Oracle PL/SQL Reference

Search the web [Search]

○ Search only this site
● Search te whole internet

hidden | Site map

Main

Manuals

Shell Scripts
PL/SQL manual
XSLT and Xpath
Robots and META tags
Guide to SSL
Home automation
Webservices
Encoding

Login

Table of contents

1 - Basics

1.1 - Quotes

There are 2 kinds of quotes, the single quote and the double quote.

- The single quote is used to tell the SQL statement that the value a not numeral value is.
- The double quote can only be used for a column alias. It tell SQL to use everything between the quotes as alias, and keep it case sensitive.

1.2 - Readable query's

To keep statements readable for our selves and others, it is recommended to write them widely. The use of tabs, spaces and newlines can make statements look simple and readable.

The example below displays a small select statement, but when the statements get bigger, they are still readable. It is easy to read this kind of notation (and ever easier if you get used to it).

```
   SELECT        alias1.column1
   ,             alias1.column2 AS "Column alias"
   ,             alias2.column1
   FROM          table1 alias1
   ,             table2 alias2
   WHERE         alias1.column1 = 'value'
   AND           alias2.column1 = (SELECT        alias3.column2
                                   FROM          table3 alias3
                                   WHERE         alias3.column1 < alias1.column3
                                  )
-- AND           alias1.column2 is not null
   ;
```

There are 4 reasons to write every field on a new line.

- If you have 20 field, and write them all on one line, it will be a very long line and hard to read.
- To change a field you can simply modify a line, when adding an alias, simply put it behind the field.
- When you want to skip a field, just comment the line out with -- (2x minus sign).
- When an error occurred, SQL often gives a line number. The less on one line, the faster the error can be found.

The reason for using tabs is that it is easier to find the table and column names. And nested statements can be followed easier.

Do not use aliases like in the example, but use more easy to remember aliases. Like firstname or grandtotal.

2 - Datatypes

Datatypes are database specific. The following datatype are Oracle datatypes. The most commonly used datatypes are the varchar2, number and date.

2.1 - VARCHAR2(size)

- Can store all possible characters.
- Length is depended on the data with a specified maximum.
- Default length is 1.
- Maximum size is 4000.

The size of the field depends on the data in that field. So a varchar2(10) filled with 'hello' only takes 5 bytes.

2.2 - CHAR(size)

- Can store all possible characters.
- Space reserved by the database is the specified size.
- Default length is 1.
- Maximum size is 4000.

2.3 - NUMBER(p,s)

- Can store only numeral values.
- Space reserved by the database is depending on the data.
- Default length is unknown.
- Maximum size is 38 digits.
- Scale can range from -84 to 127.

The precision (p) is the total number of digits without the decimal point. The scale (s) is the place of the decimal point from the right. So number(4,2) will give 99.99.

2.4 - DATE

- Can store only date and time.
- Space reserved by the database is fixed.
- Only one format can be entered.
- Date can range between January 1, 4712 B.C. and December 31, 9999 A.D.

The database always stores date and time.

2.5 - LONG

- Can store all possible characters.
- Length is depended on the data.
- No size can be specified.
- Maximum size is 2Gbytes.

2.6 - CLOB

- Can store all possible characters.
- Length is depended on the data.
- No size can be specified.
- Maximum size is 4Gbytes.

2.7 - RAW(size)

- Can store binary data.
- Length is depended on the data with a specified size.
- No default size, size must be specified.
- Maximum size is 2000bytes.

2.8 - LONGRAW

- Can store binary data.
- Length is depended on the data.
- No size can be specified.
- Maximum size is 2Gbytes.

2.9 - BLOB

- Can store binary data.
- Length is depended on the data.
- No size can be specified.
- Maximum size is 4Gbytes.

2.10 - BFILE

- Can store a link to an external binary datafile.
- Stored information in the database is only the link.
- No size can be specified.
- Maximum size of the external datafile is 4Gbytes.

3 - Formatting

Format type are used by conversion functions like to_date, to_char, to_number etc.

3.1 - Number format elements

Number format elements are used by to_char and to_number functions. If the number has more positions at the left of the decimal point than specified in the format. The number will be replaced by # (pound) signs.

The examples display the numbers 1234.5678 and -1234.5678.

Element	Example mask	Result	Description
9	99999	1235 -1235	Returns number with a leading space and/or a leading minus sign when negative.
0	00999	01235 -1235	Return leading zero's.
$	$9999	$1235 -$1235	Return leading dollar sign.
C	C9999	NLG1235 -NLG1235	Return leading ISO currency symbol.
L	L9999	fl1235 -fl1235	Return leading local currency symbol.
MI	9999MI	1235 1235-	Return trailing minus sign when negative or trailing space when positive.
S	S9999	+1235 -1235	Return leading or trailing minus sign when negative and plus sign when positive. Placement of the S give trailing or leading.
PR	9999PR	1235 <1235>	Places number between angle brackets when negative. And positive with a leading and trailing space.
D	9999D99	1234.57 -1234.57	Place of the decimal point.
,	9,999	1,235 -1,235	Returns , symbol at specified position.
.	9999.99	1234.56 -1234.56	Returns . (decimal point) symbol at specified position.
EEEE	9.9EEEE	1.2E+03 -1.2E+03	Returns value in scientific notation.
RN	RN	MCXXXV ########	Returns value in roman numerals. Negative values are not displayed.
FM	FM999999	1235 -1235	Returns value with no leading or trailing spaces.

3.2 - Date format elements

Date format elements are used for to_char and to_date functions. The example uses a date of dec. 31, 2001 and the time of 19:17:32.

Element	Example mask	Result	Description
D	D	1	Day number of the week.
day	day	monday	Day of the week. (length of 9 characters)

DY	DY	MO	Abbreviated name of day.
DD	DD	31	Day of the month.
DDD	DDD	365	Day of the year.
DY	DY	MO	Abbreviated name of day.
W	W	5	Week of the month.
WW	WW	53	Week of the year.
IW	IW	01	Week of the year based on ISO.
MM	MM	12	Month number.
MONTH	MONTH	december	Name of the month.
MON	MON	DEC	Abbreviated name of month.
Q	Q	4	Quarter of year.
Y	Y	1	Last (number of) digit(s) of the year. See also comment below.
YYYY	YYYY	2001	Year.
YEAR	YEAR	TWO TH...	Year, spelled out.
HH	HH	7	Hour of the day (1-12).
HH12	HH12	7	Hour of the day (1-12).
HH24	HH24	19	Hour of the day (0-23).
MI	MI	17	Minute (0-59).
SS	SS	17	Second (0-59).
SSSSS	SSSSS	69452	Seconds past midnight.

Extra information about year handling in Oracle. There are 2 year option when entering a 2 digit year. First is YY which will insert a year into the current century if the current year is between 2000 and 2049. The RR element inserts a year into the previous century when the year is greater then 49 and the current year is between 2000 and 2049.

Example:

Element	Example input	Result
YY	to_date('72','YY')	2072
RR	to_date('72','RR')	1972
YY	to_date('34','YY')	2034
RR	to_date('34','RR')	2034

4 - Select statement

The select statement is used to retrieve information from the database. The select statement selects one or more rows in a database and displays them, or passes them on to an other statement.

4.1 - Basic select

The most simple select statements gets for one, some or all columns all rows.

```
SELECT        [DISTINCT]{*, column[, column]}
FROM          table;
```

An asterisk (*) is used to tell SQL to get all columns. An other option is to give the names of the columns that SQL must retrieve. Columns are separated with a comma. Table is the name of the table.

The option DISTINCT tells the statement to get only the unique rows. So if there are 2 or more rows which are exactly the same, only the first will be displayed. Without distinct all rows will be displayed.

4.2 - Where statement

Where is used to select only the rows that meet a specific condition. How to write a condition is explained in a next chapter.

4.3 - Order by statement

The Order by statement is used to sort the output of the select statement.

```
SELECT        {*, column[, column]}
```

```
FROM          table
ORDER BY      {column, expr} [ASC|DESC];
```

By default the output is ordered ascending. To reverse the order the option descending (desc) must be used.

The output is first ordered by the first column, and then by seconds etc. It is also possible to order the output by a field that is not selected.

4.4 - Group by statement

The group by function is used to group the output. This means that for all rows that have the same value in the grouped column are grouped together. This function is mainly used for group functions which is explained in an other chapter.

```
SELECT        {*, column[, column]}
FROM          table
GROUP BY      expr
[HAVING       group condition];
```

The having statement is the same as the where statement. The only difference is that the where statement cannot be used with the group function.

4.5 - Connected by

Connect by is used to select record in a hierarchical way. If there is a parent child relation within one or more tables the connected by statement can display this relation.

```
SELECT                {*, column [[AS] alias],[LEVEL],…}
FROM                  table
CONNECTED BY PRIOR    column1 = column2
[START WITH           condition(s)];
```

The connected by prior option sets the parent child relation, where column2 is the child of column1. When the statement is executed, SQL will try to find the first parent (the first one without a parent relation). The SQL selects the first child, and then the child of the child etc. If there are no more child's, SQL will go 1 step up in the hierarchy and finds the child of the next parent etc.

The start with option can be used to tell SQL to start with the specific record. SQL will than display only this records child and there child's etc.

With the column LEVEL, SQL will display the records level in the hierarchy where 1 is the root (the one without a parent).

4.6 - Aliases

Aliases can be created at 2 points. When selecting columns and selecting tables.

```
SELECT        column [AS] alias[, ...]
FROM          table alias[, ...];
```

The option AS can be used in oracle, but may also be left out. However, there are databases who expect the AS command, and cannot be left out.
An column alias is displayed in the header of the output. It cannot be used in any statement except for the order by statement. An alias is very useful if there is a large expression at the select line.
The table alias must be used if more than one table is selected with one or more columns with the same name.

The example below has both aliases.

```
SELECT        a.NAME firstname, b.NAME lastname
FROM          FIRSTTABLE a, NEXTABLE b
ORDER BY      lastname;
```

The field name is used in both tables, so SQL does not know when to select which table. It is also possible to select the complete table and column like FIRSTTABLE.NAME but it is shorter to use an alias. Tablenames are often very long.

The column aliases are printed in the header, so the header doesn't display a.NAME but firstname and lastname.

5 - String manipulation

Fields can be manipulated with to following statements.

5.1 - Lower

Lower converts character data to lowercase.

```
LOWER (column|expr)
```

5.2 - Upper

Upper converts character data to uppercase.

```
UPPER (column|expr)
```

5.3 - Initcap

Initcap converts the first character of every word to uppercase and every other character to lowercase. Words are separated by spaces, tabs and special characters. Words contain alpha-numeral characters.

```
INITCAP (column|expr)
```

5.4 - Concat

Concat joins columns and/or expressions together so that SQL handles it as one column. CONCAT (column|expr, column|expr[,...]) In some database (eq. oracle) two pipes (||) can also be used to join columns and expressions.

```
SELECT column || column, 'text' || column
```

5.5 - Substr

Substr selects a part of the column or expression output.

```
SUBSTR (column|expr, n[, m])
```

The first option (n) is the start point (in characters) from where the column must be displayed. The second option (m) is the length (in characters) that must be displayed.

5.6 - Length

Length displays the length (in characters) of a column or expression output. The output is a number format.

```
LENGTH (column|expr)
```

5.7 - Instr

Instr gives the position (in characters starting from left) of a certain character or characters.

```
INSTR (column|expr, m)
```

he option m is character(s) that must be found. The option can be one single characters, a string or even a column.

5.8 - Lpad/Rpad

Lpad and Rpad fills a string up to the given number of characters. Lpad inserts the new characters from the left, and Rpad inserts them from the right.

```
LPAD (column|expr, n[, m])
```

Option n is the length or the output in characters. Option m can contain the character that is used to fill the string in stead of spaces.

5.9 - To_char

To_char converts a number or date to a string field.

```
TO_CHAR (number|date[,fmt])
```

The number or date can be the output of a column or expression. If a date is used, the option fmt sets the date format.

5.10 - To_number

To_number converts a string which only contains numeral characters to a real number.

```
TO_NUMBER (string)
```

5.11 - To_date

To_date converts a string containing a date to a real date.

```
TO_DATE (string[, fmt])
```

The string must contain a valid date in a valid format. The format can be set by the fmt option.

5.12 - Nvl

Nvl converts a null value to a user specified value.

```
NVL (column|expr, m)
```

If the column or expressions contains a null value, the null is replaced by the character or number in option m. When the column or expression contains something else than null, nothing is done.

5.13 - Decode

Decode converts anything to anything.

```
DECODE (column|expr, search1, result1[,...][,default])
```

If the output of the column or expression is search1 than it is replaced by result1. The last search without a result will be handled as default. This means that if the column doesn't contain any of the searches it is replaced by the default.

5.14 - Round

Round is used to round numbers or dates.

```
ROUND (column|expr|date,n|fmt)
```

If a number is given, it will be rounded to n decimals. If a date is given it will be rounded to the time and date to the nearest fmt format. Dates later than 12:00am will be changed to the next day.

5.15 - Trunc

Trunc strips the given number or date.

```
TRUNC (column|expr|date,n|fmt)
```

If a number is given, it will be stripped to n decimals without rounding. A date is stripped to the time and date given in the fmt format. Dates and Times are truncated to 12:01:00 at the first day of the first month.

5.16 - Mod

Mod returns the remaining of a division.

```
MOD (m,n)
```

The number m is divided by n to a whole number, the remaining is displayed.

5.17 - Months_between

Months_between calculates the months between 2 dates. The number of months is given in a floating number. To get the full months between 2 dates, the output can be truncated with trunc.

```
MONTHS_BETWEEN (date1, date2)
```

If date1 is the oldest date, the output is a positive number.

5.18 - Add_months

Add_monts adds a number of months to a date.

```
ADD_MONTHS (date,n)
```

Option n is the number of months, this can be a floating number.

5.19 - Next_day

Next_day gives the date of the next specified weekday.

```
NEXT_DAY (date,'day')
```

5.20 - Last_day

Last_day gives the date of the last day of the month.

```
LAST_DAY (date)
```

6 - Where clause

6.1 - Boolean operators

The 3 standard comparison operators can be used in where clauses. They can also be combined. This gives the following options:

Element	Description
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equalt to
<>	Not equal to
!=	Not equal to

The last 2 options (and !=) function the same. Comparison operators can be used to compare 2 field or 1 field with a value.

```
WHERE field1 = 123
WHERE field1 = field2
WHERE 123 = field1
```

The examples above are 3 separate statements.

6.2 - BETWEEN

To select records where a field must contain a value within a range the BETWEEN operator can be used. Between checks if a field lays within the given range. Ranges must be set with the lowest value first.

```
WHERE field1 BETWEEN 100 AND 200
```

Select all records where the value of field1 lays between 100 and 200.

6.3 - IN

To select only those record where a field value is equal to a value in a list, the IN operator can be used.

```
ERE field1 IN (100, 120, 300)
```

Select all record where the value of field1 is equal to 100 or 120 or 300.

6.4 - LIKE

To find records where a field value is not exactly known the LIKE operator can be used. The like operator can make use of wildcards. These wildcards are: INSERT TABLE!!! To use a real % sign in a like operator, use the \ sign. The escape character (\) must be specified in the ESCAPE operator.

```
 WHERE field1 LIKE '_A\_%'
```

Returns all records where the value of field1 contains the letter A on the second position and a real _ character on the 3rd position followed by none or more characters.

6.5 - IS NULL

To select field only containing the null-value it is not possible to use the = (equal) sign. To check the field value on null use the IS NULL operator.

```
WHERE field1 IS NULL
```

The IS operator can not be used in any other way.

6.6 - Logical operators

To combine comparison statements logical operators can be used. There are 3 operators:

Element	Description
AND	True if statement 1 is true AND statement2 is true
OR	True is statement 1 is true OR is statement 1 is not true but statement 2 is true
NOT	True if statement is NOT true OR not true if statement is true

The first statement must be proceeded by the where statement.

```
WHERE field1 = 1
AND field2 = 100
OR field2 = 200
```

NOT statement can be used with advanced operators. The syntax is as follows:

```
WHERE NOT field1 IN (100, 200)
OR field1 NOT BETWEEN 100 AND 200
OR field1 NOT LIKE '%A%'
OR field1 IS NOT NULL
```

The rules of precedence (which operator comes before which operator) is standard mathematical.

7 - Group functions

7.1 - AVG

AVG calculates the average value of the selected fields ignoring the null-values.

```
SELECT AVG([DISTINCT|ALL] expr)
```

By default the functions selects all values, but with distinct it can select all unique values.

7.2 - COUNT

Count the number of fields. To count the number of returned record, it is best to select all (*) fields. This will not cause the database to retrieve all record, it just counts them (based on the smallest index).

A count on a field will exclude null-values, a count(*) will include null-values.

```
SELECT COUNT(*|[DISTINCT|ALL] expr)
```

Count counts all selected records, to count only the unique records use the distinct option.

7.3 - MAX

Returns the largest value found in the selected records, ignoring the null-values.

```
SELECT MAX([DISTINCT|ALL] expr)
```

By default it returns all the maximal values, with distinct it will return only one. Even if the 2 maximal values are equal.

7.4 - MIN

Return the smallest value found in the selected records, ignoring the null-values.

```
SELECT MIN([DISTINCT|ALL] expr)
```

By default it returns all the minimal values, so if the 2 smallest are equal, it returns 2 values. Distinct will display only one.

7.5 - STDDEV

Return the standard deviation of the selected fields.

```
SELECT STDDEV([DISTINCT|ALL] expr)
```

The distinct option will cause the function to return the standard deviation of the unique values.

7.6 - SUM

Returns the sum of all values, ignoring the null-values.

```
SELECT SUM([DISTINCT|ALL] expr)
```

By default sum will calculate the sum over all records, distinct will do it over the unique ones.

7.7 - VARIANCE

Returns the variance of all selected fields, ignoring null-values.

```
SELECT VARIANCE([DISTINCT|ALL] expr)
```

The distinct option will cause the function to return the variance of the unique values.

7.8 - GROUP BY

Group functions return only 1 line of information. Selects with normal fields and group functions, would want to return one line and multiple lines. SQL does not understand this. The group by function tells SQL to display the field grouped together so SQL will return for every unique field value the output of the group function on that field value.

```
SELECT field1, AVG(field2) FROM table GROUP BY field1;
```

The above query will return a list ordered by unique field1 values and the average of field2 in the records where field1 is the same.

Rule of tumb:
Every field or non group function in a select statement must be entered in the group by statement if a group function is used.

7.9 - HAVING

Where clauses can not be used to limit group functions when the group by statement is used. To restrict the group by function the having function must be used. All groups are displayed from witch the having condition is true.

```
SELECT        field1, AVG(field2)
FROM          table
WHERE         field1 = 10
GROUP BY      field1
HAVING        field2 IN (100, 200)
ORDER BY      field1;
```

The order of the statements is mandatory.

8 - Sub queries

8.1 - Single row subquery

A single row subquery returns only 1 value to the main query. This type of query is often used with group functions.

```
SELECT        field
FROM          table
WHERE         field = (SELECT        MIN(field2)
                       FROM          TABLE2 );
```

If the subquery returns more than 1 value, SQL will generate an error. It is of course also possible to use the where statement to select only 1 record. If no record is selected the statement will return the null-value, this will not result in an error.

8.2 - Multiple row subquery

The multiple subquery returns more than 1 value. Since it is not possible to use the standard operators, SQL has 3 special operators for multiple row subqueries.

IN
Returns all records containing the values of the subquery in the specified field. Acts like the standard IN

operator.

```
SELECT        field
FROM          table
WHERE         field IN (SELECT        field
                        FROM          table );
```

ANY
Returns all records smaller or larger than ANY of the selected values. Any must be used with < (less) of > (greater), but = (equal) can not be used.

```
SELECT        field
FROM          table
WHERE         field < ANY (SELECT        field
                           FROM          table );
```

The example will return all records where field is smaller than any (in this case the largest) of the fields in the subquery.

ALL
Returns all records smaller or larger than ALL of the selected values. All can only be used with < and >.

```
SELECT        field
FROM          table
WHERE         field < ALL (SELECT        field
                           FROM          table );
```

The example will return all records where field is larger than all (in this case the largest) of the fields in the subquery.

9 - Data manipulation

9.1 - INSERT

With the command insert, a new record can be added to a table.

```
INSERT INTO   table [(column[, column...])]
VALUES        (value[, value...]);
```

Implicit
Insert values listed by column. There are 2 reasons way this option must be used. First to insert only the values that must be filled. On large tables only some column have to be filled.

The 2nd reason is when the statement is used in a program script. If a new column is added to the table, the implicit statement will still work the same. When the new column is not a mandatory field off course. With the command insert, a new record can be added to a table.

```
INSERT INTO   table (column1, column3)
VALUES        (100, 'text');
```

The table may consists of more than 2 columns.

Explicit
Insert values as they appear in the table. The number of values must be equal to the number of columns. Also they must be in the same order as the columns. The reason for using this method is because it is quick and dirty.

```
INSERT INTO   table
VALUES        (100, NULL, 'text');
```

The table must have 3 columns. The seconds column will be filled with the null-value, this means it will be empty.

9.2 - INSERT based on subquery

Records can be inserted based on the result of a subquery. The number of rows generated by the subquery is the number of records the statement will insert.

```
INSERT INTO   table (column1, column2)
        SELECT        name, address
        FROM          customers
        WHERE         credit > 1000;
```

This method requires the implicit mode. The VALUES statement may not be used here.

Always test the select statement first before inserting records this way.

9.3 - UPDATE

Data can be modified by the update statement. The update statement will update all records for witch the where clause is true. So without the where clause the update statement will update the entire table.

```
UPDATE table SET column = value [, column = value...] [WHERE condition];
```

The update statement act exactly like the select statement. So sub queries and multiple column sub queries can also be used with the update statement.

Always test the where clause in a select statement. If it selects the right records, than use it in the update statement.

9.4 - DELETE

The delete statement deletes records from a table. The where clause limits the records that will be deleted. If no where clause is used, all records will be deleted.

```
DELETE [FROM] table [WHERE condition];
```

Delete can handle sub queries and multiple column sub queries. The FROM option is optional in Oracle but makes the statement easier to read. The command DELETE table does not delete the table, only its rows.

Always test the where clause in a select statement. If it selects the right records, than use it in the delete statement.

10 - Database transactions

Data manipulation commands are not directly written to the database. All changes are kept in a buffer. Other users will see the old data until the changes are committed. When the program (like sqlplus) is exited in a normal way the data will be committed to the database. So changes are saved. But when a program is aborted or even if the database is killed, changes will not be saved. Data will stay as it was before the first change. The first change is the point the user logged onto the database or when the last changes were committed.

10.1 - COMMIT

To save all changes made to the data to the database the commit command is used. After a commit is given it is not possible to undo the changes.

```
COMMIT;
```

After the commit all locks are released and the buffers are freed.

10.2 - SAVEPOINT

A savepoint is used when data may not be commit yet, but you want to be able to rollback this point without undoing all other changes you've made. Savepoints must be named, so the database will know where to rollback to.

```
SAVEPOINT name;
```

It is not possible to commit all changes before a certain savepoint, you have to rollback to the savepoint first. When you rollback to a savepoint or commit all changes, the savepoint will deleted.

10.3 - ROLLBACK

The rollback command is used to undo the changes that are done on the data. A simple rollback will undo all changes made since the last commit or when the user logged on. If a savepoint is specified, rollback will undo all changes made since the savepoint was saved.

```
ROLLBACK [TO name]
```

Name is the name of a savepoint. Every savepoint made after the specified savepoint and the specified savepoint itself will be deleted.

11 - Managing tables

This chapter describes how to manage tables. To manage views, indexes etc. the same commands are used, but the will be handled in a later chapter. All Data definition commands update the database instantly so they do not need to be committed, but they can also not be rolledback. A DDL command acts like a commit, so all previous transactions are also committed.

11.1 - CREATE

The create statement is used to create new tables. When creating a table, the columns are also created. It is however possible to change the columns later.

```
CREATE TABLE    [schema.]table
                (column datatype [DEFAULT expr]
                [column_contraint]
                ,...
                [table_contraint]);
```

The schema is the name of the schema on where the table will be created. By default every user has a schema with his own loginid. If no schema is specified, the default user schema will be used. The default option can specify a default value that is used when adding records. The next chapter will handle constraints witch can be added to the table.

11.2 - CREATE base on subquery

When you want to create a table containing data that is already available in other tables, a table can be created on a subquery. The result of the subquery will be written to a new table containing the exact columns and types for this data.

```
CREATE TABLE    [schema.]table
                [column[,column...]]
AS              subquery;
```

The column names are optional, if none is given the name of the columns in the subquery are used. To generate a empty table make a where clause witch is never true.

```
CREATE TABLE    customers
AS
        SELECT          name, address
        FROM            clients
        WHERE           1 = 2
```

11.3 - ALTER

Alter can be used to change the following things on an existing table:

- Add a new column or constraint.
- Modify an existing column or constraint.
- Define a default value for a column.

The constraints are handled in an other chapter.

```
ALTER TABLE    table
ADD|MODIFY     (column datatype [DEFAULT expr],...);
```

A new column will always be added as the last column.

Columns can only be modified by datatype, size and default value. Columns can not be renamed.

11.4 - DROP

To delete a complete table the drop table command is used. The table definition and the date are deleted from the database.

```
DROP TABLE table;
```

The drop command can not be rolledback. With the drop command all indexes on the table are also deleted.

11.5 - TRUNCATE

Truncate is used to empty a table without deleting the table itself. The difference between delete and truncate is that delete can be rolledback, truncate can't. But truncate releases the storage space used by the table.

```
TRUNCATE TABLE table;
```

11.6 - RENAME

Rename can be used to rename a table to a new name.

```
RENAME old_name TO new_name
```

This command can not be rolledback, but it can of course be reversed.

11.7 - DESCRIBE

Describe displays all information about a table. The name and types of the columns and there default value. It also displays if a column may not be null.

```
DESCRIBE table
```

12 - Constraints

Constraints are the rules for inserting, changing or deleting data.

12.1 - NOT NULL

The not null constraint forces the column to be filled with anything, so the column can not be empty (null). This constraint can only be used at column level, not at table level. The not null constraint can be viewed with the describe command.

```
CREATE ...(column varchar2(10) CONSTRAINT cons_name NOT NULL,
```

12.2 - UNIQUE key

The unique key forces that the records must have unique values in the specific column. The table may contain 1 null value in the column. Because the null value is unique when it is only used once.

```
CREATE ...(column varchar2(10) CONSTRAINT cons_name UNIQUE,
CONSTRAINT cons_name UNIQUE(column));
```

The unique constraint can be entered at column level and at table level. The unique constraint will create an unique index based on the constraint.

12.3 - PRIMARY key

The primary key constraint look like a unique key. There are only 2 differences between them. First only 1 primary key can be created on a single table. A primary key may not contain a null value.

```
CREATE ...(column varchar2(10) CONSTRAINT cons_name PRIMARY KEY,
CONSTRAINT cons_name PRIMARY KEY(column));
```

The primary key can also be entered at column or table level. The primary constraint creates an unique index on the table based on the primary key.

12.4 - FOREIGN key

The foreign key point to a primary or unique key. It is also called a referential integrity constraint. A foreign key value must match a value in the primary or unique key its pointing to. If valid value can be found, it must have the null-value. A foreign key with a null-value does not point to a unique key with a null-value.

```
CREATE ...(column varchar2(10)
        CONSTRAINT name_fk FOREIGN KEY REFERENCES table (column),
CONSTRAINT name_fk FOREIGN KEY (column) REFERENCES table (column));
```

The foreign key can be used at both column level and table level. Records can not be inserted, changed or deleted when it will break a unique/primary - foreign constraint.

When using the ON DELETE CASCADE option, the database will delete the record containing the constraint when be deleting the record it must break the foreign key constraint. But when it deletes the record, it will also delete its child record.

12.5 - CHECK

The check constraint will check if the entered data satisfies an expression. The expression can be any expression know by the database. It does however may not contain subqueries other than queries that use only the manipulated row itself.

```
CREATE ...(column varchar2(10)
        CONSTRAINT cols_name CHECK (column BETWEEN 1 AND 3),
```

The check constraint can be used at column level and table level.

12.6 - Create a table with constraints

You can add constraints when creating a table. The constraints are entered in the create table statement.

```
CREATE TABLE    [schema].table
                (column datatyep [DEFAULT expr]
                [column_constraint]
                ,...
                [,table_constraint]);
```

The null-value constraint is often added while creating the table.

12.7 - Add a constraint

To add a constraint, use the alter table statement.

```
ALTER TABLE     table
ADD             CONSTRAINT table_constaint;
```

You can only add table constraints. A not-null constraint can not be used as table constraint, so it can not be created by the above alter table statement.

To add a not-null constraint use the alter table modify statement.

```
ALTER TABLE     table
MODIFY          column datatype CONSTRAINT column-constraint;
```

12.8 - Drop a constraint

To remove a constraint use the alter table statement with the drop option.

```
ALTER TABLE     table
DROP PRIMARY KEY|UNIQUE(column)|CONSTRAINT cons_name [CASCADE];
```

To drop a primary constraint the name of the constraint does not have to be specified since a table can have only 1 primary constraint. The unique constraint however must be specified by column name or constraint name. The cascade option will remove al parent and child constraints witch will be broken if the constraint is removed.

12.9 - Disable a constraint

Constraints can be disabled. They are not removed but simply stop working. This can be very handy while loading sequential data into a number of tables. The data does not have to be inserted in the right order.

```
ALTER TABLE     table
DISABLE CONSTRAINT cons_name [CASCADE];
```

The cascade option will also delete the dependent constraints when a integrity constraints is disabled.

12.10 - Enable a constraint

Disabled constraints can be enabled with this statement.

```
ALTER TABLE     table
ENABLE CONSTRAINT cons_name;
```

Constraints can only be enabled if the data matches the constraint. If a unique or primary key constraint is enabled an index will also be created if it does not exist. The enable constraint does not have a cascade option. So constraints disabled by the cascade option must be enabled one by one.

12.11 - View a constraint

Only the not-null constraint is visible by the describe statement. All other constraints are only visible in the user_constraints table (and the dba_constraints table). The names of the columns on where the constraints apply to are visible in the user_cons_columns table (and dba_cons_columns table).To view all user constraints on a specific table use the next statement.

```
SELECT        a.table_name
,             a.column_name
,             b.constraint_name
,             b.constraint_type
,             b.search_condition
FROM          user_cons_columns a
,             user_constraints b
WHERE         a.constraint_name = b.constraint_name
AND           a.table_name IN ('TABLE1', 'TABLE2')
ORDER BY      a.table_name
,             a.column_name;
```

It is also possible to view the constraints with special applications like designer and the enterprise manager.

13 - Sequences

Sequences can be used to generate serial numbers when inserting data into tables. A sequence generates a new number every time it is called.

13.1 - Create a sequence

Sequences can be created like any other object in an Oracle database by using the create sequence command. The command has several options, some of them are listed below:

Option	Description
INCREMENT BY	Specifies the interval between sequence numbers. Can be positive and negative, but not null.
START WITH	Specifies the start number.
MAXVALUE	Specifies the maximum value for a sequence.
MINVALUE	Specifies the minimal value for a sequence.
CYCLE	Specifies that after reaching the maximal or minimal value, the sequence rotates by starting again at the start value.
NOCYCLE	Specifies that the sequence doesn't reuse its numbers after reaching the maximal or minimal values.

By default, a sequence starts with the number 1 and is incremented by 1.

```
CREATE SEQUENCE schema.sequencename
                START WITH 10
                INCREMENT BY 5
                MAXVALUE 500
                CYCLE;
```

The example statement creates a sequence starting with 10. Every next number is incremented by 5 until it reaches 500. After reaching its maximal value, the sequence continues starting with 10 again.

13.2 - Alter a sequence

The alter statement is used to change the behavior of an existing sequence. The same options as the create sequence statement can be used.

```
ALTER SEQUENCE scheme.sequencename
      INCREMENT BY 3;
```

The sequence is changed so that the next value is incremented by 3.

13.3 - Drop a sequence

To delete a sequence use the drop statement. No options are available, and it can't be rolled back.

```
DROP SEQUENCE scheme.sequencename;
```

If a dropped sequence is recreated, it will start as a new sequence and will not remember the old 'current' value of the deleted sequence.

13.4 - Using sequences

Using a sequence is easier than it looks. A sequence is not a real table, but we can use it as if it is a table. A Sequence has two Pseudocolumns CURRVAL and NEXTVAL. The CURRVAL column returns the current sequence number. The NEXTVAL column generates a new number and returns it. The columns can be called from in any existing table. The most commonly used table to call these columns is DUAL.

```
SELECT        schema.sequencename.CURRVAL
,             schema.sequencename.NEXTVAL
FROM          DUAL;
```

The above example returns the current value of the sequence in the first column. The next column displays the next (new) value of the sequence.

```
INSERT INTO   tablename
VALUES        ((        SELECT  sequence.NEXTVAL
                        FROM    DUAL)
,             'column2');
```

This example inserts a record into a table where the first column contains a new sequence number (serial number) and the second some text.

14 - Variables

With SQL/plus comes the possibility to use variables in your SQL statements and scripts. I use a lot of small scripts which retrieve data depending on a variable. The script will ask to fill the variable. (it is explained later).

14.1 - & substitution variable

This is the basic, and most commonly used, variable. This type of variable is also used for user-defined and system variables. To use this variable enter a & sign before the variable name.

```
SELECT        field1
,             field2
,             &variable1
FROM          table
WHERE         field1 = &variable2
ORDER BY      &variable1;
```

Variables can be used in several places as you can see above. If SQL/plus does not know the variable (not a system or user-defined) it will ask the variables value while executing the statement. SQL/plus does not remember the variable when it's used this way. So the next time it passes the same variable name, it will ask its value again.

```
Enter value for variable1: field3
Enter value for variable2: 'text'
Enter value for variable1: field1
```

In the example you can see it is possible to use 2 different values for the same variable in the same statement.

14.2 - && substitution variable

This type of variable causes SQL/plus to remember its value. So if you use the same variable name in several parts of your statement, it will only ask for it's value once.

```
SELECT        field1
,             &&variable1
FROM          table
WHERE         &&variable1 = 123;
```

```
Enter value for variable1: field2
```

The only disadvantage of this variable type is than SQL/plus also remembers it when the statement is finished. So the same statement is executed again, it will not ask for the value of it's variable since it already knows it. The value is remembered until you logoff or the variable is undefined.

14.3 - ACCEPT

Normally SQL/plus generates a question as in the previous chapter. (Enter value for variablename:) This is often a very unfriendly question. With accept it is possible to define your own question.

```
ACCEPT variablename [datatype] [FORMAT format] [PROMPT text] [HIDE]
```

The prompt option specifies the text that must be printed instead of the default question. If the hide option is enabled, the characters the user types are not displayed. This is often used for passwords.

The accept command will create a variable that SQL/plus will remember. So if the variable is used several times, its value has only to set once with accept. The next time the accept command is executed, it will ask for a new value. The old value will be deleted, even if the you only give an enter.

14.4 - DEFINE

Define creates a variable or gives it a new value.

```
DEFINE variablename [= value]
```

If no value is given, it will create a variable that contains the null-value. The null-value does not mean it is empty, the variable will work in statements!

14.5 - UNDEFINE

To delete a value use the undefine command. Undefine will not empty the variable, but removes it completely.

```
UNDEFINE variablename
```

If a undefined variable is used in a statement, it will be treated as a & substitution variable.

15 - System variables

This chapter handles some system variables that makes life easier. To load the variable automatically when you logon to SQL/plus, make a file named login.sql. The commands and statements in this file are automatically executed on logon.

15.1 - _EDITOR

The _EDITOR variable defines the editor. The build in editor from SQL/plus is a line editor. Some of us can't work with line editors. With this variable you can change it to any Unix editor e.g. vi.

```
DEFINE _EDITOR=vi
```

16 - Customizing output

The following SQL/plus commands change the report layout, the layout will be changed for the whole session. The only way to get the default layout back is to logoff and logon again.

16.1 - COLUMN

With the column command you can change the layout of the column. The changes will be affected to all columns and aliases named by the command.

```
COL[UMN]  [{COLUMN|ALIAS} [OPTION]]
```

Column without a column or alias name will display all setting for all columns. The column with only a column or alias name will display the settings for that column or alias. The command CLEAR COLUMN will erase all formats for all columns.

Option	Description
CLE[AR]	Clears any columns format.
FOR[MAT] format	Changes the format of the column.
HEA[DING] text	Changes the column heading, a semicolon (|) forces a linefeed.
JUS[TIFY] [align]	Justifies the column heading to left, center or right. It does not change to data layout.
NOPRI[NT]	Hides the column.
NUL[L] text	Specifies text that must be printed for null-values.
PRI[NT]	Show the column.
TRU[NCATED]	Truncates the string at the end of the first line.
WRA[PPED]	Wraps the end of the string to the next line.

The following example changes the display format of the column FIELD1. The values will be printed in the new format.

```
COLUMN FIELD1 FORMAT $009999
     FIELD1 FIELD2  FI FIELD4
----------- ------- -- ----------------------------
    $001234 TEXT    01 Some text here
    $001236 TEXT    04 Another text here
```

The format option has several format models, only the most common are displayed here.

Option	Desription
An	Sets the display width to n characters.
9	Single digit with no leading zero.
0	Single digit with leading zero forced.
$	Floating dollar sign, this is not a currency.
L	Floating local currency sign.
.	Position of decimal point. (9999.99 will be 1234.56)
,	ousand separator. (9,999 will be 1,234)

If the number of digits in a field exceeds the number of digits in the format model, Oracle will display pound (#) signs.

16.2 - TTITLE

This command is used for report headers. The header always contains the print date in the upper left and the page number in the upper right. Optionally is a title text.

```
TTI[TLE] [text|OFF|ON]
```

With off and on you can set the title off or on. If a text contains more than 1 word, use single quotes. A semi column (|) will create a linefeed. The text is always centered.

16.3 - BTITLE

This command is used for report footers. It will print a text at the bottom of the page. It does not contain any logic information like page numbers etc.

```
BTI[TLE] [text|OFF|ON]
```

The off and on option will turn the footer off or on.

16.4 - BREAK

Break will force a pagebreak or blank lines on a value change. The command is used to group the same values together. To use it effetely specify the break columns in the order by statement.

```
BREAK ON {column|alias|row} [SKIP n|DUP|PAGE] ON ... [ON REPORT]
```

When the value of the specified column change it will SKIP n number of lines or insert a PAGEbreak. The option DUP will display duplicate values. Default duplicate values are not printed. The option ON REPORT will generate a grand total if the column uses the number format.

The command CLEAR BREAK will delete the break options.

16.5 - FEEDBACK

By default SQL/plus displays the number of selected records at the end of the report if the number of selected records is more than 6. The set-variable feedback controls this.

The option off and on turns the feedback off or on. You can also specify when the feedback must be printed by setting the n value.

16.6 - LINESIZE

The set-variable linesize sets the number of characters that can be displayed on 1 line.

```
SET LIN[ESIZE] n
```

By default the linesize is 80 characters, but can be set to n characters.

16.7 - PAGESIZE

The set-variable pagesize sets the number of lines that can be displayed on 1 page including the titles etc.

```
SET PAGES[IZE] n
```

The default pagesize is 24

16.8 - PAUSE

The set-variable pause enables or disables a pause at the begin of every page starting with page 1. The user must press a key to continue. The pause will be displayed before the report prints its first page.

```
SET PAU[SE] {OFF|ON|text}
```

With the text option you can create your own pause text.

17 - Customizing output

PL/SQL scripts are programmed in blocks. Blocks may be nested. The maintypes of blocks are declared here.

17.1 - Anonymous

This is the most common PL/SQL block type. It is often used as a single program or past as a program.

```
[DECLARE]

BEGIN
  -- statements

[EXCEPTION]

END;
```

The declare part is part where variable, cursors etc. are declared. It works the same as in program languages like Pascal. After begin the actual program starts. It may contain loops, function calls etc. But its also possible to include another block.

The exception part is used for error handling. When no error accurse, this part will be skipped. The end statement is the end of this block.

17.2 - Procedure

A procedure is a small program that can be called from anywhere. The procedure will be stored at application or server level. A procedure is used for repeated actions.

```
PROCEDURE name
IS

BEGIN
  -- statements

[EXCEPTION]

END;
```

All procedures must have an unique name. In a procedure variables can not be declared. Errors are handled in the exception part.

17.3 - Function

A function is almost the same as a procedure, the only difference is that a function must return a value. Functions are also stored, and they are used for computing a value.

```
FUNCTION name
RETURN datatype
IS
BEGIN
  -- statements
  RETURN value;

[EXCEPTION]

END;
```

Functions must also have an unique name. The return option defines the datatype of the value that the function returns. The value that is returned is specified at the return option of the begin part. The value may also be a SQL statement. Errors are handled in the exception part.

18 - System management examples

This chapter contains some example scripts that can be used to manage an Oracle database.

18.1 - Find running statements

This script is used to display all logged on users who are running a statement.

```
SELECT    s.username
,         s.status
,         a.sql_text
FROM      v$session s
,         v$sqlarea a
WHERE     s.username IS NOT NULL
            -- Ignore the Oracle background processes
AND       s.audsid <> userenv('SESSIONID')
            -- Ignore the current session
AND       s.sql_address = a.address
AND       s.sql_hash_value = a.hash_value
ORDER BY  s.status
            -- to have active sessions first
```

The script looks at 2 tables v$session which contains information about active sessions and there username. The second table is v$sqlarea which contains the information about all executed statements. The address and hashvalue are used to find the statement of a user.