Quick Reference for:

**Data Manipulation Language (Advance)**

Contents

[**Summary Queries and Aggregation** 1](#_Toc82935861)

[How to work with AGGREGATE FUNCTIONS 1](#_Toc82935862)

[Using DISTINCT keyword in a function 1](#_Toc82935863)

[How to GROUP and Summarize data 1](#_Toc82935864)

[How to use HAVING to filter aggregates 2](#_Toc82935865)

[Using ROLLUP and CUBE to create subtotal rows 2](#_Toc82935866)

[**Joins & Multi-table Selects** 3](#_Toc82935867)

[Types of Joins 3](#_Toc82935868)

[INNER JOIN 3](#_Toc82935869)

[TABLE ALIAS to abbreviate code 4](#_Toc82935870)

[Joining more than 2 tables 4](#_Toc82935871)

[SELF JOIN 4](#_Toc82935872)

[LEFT Join 5](#_Toc82935873)

[FULL OUTER JOIN 5](#_Toc82935874)

[UNION to stack queries on queries 5](#_Toc82935875)

[MINUS statement 6](#_Toc82935876)

[**Subqueries** 7](#_Toc82935877)

[Overview 7](#_Toc82935878)

[SUBQUERY in WHERE clause 7](#_Toc82935879)

[Subquery in WHERE clause using ALL or ANY keywords 8](#_Toc82935880)

[SUBQUERY in FROM clause 8](#_Toc82935881)

[Example of Complex query using Subqueries 9](#_Toc82935882)

[**Working with datatypes and Advanced SQL Functions** 10](#_Toc82935883)

[Formatting numbers and dates in ORACLE 10](#_Toc82935884)

[Common STRING Functions 10](#_Toc82935885)

[String Parsing and Manipulation Functions 11](#_Toc82935886)

[Converting numeric-like strings to number using TO\_NUMBER 11](#_Toc82935887)

[Common NUMERIC Functions 12](#_Toc82935888)

[Common DATE Functions 13](#_Toc82935889)

[DATE and TIME 14](#_Toc82935890)

[Handling logical output with CASE 14](#_Toc82935891)

[Handling NULL values 15](#_Toc82935892)

[Window Functions (ROW\_NUMBER, RANK, DENSE\_RANK) 16](#_Toc82935893)

[**More Resources** 16](#_Toc82935894)

# **Summary Queries and Aggregation**

## How to work with AGGREGATE FUNCTIONS

|  |  |
| --- | --- |
| **Syntax** | Select *FUNCTION(\* or column)*  FROM *table;* |
| **Example** | SELECT 'After May-1-2014' AS selection\_date,  COUNT(\*) AS count\_of\_rows\_on\_invoices,  SUM(invoice\_total) AS total\_invoice\_amt,  MIN(invoice\_total) AS lowest\_invoice\_total,  MAX(invoice\_total) AS highest\_invoice\_total,  AVG(invoice\_total) AS avg\_invoice\_amt,  ROUND(AVG(invoice\_total), 2) AS rounded\_avg\_invoice\_amt  FROM invoices  WHERE invoice\_date > '01-MAY-2014'; |
| **More info** | Count, AVG, Sum - <https://www.w3schools.com/sql/sql_count_avg_sum.asp>  Min, Max - <https://www.w3schools.com/sql/sql_min_max.asp> |

## Using DISTINCT keyword in a function

|  |  |
| --- | --- |
| **Syntax** | --To count a distinct number of values, you can use DISTINCT inside the COUNT() function  Select *FUNCTION(***DISTINCT** *column)*  FROM *table;* |
| **Example** | SELECT COUNT(DISTINCT vendor\_id) AS number\_of\_vendors,  COUNT(vendor\_id) AS number\_of\_invoices  FROM invoices; |

## How to GROUP and Summarize data

|  |  |
| --- | --- |
| **Syntax** | --Order of SELECT clauses when GROUP BY is used. It happens after WHERE but before ORDER BY  SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)ORDER BY column\_name(s);  --Important: When including non-aggregate and aggregate columns in a SELECT, you need to be sure that all the non-aggregate columns are included in the GROUP BY clause. See below |
| **Example** | -- Group by 1 column and count rows associated to each value in that column  SELECT vendor\_id,  COUNT(\*) AS invoice\_qty  FROM invoices  GROUP BY vendor\_id;  -- Group by more than 1 column while using an aggregate function  SELECT vendor\_id, invoice\_date,  COUNT(\*) AS invoice\_qty  FROM invoices  GROUP BY vendor\_id, invoice\_date  order by vendor\_id; |
| **More info** | <https://www.w3schools.com/sql/sql_groupby.asp> |

## How to use HAVING to filter aggregates

|  |  |
| --- | --- |
| **Syntax** | --HAVING filters aggregates (e.g. HAVING Count\*) > 10), so it has to come after GROUP BY  SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s); |
| **Example** | --Example that filters out COUNT (i.e. an aggregate column) greater than 1  SELECT vendor\_id,invoice\_date,  COUNT(\*) AS invoice\_qty  FROM invoices  GROUP BY vendor\_id, invoice\_date  HAVING COUNT(\*) > 1  order by vendor\_id;  --Example that included more than one aggregate filter in the HAVING  SELECT invoice\_date,  COUNT(\*) AS invoice\_qty,  SUM(invoice\_total) AS invoice\_sum  FROM invoices  GROUP BY invoice\_date  HAVING COUNT(\*) > 1  AND SUM(invoice\_total) > 100  ORDER BY invoice\_date DESC; |
| **More info** | <https://www.w3schools.com/sql/sql_having.asp> |

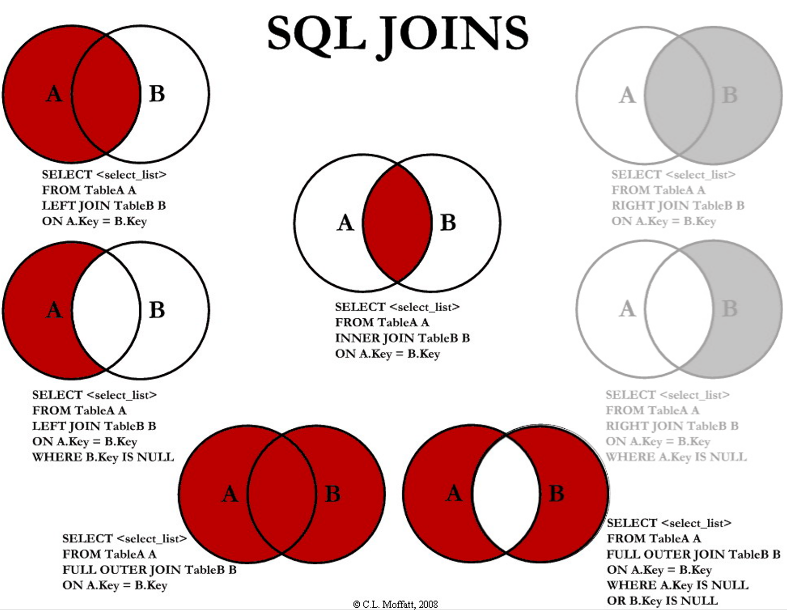
## Using ROLLUP and CUBE to create subtotal rows

|  |  |
| --- | --- |
| **Syntax** | -- Rollup can be used to add a subtotal row for the non-aggregate combination from right to left.  GROUP BYROLLUP*(columns)*  -- Cube can be used to create a subtotal row for all possible combinations on non-aggregate. Commonly used in Analytics and BI.  GROUP BYCUBE*(columns)* |
| **Example** | -- Using ROLLUP to create a final summary row  SELECT vendor\_id, COUNT(\*) AS invoice\_count,  SUM(invoice\_total) AS invoice\_total  FROM invoices  GROUP BY ROLLUP(vendor\_id);  -- using ROLLUP to create a final summary row for state/city, city, and overall total  SELECT vendor\_state, vendor\_city, COUNT(\*) AS qty\_vendors  FROM vendors  WHERE vendor\_state IN ('IA', 'NJ')  GROUP BY ROLLUP (vendor\_state, vendor\_city)  ORDER BY vendor\_state, vendor\_city;  -- using CUBE to create a final summary row for state/city, city, state, and overall total  SELECT vendor\_state, vendor\_city, COUNT(\*) AS qty\_vendors  FROM vendors  WHERE vendor\_state IN ('IA', 'NJ')  GROUP BY CUBE(vendor\_state, vendor\_city)  ORDER BY vendor\_state, vendor\_city; |
| **More info** | Rollup - <https://www.sqltutorial.org/sql-rollup/>  Cube - <https://www.oracletutorial.com/oracle-basics/oracle-cube/> |

# **Joins & Multi-table Selects**

## Types of Joins

* **Inner join –** Used to find only rows that match between Table A and Table B
* **Outer Joins:** 
  + **Left** – Used to pull all rows on Table A and rows that match on Table B. See 2nd left join option below that uses IS NULL in WHERE clause to only pull Table A records that don’t exist on Table B.
  + **Full Outer** – Used to pull all rows on Table A and B regardless if they match or not. See 2nd full join option below that use IS NULL in WHERE clause to only pull Table A and B records that don’t match.
  + **What about Right?** – Right join isn’t needed since you can just switch the order of the tables in the FROM clause and use a left join. This makes complex joins of more than 2 tables easy to read.
* **Self Join** – A self join is a regular join, but the table is joined with itself.
* **More info** - <https://www.w3schools.com/sql/sql_join.asp>
* **Help visual for remembering**



## INNER JOIN

|  |  |
| --- | --- |
| **Syntax** | Use the INNER JOIN keyword to select records that have matching values in both tables. You can also just use the keyword JOIN which will perform the same as an INNER JOIN.  SELECT column\_name(s) FROM table1 INNER JOIN table2ON table1.column\_name = table2.column\_name; |
| **Example** | SELECT invoice\_number, vendor\_name  FROM vendors INNER JOIN invoices  ON vendors.vendor\_id = invoices.vendor\_id  ORDER BY invoice\_number; |
| **More info** | <https://www.w3schools.com/sql/sql_join_inner.asp> |

## TABLE ALIAS to abbreviate code

|  |  |
| --- | --- |
| **Syntax** | --You can use table alias to shorten the SQL statement.  FROM table *alias* |
| **Example** | SELECT invoice\_number, vendor\_name  FROM vendors v INNER JOIN invoices i  ON v.vendor\_id = i.vendor\_id  ORDER BY invoice\_number; |
| **More info** | <https://www.w3schools.com/sql/sql_alias.asp> |

## Joining more than 2 tables

|  |  |
| --- | --- |
| **Syntax** | SELECT column\_name(s) FROM table1 JOIN table2 ON table1.column\_name = table2.column\_name  JOIN *table3* ON table3.column\_name  JOIN ...; |
| **Example** | SELECT vendor\_name, invoice\_number, invoice\_date,  line\_item\_amt, account\_description  FROM vendors v  JOIN invoices i  ON v.vendor\_id = i.vendor\_id  JOIN invoice\_line\_items li  ON i.invoice\_id = li.invoice\_id  JOIN general\_ledger\_accounts gl  ON li.account\_number = gl.account\_number  WHERE (invoice\_total - payment\_total - credit\_total) > 0  ORDER BY vendor\_name, line\_item\_amt DESC; |

## SELF JOIN

|  |  |
| --- | --- |
| **Syntax** | --These are useful in cases where you want join a column to the same column or another column that stores a similar value.  SELECT column\_name(s) FROM table1 T1 join table1 T2  ON (column = column); |
| **Example** | select e2.employee\_id, e2.first\_name, e2.last\_name,  e1.first\_name || ' ' || e1.last\_name as manager  from employees e1 inner join employees e2  on e1.employee\_id = e2.manager\_id; |
| **More info** | <https://www.w3schools.com/sql/sql_join_self.asp> |

## LEFT Join

|  |  |
| --- | --- |
| **Syntax** | SELECT column\_name(s) FROM table1 LEFT JOIN table2ON table1.column\_name = table2.column\_name; |
| **Example** | SELECT vendor\_name, invoice\_number, invoice\_total  FROM vendors LEFT JOIN invoices  ON vendors.vendor\_id = invoices.vendor\_id  ORDER BY vendor\_name; |
| **More info** | https://www.w3schools.com/sql/sql\_join\_left.asp |

## FULL OUTER JOIN

|  |  |
| --- | --- |
| **Syntax** | SELECT column\_name(s) FROM table1 FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition; |
| **Example** | SELECT department\_name AS dept\_name,  d.department\_number AS d\_dept\_no,  e.department\_number AS e\_dept\_no,  last\_name  FROM departments d  FULL JOIN employees e  ON d.department\_number =  e.department\_number  ORDER BY department\_name; |
| **More info** | <https://www.w3schools.com/sql/sql_join_full.asp> |

## UNION to stack queries on queries

|  |  |
| --- | --- |
| **Syntax** | SELECT column\_name(s) FROM table1 UNION SELECT column\_name(s) FROM table2; |
| **Example** | **--UNION FROM THE SAME TABLE**  SELECT 'Active' AS source, invoice\_number, invoice\_date, invoice\_total  FROM invoices  WHERE (invoice\_total - payment\_total - credit\_total) > 0  UNION  SELECT 'Paid' AS source, invoice\_number, invoice\_date, invoice\_total  FROM invoices  where (invoice\_total - payment\_total - credit\_total) <= 0  ORDER BY invoice\_total DESC; |
| **More info** | <https://www.w3schools.com/sql/sql_union.asp> |

## MINUS statement

|  |  |
| --- | --- |
| **Syntax** | --The MINUS keyword is used to return the results of the first query with matching results from query removed. Essentially is does a DIFF between the two queries and removes duplicate rows. NOTE: This can only be used when column number and datatypes match between both queries  Query 1  MINUS  Query 2; |
| **Example** | select vendor\_id  from vendors  minus  select distinct vendor\_id  from invoices; |
| **More info** | <https://www.sqltutorial.org/sql-minus/>  **TIP:** The MINUS operator is not supported in all SQL databases. It can used in databases such as Oracle. For databases such as SQL Server, PostgreSQL, and SQLite, use the [EXCEPT operator](https://www.techonthenet.com/sql/except.php) to perform this type of query. |

# **Subqueries**

## Overview

Subqueries (SQL statements within other SQL statements) can be placed in many parts of your query including the FROM, WHERE, and SELECT clauses. For the purposes of this reference we’ll only talk about subqueries in the WHERE and subqueries in the FROM (also known as an in-line view). **TIP** -All subqueries (i.e. the inner query) will materialize first where then the result(s) of the subquery will then be passed to the outer query.

***Subqueries in the WHERE***

* If your subquery returns a single value, you can use it the WHERE clause with standard arithmetic operators like >, <, >=, <=, and = like the following example: **WHERE column > (subquery).**
* If your query returns a column of values, the standard arithmetic operators. You can though use the ALL or ANY keywords between the operator and subquery like this example: **WHERE column > ALL (subquery)** or **WHERE column > ANY (subquery)**. Adding the ALL keyword would return records that are higher than all values or the MAX value in the list. Adding the ANY keyword would return records that are higher than the MIN value in the list.
* You can also use the IN operator to filter rows that equal any of the value of a list like this example:

**WHERE column IN (subquery)**

***Subqueries in the FROM*** – When your subquery returns more than one column, it can’t be evaluated in the WHERE but you can actually treat it like a materialized view or virtual table in the FROM. You can also join existing tables to a query just like so:

|  |  |  |
| --- | --- | --- |
| **Select \***  **FROM Table\_A a inner join (subquery) s**  **on a.column\_name = s.column\_name;** | **Select \***  **FROM (subquery);** |  |

## SUBQUERY in WHERE clause

|  |  |
| --- | --- |
| **Example** | --Subquery with a single value in WHERE  SELECT invoice\_number, invoice\_date, invoice\_total  FROM invoices  WHERE invoice\_total > (SELECT AVG(invoice\_total)  FROM invoices)  ORDER BY invoice\_total;  --Subquery with a list of value in WHERE using IN clause  SELECT invoice\_number, invoice\_date, invoice\_total  FROM invoices  WHERE vendor\_id IN  (SELECT vendor\_id  FROM vendors  WHERE vendor\_state = 'CA')  ORDER BY invoice\_date;  --NOTE: Here’s another version of the 2nd query above that returns the same results using a join.  SELECT invoice\_number, invoice\_date, invoice\_total  FROM invoices JOIN vendors  ON invoices.vendor\_id = vendors.vendor\_id  WHERE vendor\_state = 'CA'  ORDER BY invoice\_date; |

## Subquery in WHERE clause using ALL or ANY keywords

|  |  |  |
| --- | --- | --- |
| **Syntax** | SELECT column\_name(s) FROM table\_name WHERE column\_name operator ALL   (SELECT column\_name  FROM table\_name  WHERE condition); | SELECT column\_name(s) FROM table\_name WHERE column\_name operator ANY   (SELECT column\_name  FROM table\_name  WHERE condition); |
| **Example** | --Using ALL keyword to invoice\_totals greater than vendor 34’s MAX invoice\_total.  SELECT vendor\_name, invoice\_number, invoice\_total  FROM invoices i JOIN vendors v ON i.vendor\_id = v.vendor\_id  WHERE invoice\_total > ALL  (SELECT invoice\_total  FROM invoices  WHERE vendor\_id = 34)  ORDER BY vendor\_name;  --Using ALL keyword to invoice\_totals greater than vendor 34’s MIN invoice\_total.  SELECT vendor\_name, invoice\_number, invoice\_total  FROM invoices i JOIN vendors v ON i.vendor\_id = v.vendor\_id  WHERE invoice\_total > ANY  (SELECT invoice\_total  FROM invoices  WHERE vendor\_id = 34)  ORDER BY vendor\_name; | |
| **More info** | <https://www.w3schools.com/sql/sql_any_all.asp> | |

## SUBQUERY in FROM clause

|  |  |  |
| --- | --- | --- |
| **Syntax** | --Select from a query  SELECT COLUMNS  FROM (SUBQUERY); | --Select from a table that is joined to a query  SELECT COLUMNS  FROM TABLE t JOIN (SUBQUERY) s  ON t.column\_name = s.column\_name; |
| **Example** | --Select from a query that has an alias of “subquery”  SELECT \*  FROM (SELECT vendor\_id,  max(invoice\_date) as last\_invoice\_date,  AVG(invoice\_total) AS average\_invoice\_total  FROM invoices  HAVING AVG(invoice\_total) > 4900  GROUP BY vendor\_id) subquery;    --Select from a table that is joined to a query with an alias of “subquery”  SELECT vendor\_name, average\_invoice\_total, last\_invoice\_date  FROM vendors v JOIN  ( SELECT vendor\_id,  max(invoice\_date) as last\_invoice\_date,  AVG(invoice\_total) AS average\_invoice\_total  FROM invoices  HAVING AVG(invoice\_total) > 4900  GROUP BY vendor\_id) subquery  ON v.vendor\_id = subquery.vendor\_id; | |

### Example of Complex query using Subqueries

|  |  |
| --- | --- |
| **Example** | SELECT summary1.vendor\_state, summary1.vendor\_name, top\_in\_state.sum\_of\_invoices  FROM  (  SELECT v\_sub.vendor\_state, v\_sub.vendor\_name,  SUM(i\_sub.invoice\_total) AS sum\_of\_invoices  FROM invoices i\_sub JOIN vendors v\_sub  ON i\_sub.vendor\_id = v\_sub.vendor\_id  GROUP BY v\_sub.vendor\_state, v\_sub.vendor\_name  ) summary1  JOIN  (  SELECT summary2.vendor\_state,  MAX(summary2.sum\_of\_invoices) AS sum\_of\_invoices  FROM  (  SELECT v\_sub.vendor\_state, v\_sub.vendor\_name,  SUM(i\_sub.invoice\_total) AS sum\_of\_invoices  FROM invoices i\_sub JOIN vendors v\_sub  ON i\_sub.vendor\_id = v\_sub.vendor\_id  GROUP BY v\_sub.vendor\_state, v\_sub.vendor\_name  ) summary2  GROUP BY summary2.vendor\_state  ) top\_in\_state  ON summary1.vendor\_state = top\_in\_state.vendor\_state AND  summary1.sum\_of\_invoices = top\_in\_state.sum\_of\_invoices  ORDER BY summary1.vendor\_state; |

# **Working with datatypes and Advanced SQL Functions**

## Formatting numbers and dates in ORACLE

|  |  |
| --- | --- |
| **Syntax** | --To change the format a number or date, it must be converted to string and then altered using mask. Oracle simplifies this with a built-in function called the TO\_CHAR() functions  TO\_CHAR(column or value, '<Format masking>') |
| **Example** | SELECT invoice\_id,  invoice\_date,  TO\_CHAR(invoice\_date, 'MON-DD-YYYY') AS varchar\_date,  invoice\_total,  TO\_CHAR(invoice\_total,'$999,999.99') AS integer\_total  FROM invoices; |
| **More info** | Note that TO\_CHAR, TO\_DATE, and TO\_NUMBER are unique to Oracle. The standard method of doing this in other Relational Database Systems is use the CAST Function. |

## Common STRING Functions

|  |  |
| --- | --- |
| **Syntax** | **LTRIM -** By default, this function removes any spaces from the left side of the specified string. If you specify a trim string, this function removes all characters specified in the trim string. LTRIM(string[, trim\_string])  **RTRIM -** Same as LTRIM but removes characters from the right side of the string instead of the left. RTRIM(string[, trim\_string])  **TRIM -** Removes any spaces from the left and right sides of the specified string. If you specify a trim character, this function removes the trim character from both sides of the string. TRIM([trim\_char FROM ]string)  **LPAD -** Pads the left side of the string to the specified length with spaces or with the characters specified by the pad string. LPAD(string, length[, pad\_string])  **RPAD -** Pads the right side of the string to the specified length with spaces or with the characters specified by the pad string. RPAD(string, length[, pad\_string])  **LOWER -** Converts the string to lowercase letters. LOWER(string)  **UPPER -** Converts the string to uppercase letters. UPPER(string)  **INITCAP -** Converts the initial letter in each word to upper- case. INITCAP(string) |
| **Example** | SELECT  LTRIM(' John Smith ') AS "LTRIM"  ,RTRIM(' John Smith ') AS "RTRIM"  ,TRIM(' John Smith ') AS "TRIM"  ,LTRIM('$0019.99', '$0') AS "LTRIM2"  ,TRIM('$' FROM '$0019.99') AS "TRIM2"  ,LPAD('$19.99', 15) AS "LPAD1"  ,LPAD('$2150.78', 15)AS "LPAD2"  ,LPAD('$2150.78', 15, '.') AS "LPAD3"  ,RPAD('John', 15) AS "RPAD1"  ,RPAD('John', 15, '.') AS "RPAD2"  ,LOWER('CA') AS "LOWER"  ,UPPER('ca') AS "UPPER"  ,INITCAP('john smith') AS "INITCAP"  ,INITCAP('JOHN SMITH') AS "INITCAP2"  FROM DUAL; |
| **More info** | <https://www.oracletutorial.com/oracle-string-functions/> |

## String Parsing and Manipulation Functions

|  |  |
| --- | --- |
| **Syntax** | **SUBSTR -** Returns the specified number of characters (length) from the string (string) at the specified starting position (start). SUBSTR(string, start[, length])  **INSTR -** Returns an integer for the position of the first occurrence of the specified find string in the specified string starting at the specified position. If the starting position isn’t specified, the search starts at the beginning of the string. If the string isn’t found, the function returns zero.  INSTR(string, find [,start])  **LENGTH -** Returns an integer for the number of characters in the specified string.  LENGTH(string)  **REPLACE -** Returns the string with all occurrences of the specified find string replaced with the specified replace string.  REPLACE(string, find, replace) |
| **Example** | --Many examples of SUBSTR in SELECT, concatenation, and in WHERE clause  SELECT vendor\_name,  vendor\_contact\_first\_name  || ' ' ||  SUBSTR(vendor\_contact\_last\_name, 1, 1) || '.' AS contact\_name,  SUBSTR(vendor\_phone, 7) AS phone  FROM vendors  WHERE SUBSTR(vendor\_phone, 2, 3) = '559'  ORDER BY vendor\_name;  --How to utilize INSTR and SUBSTR together to dynamic parse based on length and patterns  SELECT SUBSTR(name, 1, (INSTR(name, ' ') - 1)) AS first\_name,  SUBSTR(name, (INSTR(name, ' ') + 1)) AS last\_name  FROM string\_sample; |
| **More info** | <https://www.oracletutorial.com/oracle-string-functions/> |

## Converting numeric-like strings to number using TO\_NUMBER

|  |  |
| --- | --- |
| **Syntax** | *-- In situations where you have numeric data stored as characters, the data will not sort numerically or allow for arithmetic. If you want to convert text to number, use the TO\_NUMBER() function* |
| **Example** | *-- Using TO\_NUMBER() to convert text to number*  SELECT \*  FROM string\_sample  ORDER BY TO\_NUMBER(id); |

## Common NUMERIC Functions

|  |  |
| --- | --- |
| **Syntax** |  |
| **Example** |  |
| **More info** | <https://docs.oracle.com/cd/E40518_01/server.761/es_eql/src/ceql_functions_numeric.html> |

## Common DATE Functions

|  |  |
| --- | --- |
| **Syntax** |  |
| **Example** |  |
| **More info** | <https://www.oracletutorial.com/oracle-date-functions/> |

## DATE and TIME

|  |  |
| --- | --- |
| **FYI** | In Oracle, the time is automatically stored with the DATE datatype. If it is not specified, then the time is by default set to 12:00AM which is the first second of any day. You can’t see the time when you select a DATE unless you covert the date to text and then mask it to show hours, minutes, or seconds like below. |
| **Example** | SELECT date\_id,  start\_date,  TO\_CHAR(start\_date,'DD-MON-YYYY HH24:MI:SS') AS start\_date  FROM date\_sample;  SELECT \*  FROM date\_sample  WHERE start\_date = TO\_DATE('10:00:00', 'HH24:MI:SS');  SELECT \*  FROM date\_sample  WHERE TO\_CHAR(start\_date, 'HH24:MI:SS') = '10:00:00'; |
| **More info** | Tip – Remember that if you use the = operator to look for specific dates, it only returns dates with 12:00am as the time. If you have times stored, you’re better off using range operators like >= or <= . |

## Handling logical output with CASE

|  |  |
| --- | --- |
| **Syntax** | CASE [column\_name only if condition is not a range]     WHEN condition1 THEN result1     WHEN condition2 THEN result2     WHEN conditionN THEN resultN     ELSE result END; |
| **Example** | --Case on terms\_id column that returns different strings based on finite set of values  SELECT invoice\_number, terms\_id,  CASE terms\_id  WHEN 1 THEN 'Net due 10 days'  WHEN 2 THEN 'Net due 20 days'  WHEN 3 THEN 'Net due 30 days'  WHEN 4 THEN 'Net due 60 days'  WHEN 5 THEN 'Net due 90 days'  END AS terms  FROM invoices;  --Case returning different strings based on a range of values. Note no column name is used after CASE  SELECT invoice\_number, invoice\_total, invoice\_date, invoice\_due\_date,  CASE  WHEN (TO\_DATE('20-JUL-14') - invoice\_due\_date) > 30  THEN 'Over 30 days past due'  WHEN (TO\_DATE('20-JUL-14') - invoice\_due\_date) > 0  THEN '1 to 30 days past due'  ELSE 'Current'  END AS status  FROM invoices  WHERE invoice\_total - payment\_total - credit\_total > 0; |
| **More info** | <https://www.w3schools.com/sql/sql_case.asp> |

## Handling NULL values

|  |  |
| --- | --- |
| **Syntax** | **COALESCE -** Returns the first non-null value in a list.  COALESCE(*val1*, *val2*, *....*, *val\_n*)  **NVL -** Returns the value argument if the specified string is a null value. Otherwise, this function returns the specified string  NVL(string, value)  **NVL2 -** Returns the value2 argument if the specified string is a null value. Otherwise, this function returns the value1 argument  NVL2(string, value1, value2) |
| **Example** | --Example using the standard Coaleasce function  SELECT payment\_date, invoice\_due\_date,  COALESCE(payment\_date, invoice\_due\_date, TO\_DATE('01-JAN-1900'))  AS payment\_date\_2  FROM invoices;  --Example using the standard Coaleasce function  SELECT payment\_date,  NVL(TO\_CHAR(payment\_date), 'Unpaid') AS payment\_date\_2  FROM invoices;  --Example using the standard Coaleasce function  SELECT payment\_date,  NVL2(payment\_date, 'Paid', 'Unpaid') AS payment\_date\_2  FROM invoices; |
| **More info** | <https://www.w3schools.com/sql/func_sqlserver_coalesce.asp> (Works for Oracle too)  <https://www.w3schools.com/sql/sql_isnull.asp> |

## Window Functions (ROW\_NUMBER, RANK, DENSE\_RANK)

|  |  |
| --- | --- |
| **Syntax** | A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row — the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.  *WINDOW\_FUNCTION\_NAME() OVER (*  *[PARTITION BY expr1, expr2,...]*  *ORDER BY expr1 [ASC | DESC], expr2,...*  *)* |
| **Example** | --Assign unique row numbers to each group of vendors by vendor\_state, sorted by name  SELECT ROW\_NUMBER() OVER  (PARTITION BY vendor\_state  ORDER BY vendor\_name) AS row\_num,  vendor\_name,  vendor\_state  FROM vendors;  --If you use the above query as an in-line join subquery, you can use the row\_number in your WHERE  select \* from (  SELECT ROW\_NUMBER() OVER  (PARTITION BY vendor\_state  ORDER BY vendor\_name) AS row\_num,  vendor\_name,  vendor\_state  FROM vendors)  where row\_num = 10;  --Window functions that provide different versions of ranking  SELECT RANK() OVER (ORDER BY invoice\_total) AS rank,  DENSE\_RANK() OVER (ORDER BY invoice\_total) AS dense\_rank,  invoice\_total, invoice\_number  FROM invoices; |
| **More info** | <https://www.oracletutorial.com/oracle-analytic-functions/oracle-row_number/>  <https://www.oracletutorial.com/oracle-analytic-functions/oracle-dense_rank/> |

# **More Resources**

For information about Oracle’s Procedural Language (PL/SQL) check out the link below. This is how you can create stored procedures on the server and your own customer functions: <https://www.oracle.com/database/technologies/appdev/plsql.html>