Table of Contents

Data can be manipulated with operators, such as the concatenation operator for text and the arithmetic operators for numbers. Another way to manipulate data is with built-in functions which we will start looking at here. These manipulations result in new values being displayed in the query output or used in the Where clause or Order By clause. The queries shown here do not change the data in the tables.

Oracle will attempt to do automatic conversion of values from one data type to another. Relying on automatic conversion is risky. You should not assume that your data values are always clean or that the conversions work as you anticipate.

1. Data types and literals

There are several basic data types available for data to be stored in a table or to be used in expressions. The basic types are:

- String types
- Numeric types
- Date and time types

For our tables we will use only a few types of data.

1.1. Character strings

String types have two general types

- Char(99)
- Varchar2 (99)

CHAR is used for fixed length strings. We use CHAR (2) to store state abbreviations since they all have a 2 letter. If you are storing data where all of the data has the same number of characters- such as SSN, ISBN13, some product codes, then CHAR is appropriate. If necessary, the data will be end-padded with blanks to the stated length when stored.

VARCHAR2 is used for variable length strings. We could use VARCHAR2 (25) to store customer last names assuming we won't need to store a name longer than 25 characters.

For both of these - if you are defining a table column with char or varchar and try to insert a value longer than the defined length, you will get an error. Char defaults to a length of 1 if you do not state a length; varchar requires a length.

1.2. Numbers

Numeric types are divided into two categories

- Exact numbers
  - Numbers that store integers- like 15, 0, -35
  - Fixed point numbers which store numbers with a set number of digits- such as a number that can have a total of 6 digits and two of those digits are after the decimal. So that type could store a value such as 1234.56 or 0.3 but not a value such as 34.5678
- Approximate numbers
  - Floating point numbers that store values such as 5.876E2

The details of the data types is the type of stuff that you can look up in a book or manual when you need to know the range of each type. What you do need to know about numeric types is:

- Integer types are exact representation of numbers.
- Integers are faster for calculations
- Fixed decimal types may take more space and more processing time. A type such as decimal(6,2) can hold numbers from -9999.99 to +9999.99. The first value is the number of digits and the second the number of digits after the decimal point
- Floating point numbers are stored as approximation of the value; these vary with the overall size of the values they can store and the numbers of digits of accuracy. These types might be appropriate for storing a value such as a weight or distance.

1.3. Temporal Data

Date and Time data types vary more between the various dbms.

Oracle has a date type which stores a point in time as a date and time value.

1.4. Literals

Literals are constants- they have a specific value that does not change.

For string literals, enclose the data value within single quotes: `'San Francisco', 'cat';` if you need to include a quote within your literal, use two single quotes; `'San Francisco''s favorite cat';`

For numeric literals, do not use delimiters; '97' is not a number- it is a string. Do not use commas or percent symbols etc. You can use a leading + or - symbol and a single decimal point. You can also use E notation.

These are valid numeric literals: `5` `-58.89` `+ 0.002` `2.3E5` (which is equal to 23000) `2.3E-3` (which is equal to 0.00 23)

For date literals you use the ANSI standard date expression: `date '2009-05-14'.` You can also use the Oracle traditional default format of `to_date('06-JUL-09')`

2. Null

If a column in a row has no value then we say the column is null. We use a null when we do not know the value or when a value would not be meaningful. A null is not the same thing as a value of 0 or a space. Oracle has a rule that it treat a zero length string as a null. A zero length string is a string that has no characters. For example in the vt_animals table we have a column an_type for the kind of animal this is; this is defined as Not Null. We expect that the vet can figure out what type of animal this is. We also have a column an_name; this is defined as null which means we could have an animal where we do not know the animals name. Maybe the animal doesn't have a name yet. I doubt the vet would refuse to treat a puppy because the client hasn't decided on a name yet.

Nulls will need special handling in various situation we will deal with over the semester. For this unit you need to know that nulls propagate in some expressions. For example, if any of the operands in an arithmetic expression is Null, then the calculated value is Null.

Whether or not you should allow nulls in database tables will always be an issue. Some people believe that you should never allow nulls in your tables. Certainly it is easier to work with tables that cannot accept nulls.

However you might have troubles if you always insist on non-null attributes. Do you really want to turn away sales because a customer says they have no first name? Are you positive that you can legally refuse to hire someone because they have no phone number just because your employee table has a not null phone number attribute?

You might make a good argument that the order tables should never have a missing quantity or price. But perhaps you are setting up an order and the price is negotiable- you might want to store the data that you have about the customer and the items to be ordered and then fill in the price later. Perhaps the customer does not know if they want to buy 4000 refrigerators or 4500 until they know what the negotiated price is.

3. Select from the DUAL table

Sometimes we want to just see how an expression works. We can use a Select statement with an expression but Oracle requires that every Select statement include a From clause with a table. We do not really want to see data from any table but we need to use a table to have a valid statement.

Oracle provides a special table named DUAL that consists of a single row and a single column. The column name is DUMMY and the value is 'X'.

You can use this to advantage in trying out calculations, functions, etc.

```
desc DUAL
```

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
DUMMY                                              VARCHAR2(1)
```

Demo 01: Display the contents of the dual table

```
select *
from dual;
```

```
D
_
X
```

Demo 02: Display literals using the dual table. If you do not use a column alias, the expression will be used for the header.

```
select 'Hello world', 'Hello world' as Greeting, 2015 as Year, 'Paul'
from dual
;
```

```
'HELLOWORLD GREETING          YEAR 'PAU
----------- ----------- ---------- ----
Hello world Hello world       2015 Paul
```

4. Manipulating numeric data

Oracle uses the standard arithmetic operators: +, -, * and /. Arithmetic expressions can be used in the Select clause, the Where clause and the Order By clause.

Oracle follows the standard rules for arithmetic precedence. Multiplication and division are carried out before addition and subtraction. Use parentheses to change the order of evaluation.

With the expression: 5 + 3 * 8 which has no parentheses, the multiplication is done first. 5 + 24 = 29.

With the expression: (5 + 3) * 8 which has parentheses, the operation in parentheses is done, 8 * 8 = 64.

Demo 03: Use the dual table for testing arithmetic with literals as arguments.

```
select
  5 * 3 as Col1
, 5 + 8   as Col2
, 5 + 3 * 8 as Col4
, (5 + 3 ) * 8 as Col5
, (5 + 3 )  /  3
from dual;
```

```
      COL1       COL2       COL4       COL5    (5+3)/3
---------- ---------- ---------- ---------- ----------
        15         13         29         64 2.66666667
```

If any of the operands in an arithmetic expression is Null, then the calculated value is Null. Nulls propagate in arithmetic.

For working with calculations, I have created a small test table. You can create test tables easily to try out ideas. I put these tables in the a_testbed database.

Demo 04: Create the Table; see the demo for the inserts.

```
create table z_tst_calc (
   item_id integer primary key
 , quantity integer not null
 , price  decimal (6,2)
 );
```

```
   ITEM_ID   QUANTITY      PRICE
---------- ---------- ----------
       101          1     125.12
       102          5         30
       103         10     101.05
       104          1       75.5
       105         12      33.95
```

Demo 05: Using a calculated column. Price * quantity gives a value commonly called the extended cost.

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
from z_tst_calc;
```

```
   ITEM_ID      PRICE   QUANTITY EXTENDEDCOST
---------- ---------- ---------- ------------
       101     125.12          1       125.12
       102         30          5          150
       103     101.05         10       1010.5
       104       75.5          1         75.5
       105      33.95         12        407.4
```

Demo 06: Another calculation. We repeat the calculations of quoted_price * quantity_ordered for the last column. You cannot use an alias as an operand in a calculation in the Select clause.

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
 , price * quantity * 1.085 as ExtCostWithTax
from z_tst_calc;
```

```
   ITEM_ID      PRICE   QUANTITY EXTENDEDCOST EXTCOSTWITHTAX
---------- ---------- ---------- ------------ --------------
       101     125.12          1       125.12       135.7552
       102         30          5          150         162.75
       103     101.05         10       1010.5      1096.3925
       104       75.5          1         75.5        81.9175
       105      33.95         12        407.4         442.029
```

Demo 07: You cannot meaningfully use the column alias in the Select to continue the calculations. Suppose you want to add a $5 handling fee per item line.

The following does not run; you get an error message that ExtendedCost is an invalid identifier

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
 , extendedcost + 5  as ExtCostWithShipping
from z_tst_calc;
```

Demo 08: We will get to functions soon; this uses the Round function. Here it rounds the expression to two digits after the decimal point.

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
 , round(price * quantity* 1.085, 2) as ExtCostWithTax
from z_tst_calc;
```

```
   ITEM_ID      PRICE   QUANTITY EXTENDEDCOST EXTCOSTWITHTAX
---------- ---------- ---------- ------------ --------------
       101     125.12          1       125.12         135.76
       102         30          5          150         162.75
       103     101.05         10       1010.5        1096.39
       104       75.5          1         75.5          81.92
       105      33.95         12        407.4         442.03
```

Demo 09: Using a calculated value as a sort key. We could also use the clause: Order by ExtCostWithTax

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
 , round(price * quantity* 1.085,2) as ExtCostWithTax
from z_tst_calc
order by round(price * quantity* 1.085,2) ;
```

Demo 10: We are having a $50.00 off sale. This might not be such a good idea! We have negative prices

```
select item_id
 , price
 , quantity
 , price * quantity as extendedcost
 , (price- 50) * quantity as saleccost
from z_tst_calc;
```

```
   ITEM_ID      PRICE   QUANTITY EXTENDEDCOST  SALECCOST
---------- ---------- ---------- ------------ ----------
       101     125.12          1       125.12      75.12
       102         30          5          150       -100
       103     101.05         10       1010.5      510.5
       104       75.5          1         75.5       25.5
       105      33.95         12        407.4     -192.6
```

5. Concatenation of strings

Concatenating strings means to put the strings together one after the other. You use the concatenation operator `||` to concatenate strings.

Concatenation does not add spaces between the strings being put together. Include spacing by concatenating in a literal space. `|| ' ' ||`

Oracle will concatenate a null with a non-null string to give the string. This is different than some other dbms.

Demo 11: Concatenating strings. Oracle concatenates a null as if it were a zero-length string. With Oracle nulls do not propagate in concatenation. The last row displayed here does not have a value for the an_name column. The space at the start of the concatenated column value is the literal space being concatenated into the expression.

```
select an_name
, an_type
, an_name ||  ' is a '||  an_type as Animal
from vt_animals;
-- Selected rows
```

```
AN_NAME          AN_TYPE      ANIMAL
---------------- ------------ ------------------------------------------
Burgess          dog          Burgess is a dog
Pinkie           lizard       Pinkie is a lizard
Pinkie           dog          Pinkie is a dog
Yoggie           hedgehog     Yoggie is a hedgehog
                 bird          is a bird
```

Demo 12: You can concatenate values of different data types and Oracle will do type casts. Later we will see ways to improve the appearance of this output.

```
select srv_desc || ' $'  || srv_list_price As "ServicePrice"
from vt_services
where rownum <= 5;
```

```
ServicePrice
-----------------------------
Dental Cleaning-Canine $50
Routine Exam-Canine $80
Dental Cleaning-Other $100
Feline PCR Series $75
Dental Cleaning-Feline $45
```

6. Testing calculations with nulls

What you need to know for now is how nulls are treated in calculations. We will set up a small test table to look at these calculations. We will use an id column, a nullable column that stores strings, a nullable column that stores integers, and a nullable column that stores floats.

Demo 13:

```
create table z_tst_nulls (
  col_id      int          not null primary key
, col_string  varchar2(10) null
, col_int     int          null
, col_float   float        null
);
```

We need very few rows to test this.

```
insert into z_tst_nulls values (1, 'abc', 10,   10.567);
insert into z_tst_nulls values (2, 'abc', null, 20.222);
insert into z_tst_nulls values (3, null,  30,   null);
insert into z_tst_nulls values (4, null,  null, null);

select *
from z_tst_nulls ;
```

```
COL_ID               COL_STRING COL_INT              COL_FLOAT
-------------------- ---------- -------------------- ----------------------
1                    abc        10                   10.567
2                    abc                             20.222
3                               30
4
```

Demo 14: Now a few expressions

```
select col_id
, 'XYZ' || col_string  as stringTest
,  25 + col_int  as intTest
,  25 + col_float  as floatTest
from z_tst_nulls;
```

```
    COL_ID STRINGTEST       INTTEST  FLOATTEST
---------- ------------ ---------- ----------
         1 XYZabc               35     35.567
         2 XYZabc                      45.222
         3 XYZ                  55
         4 XYZ
```

Comparing these two selects, we can see that nulls propagate in arithmetic but not in string concatenation.

6.1. The Coalesce function

Since nulls create problems for us, a dbms has a function to substitute another value for the null. This function is coalesce. For now we will use coalesce with two arguments- the first will be a column name and the second argument is a value to use if the column value is null.

Demo 15: Using Coalesce where both arguments are of the same data type

```
select col_id
, coalesce(col_string, 'DataMissing')  as stringTest
, coalesce(col_int, -20)  as intTest
, coalesce(col_float, 29.95)  as floatTest
from z_tst_nulls;
```

```
    COL_ID STRINGTEST     INTTEST  FLOATTEST
---------- ----------- ---------- ----------
         1 abc                 10     10.567
         2 abc                -20     20.222
         3 DataMissing         30      29.95
         4 DataMissing        -20      29.95
```

Demo 16: Note that you have to return a numeric value for the numeric columns.

```
select col_id
, coalesce(col_string, 'DataMissing')  as stringTest
, coalesce(col_int, 'DataMissing')  as intTest
from z_tst_nulls;
```

```
, coalesce(col_int, 'DataMissing')  as intTest
                    *
ERROR at line 3:
ORA-00932: inconsistent datatypes: expected NUMBER got CHAR
```

Part of the reason for doing this is to encourage you to create small tests to find out how your dbms works. It is generally quicker to set up a small test table like this than to page through a book or try an internet search.

One of the problems with internet searches is that many web pages do not tell you which version of the software they are using, which system settings are in place- and that does not even count the web pages that are just wrong. Some web pages don't even seem to say which dbms they are using. You certainly should be able to check some things out on web pages- but you still need to verify anything you find.

7. Functions we have seen

These are some functions I have used so far that seem easy to work with.

```
extract(Month from a date value)       -- get the month number
extract(Year from a date value)        -- get the year
Round(number, position)          -- rounds at the indicated postion
Coalesce(exp1, exp2, exp3)       -- get the first not-null value


Examples
select extract(Month from date '1015-05-12')
, extract (year from date '1015-05-12')
from dual;
```

```
EXTRACT(MONTHFROMDATE'1015-05-12') EXTRACT(YEARFROMDATE'1015-05-12')
---------------------------------- ----------------------------------
                                 5                               1015
```

```
select Round(458.873), Round(458.873, 0), Round(458.873, 2), Round(458.87,-2)
from dual;
```

```
ROUND(458.873) ROUND(458.873,0) ROUND(458.873,2) ROUND(458.87,-2)
-------------- ---------------- ---------------- ----------------
           459              459           458.87              500
```

```
select coalesce(345,34), coalesce(345,null)
     , coalesce(null,34), coalesce(null, null)
from dual;
```

```
COALESCE(345,34) COALESCE(345,NULL) COALESCE(NULL,34) C
---------------- ------------------ ----------------- -
             345                345                34
```

```
select coalesce(null, null, 56, null)
from dual;
```

```
COALESCE(NULL,NULL,56,NULL)
---------------------------
                         56
```