates not-null constraint

(providing you made customer\_id non-nullable, which I'm sure you did). When that exception occurs, insert the customer into the customer table and redo the insert into the order table:

insert into customer(name) values ('John');

insert into "order" (customer\_id, price) values \

((select customer\_id from customer where name = 'John'), 12.34);