Hoofdstuk1: Retrieving data using the SQL SELECT statement

Capabilities SELECT statement:

1. **Projection:** select the columns in a table that are returned by a query. Select as few or as many of the columns as required.
2. **Selection:** Select the rows in a table that are returned by a query. Various criteria can be used to restrict the rows that are retrieved.
3. **Joining:** Bring together data that is stored in different tables by specifying the link between them.

Basic select statement:

Select \* |{[DISTINCT] column | expression [alias],….}  
 From table;

Which means:

SELECT is a list of one or more columns  
 \* selects all columns  
 DISTINCT suppresses duplicates  
 column|expression selects the named column or the expression  
 alias gives the selected columns different headings  
 FROM table specifies the table containing the columns

Select statements must include the following:  
 1) a SELCT clause, this will specify the columns to be displayed  
 2) a FROM clause, which identifies the table containing the to-be-displayed columns

Guidelines to make a statement easy to read and edit:

1. SQL statements are not case-sensitive (unless indicated)
2. SQL statements can be entered on one or many lines
3. Keywords cannot be split across lines or abbreviated
4. Clauses are usually placed on separate lines for readability and easy of editing
5. Indents should be used to make code more readable
6. Keywords typically are entered in uppercase; all other words, such as table names and column names are entered in lowercase

Create expressions with number and date data by using arithmetic operators:

|  |  |
| --- | --- |
| Operator | Description |
| + | Add |
| - | Substract |
| \* | Multiply |
| / | Divide |

# The Null Value

Null is a value that is unavailable, unassigned, unknown or inapplicable.  
Null is not the same as zero or a blank space

If a row lacks a data value for a particular column, that value is said to be null or to contain a null.

# Column Alias

1. Renames a column heading
2. Is useful with calculations
3. Immediately follows the column name (There can also be the optional AS keyword between the column name and the alias.)
4. Requires double quotation, marks if it contains spaces or special characters, or if it is case-sensitive.

Examples:

Select last\_name AS name, commission\_pct comm  
 From employees;

Select last\_name “Name”, salary\*12 “Annual Salary”  
 From employees;

Not using “ “ to give the alias name will result in the alias as a uppercase name  
double quotation marks also required if the alias contains a space.

# Concatenation Operator

1. Links colums or character strings to other columns
2. Is represented by two vertical bars ||
3. Creates a resultant column that is a character expression

Example:

Select last\_name||job\_is AS “employees”  
 From employees;

The As keyword before the alias name makes the SELECT clause easier to read

# Literal Character Strings

1. A literal is a character, a number, or a date that is included in the SELECT statement.
2. Date and character literal values must be enclosed within single quotation marks.
3. Each character string is output once for each row returned.
4. It is printed for each row returned

Date and character literals MUST be enclosed within single quotation marks (‘ ‘);

# Alternative Quote (q) Operator

1. Specify your own quotation mark delimiter
2. Select any delimiter
3. Increase readability and usability

Example:

Select department\_name || q’[Department’s Manager ID: ]’||manager\_id   
 AS “Department and Manager”  
 From departments;

The string contains a single quotation mark, which is normally interpreted as a delimiter of a character string. By using the q operator, the brackets [] are used as the quotation mark delimiters. The string between the brackets delimiters is interpreted as a literal character string.

If you’d like to be fancy, this shit works as well with the following character pairs:

1. [ ]
2. { }
3. ( )
4. < >

# Duplicate Rows

The DISTINCT keyword affects all the (multiple possible) selected columns, and the result is every distinct combination of the columns.

Example:

SELECT DISTINCT department\_id  
 From employees;

# Describe command

Use the describe command to display the structure of a table

DESCRIBE tablename;  
DESC tablename;

Data types are described in the following table:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| NUMBER(p,s) | Number value having a maximum number of a digits p, with s digits to the right of the decimal point |
| VARCHAR2(s) | Variable-length character value of maximum size s |
| DATE | Date and time value between January 1, 4712 B.C. and December 31, A.D. 9999 |
| CHAR(s) | Fixed-length character value of size s |

Hoofdstuk2: Restricting and Sorting Data

Data

You can restrict the rows that are returned from the query by using the WHERE clause.  
It contains a condition that must be met and it directly follows the FROM clause.

Example:

SELECT \*|{[DISTINCT] column|expression [alias],…} FROM table WHERE condition(s)]

In the syntax:

WHERE Restricts the query to rows that meet a condition  
 condition is composed of column names, expressions, constants, and a comparison operator. A condition specifies a combination of one or more expressions and logical (Boolean) operators, and returns a value of TRUE, FALSE, UNKNOWN.

Important notes:

1. Character strings and dates in the where clause are enclosed with single quotation marks.
2. Character values are case-sensitive and date values are format-sensitive
3. The default date display format is DD-MM-RR

Comparison operators:

|  |  |
| --- | --- |
| Operator | Meaning |
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| <> | Not equal to |
| BETWEEN (lower limit) AND (upper limit) | Between two values (inclusive) |
| IN(set) | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

An alias cannot be used in the WHERE clause.

To test for values in a specified set of values, use the IN operator, also known as membership operator.  
The IN operator can be used with any type of data, if it is used for characters or dates, they must be enclosed with single quotation marks (‘ ’)

Example:

SELECT employee\_id, last\_name, salary, manager\_id   
 FROM employees   
 WHERE manager\_id IN (100, 101, 201);

Use the LIKE operator to perform wildcard search of valid search string values:

1. % denotes zero or many characters.
2. \_ denotes one character.

You can select rows that match a character pattern, it is referred to as a wildcard search,  
The LIKE operator can be used as a shortcut for some BETWEEN comparisons.  
reminder: LIKE clause is case sensitive!

The % and \_ symbols can be used in any combination with literal characters. If you want to specify with those character, use the ESCAPE identifier.  
This option specifies what the escape character is. If you want to search for strings that contain SA\_, use this example:

SELECT employee\_id, last\_name, job\_id   
 FROM employees WHERE jod\_id LIKE ‘SA\\_%’ ESCAPE ‘\’;

The escape identifier identifies the \ as the escape character. This causes the Oracle server to interpret the underscore literally.

The NULL conditions include the IS NULL condition and the IS NOT NULL condition.  
A null value means that the value is unavailable, unassigned, unknown or inapplicable.

|  |  |
| --- | --- |
| Operator | Meaning |
| AND | Returns TRUE if both component conditions are true |
| OR | Returns TRUE is either component condition is true |
| NOT | Returns TRUE if the condition is false |

You can use several conditions in a single WHERE clause using the AND and OR operators.

The AND operator:

|  |  |  |  |
| --- | --- | --- | --- |
| AND | TRUE | FALSE | NULL |
| TRUE | TRUE | FALSE | NULL |
| FALSE | FALSE | FALSE | FALSE |
| NULL | NULL | FALSE | NULL |

All character searches are case-sensitive, that is no rows are returned if ‘MAN’ is not uppercase.

Example:

SELECT employee\_id, last\_name, job\_id, salary  
 FROM employees  
 WHERE salary >= 10000  
 AND job\_id LIKE ‘%MAN%’

The OR operator:

|  |  |  |  |
| --- | --- | --- | --- |
| OR | TRUE | FALSE | NULL |
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | NULL |
| NULL | TRUE | NULL | NULL |

Either component condition can be true for any record to be selected.

Example:

SELECT employee\_id, last\_name, job\_id, salary  
 FROM employees  
 WHERE salary >= 10000  
 OR job\_id LIKE ‘%MAN%’

The NOT operator:

|  |  |  |  |
| --- | --- | --- | --- |
| NOT | TRUE | FALSE | NULL |
|  | FALSE | TRUE | NULL |

Example:

SELECT last\_name, job\_id   
 FROM employees  
 WHERE job\_id  
 NOT IN (‘IT\_PROG’, ‘ST\_CLERK’, ‘SA\_REP’)

Rules of precedence:

|  |  |
| --- | --- |
| Operator | Meaning |
| 1 | Arithmetic operators |
| 2 | Concatenation operator |
| 3 | Comparison condition |
| 4 | IS [NOT] NULL, LIKE, [NOT] IN |
| 5 | [NOT] BETWEEN |
| 6 | Not equal to |
| 7 | NOT logical condition |
| 8 | AND logical condition |
| 9 | OR logical condition |

ORDER BY clase:

1. ASC: ascending order, default
2. DESC: Descending order

The ORDER BY clause comes last in the SELECT statement.  
Example:

SELECT last\_name, job\_id, department\_id, hire\_date   
 FROM employees  
 ORDER BY hire\_date

If the ORDER BY clause is not used, the sort order is undefined.  
You can also sort by a column that is not in the SELECT list.

Substituion variables:  
Use them in place of the exact values in the WHERE clause, you can run the same query for different values.  
You can embed substitution variables in a command file or in a single SQL-statement.  
You can use single ampersand (&) substitution variables to temporarily store values.

Using a variable prefixed with an ampersand to prompt the user for a value.  
Example:

SELECT employee\_id, last\_name, salary, department\_id  
 FROM employees  
 WHERE employee\_id = &employee\_num;

&user\_variable = Indicates a variable in a SQL statement; if the variable does not exist, SQL\*Plus or SQL Developer promps the user for a value (the new variable is discarded after it is used.)  
The user is prompted every time the command is executed if the variable does not exist.

Enclose the variable with single quotation marks within the SQL statement itself.

You can use double-ampersand (&&) substitution variable if you want to reuse the variable value without prompting the user each time.  
Example:

SELECT employee\_id, last\_name, job\_id, &&column\_name  
 FROM employees  
 ORDER BY &column\_name;

SQL DEVELOPER COMMANDS ONLY:

Use the DEFINE command to create and assign a value to a variable.  
Use UNDEFINE command to remove a variable.

If you use the DEFINE command to define a variable, the user is not prompted to enter a value, instead the defined variable value is automatically substituted in the SELECT statement.  
The substitution variable is present in the session until the user undefines it or exits the SQL Developer session.

Use the VERIFY command to toggle the display of the substitution variable, both before and after SQL Developer replaces substitution variables with values.

SET VERIFY ON 🡪 forces SQL Developer to display the text of a command after it replaces substitution variables with values.  
SQL Developer displays the text of a command after it replaces substitution variables with values;

Hoofdstuk3: Using single-row functions to customize output

Data

SQL functions are powerful, they can be used to:

* Perform calculations on data
* Modify individual data items
* Manipulate output for groups of rows
* Format dates and numbers for display
* Convert column data types

SQL functions sometimes take arguments and always return a value.

Types of functions:

1. Single-row functions
2. Multiple-row functions

These functions operate on single row and only return one result per row:

* Character
* Number
* Date
* Conversion
* General

Functions can manipulate groups of rows to give one result per group of rows = group functions

Single-row functions:

* Manipulate data items
* Accept arguments and return one value
* Act on each row that is returned
* Return one result per row
* May modify the data type
* Can be nested
* Accept arguments that can be a column or an expression

Examples of arguments:

* User-supplied constant
* Variable value
* Column name
* Expression

Features of single-row functions include:

* Acting on each row that is returned in the query
* Returning one result per row
* Possibly returning a data value of a different type than the one that is referenced
* Possibly expecting one or more arguments
* Can be used in SELECT, WHERE and ORDER BY clauses; can be nested

Syntax: function\_name[(arg1, arg2,…)]

Function\_name is the name of the function  
 arg1, arg2 is any argument to be used by the function. This can be represented by a column name or expression.

2 types of character functions:

1. Case-conversion functions (LOWER, UPPER, INITCAP..)
2. Character-manipulation functions (CONCAT, SUBSR, TRIM..)

Character function list:

**LOWER(column | expression)**   
 converts alpha character values to lowercase

**UPPER(column | expression)**   
 converts alpha character values to uppercase

**INITCAP(column | expression)**   
 converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase

**CONCAT(column | expression1, column | expression2)**   
 concatenates the first character value to the second character value; equivalent to concatenation operator ||

**SUBSTR(column | expression, m[,n])**   
 returns specified characters from character value starting at character position m, n c haracters long (if m is negative, the count starts from the end of the character value. If n is omitted, all characters to the end of the string are returned.

**LENGTH(column | expression)**  
 returns the number of characters in the expression

**INSTR(column | expression, ‘string’, [,m], [n])** returns the numeric position of a named string. Optionally, you can provide a position m o start searching and the occurrence n of the string, m and n default to 1, meaning start the search at the beginning of the string and report the first occurrence.

**LPAD(column | expression, n, ‘string’)  
RPAD(column | expression, n, ‘string’)** returns an expression left-padded/right-padded to the length of n characters with a character expression.

**TRIM(leading|trailing|both, trim\_character FROM trim\_source)** enables you to trim leading or trailing characters (or both) from a character string. If trim\_character or trim\_source is a character literal, you must enclose it in single quotation marks.

**REPLACE(text, search\_string, replacement\_string)** searches a text expression for a character string and, if found, replaces it with a specified replacement string

Number function list:

**ROUND(column | expression, n)** rounds the column, expression, or value to n decimal places or, if n is omitted, no decimal places (If n is negative, numbers to the left of the decimal point are rounded)

**TRUNC(column, expression, n)** truncates the column, expression, or value to n decimal places or, if n is omitted, n defaults to zero

**MOD(m, n)** returns the remainder of m divided by n

Dual table:  
The dual table is owned by the user SYS and can be accessed by all users. When you want to return a value only once, use the dual table, it is general used for completeness of the SELECT clause syntax.

RR date format:

|  |  |  |  |
| --- | --- | --- | --- |
| **Current year** | **Given date** | **Interpreted (RR)** | **Interpreted (YY)** |
| 1994 | 27-OCT-95 | 1995 | 1995 |
| 1994 | 27-OCT-17 | 2017 | 1917 |
| 2001 | 27-OCT-17 | 2017 | 2017 |

RR specifies the different centuries.

Arithmetic dates:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Result** | **Description** |
| Date + number | Date | Adds a number of days to a date |
| Date – number | Date | Substracts a number of day from a date |
| Date – date | Number of days | Substracts one date from another |
| Date + numner/24 | Date | Adds a number of hours to a date |

Date functions:

**SYSDATE** is a date function that returns the current database server date and time.

**MONTHS\_BETWEEN(date1, date2)** find the number of months between date1 and date2, result can be positive or negative.   
 The noninteger part of the result represents a portion of the month

**ADD\_MONTHS(date, n)** adds n number of calendar months to date. The value of n must be an integer and can be negative.

**NEXT\_DAY(date, ‘char’)** finds the date of the next specified day of the week (‘char’) following date. The value of char may be a number representing a day or a character string.

**LAST\_DAT(date)** finds the date of the last day of the month that contains date.

**ROUND(date[, ‘fmt])** returns date rounded to the unit that is specified by the format model fmt, if the format fmt is omitted, date is rounded to the nearest day

**TRUNC(date, [, ‘fmt’])** returns date with the time portion of the day truncated to the unit that is specified by the format model fmt. If the format model fmt is omitted, date is truncated to the nearest day.

Example trunc/round: Assume SYSDATE = ’25-JUL-03’

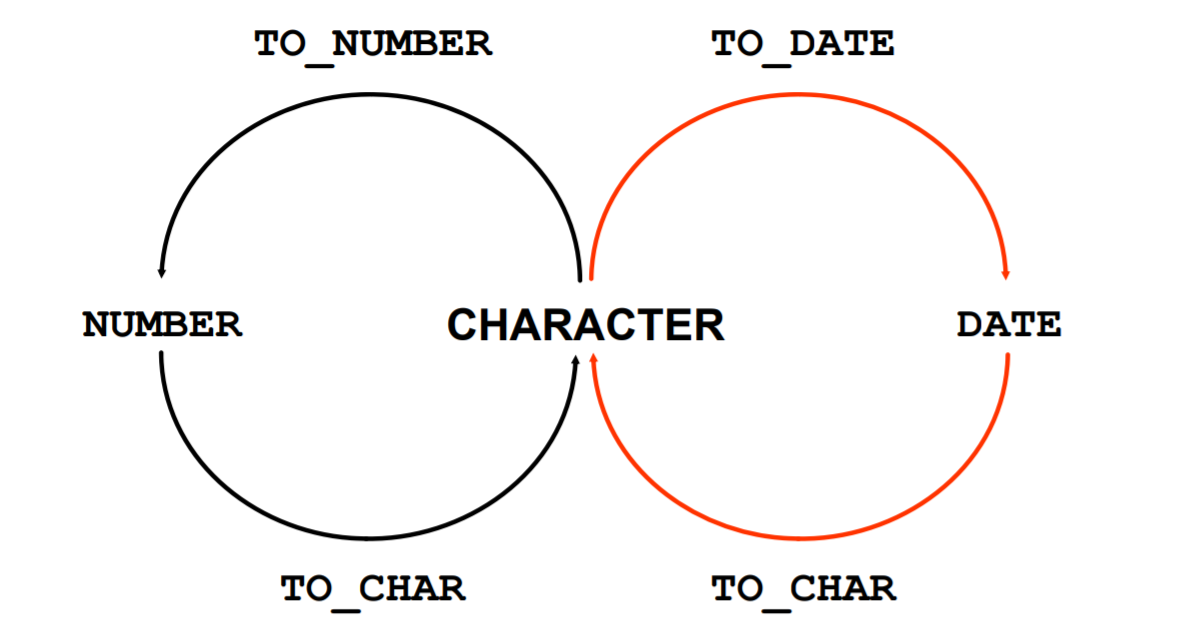
|  |  |
| --- | --- |
| **Function** | **Result** |
| ROUND(SYSDATE, ‘MONTH’) | 01-AUG-03 |
| ROUND(SYSDATE, ‘YEAR’) | 01-JAN-04 |
| TRUNC(SYSDATE, ‘MONTH’) | 01-JUL-03 |
| TRUNC(SYSDATE, ‘YEAR’) | 01-JAN-03 |

Hoofdstuk4: Using conversion functions and conditional expressions

Data

If the Oracle server receives data of one data type where it expects data of a different data type, it can automatically convert the data to the expected data type.  
Explicit data type conversions are done by using the conversion functions.

VARCHAR2 and CHAR values can be implicitly converted to a number or date data type in an expression.



|  |  |
| --- | --- |
| **Function** | **Purpose** |
| TO\_CHAR(number|date, [fmt],[nlsparams]) | Converts a number or date value to a VARCHAR character string with the format model fmt **Number conversion:** the nlsparams parameter specifies the following characters, which are returned by number format elements:   1. Decimal character 2. Group separator 3. Local currency symbol 4. International currency symbol   If nlsparams or any other parameter Is omitted, this function uses the default parameter values for the session **Date conversion:** the nlsparams parameter specifies the language in which the month day names and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session. |

|  |  |
| --- | --- |
| TO\_NUMBER(char, [fmt], [nlsparams]) | Converts a character string containing digits to a number in the format specified by the optional format model fmt.  The nlsparams parameter has the same purpose in this function as in the TO\_CHAR fuction for number conversion |
| TO\_DATE(char, [fmt], [nlsparams]) | Converts a character string representing a date to a date value according to the fmt that is specified. If fmt is omitted, the format is DD-MON-YY The nlsparams parameter has the same purpose in this function as in the TO\_CHAR function for date conversion |

# Using TO\_CHAR function with dates

The format model:

* Must be enclosed with single quotation marks
* Is case-sensitive
* Can include any valid date format element
* Has an fm element to remove padded blanks or suppress leading zeros
* Is separated from the date value by a comma

A format model is a character literal that describes the format of datetime stored in a character string.

## Elements of the date format model

|  |  |
| --- | --- |
| **Element** | **Result** |
| YYYY | Full year in numbers |
| YEAR | Year spelled out (in English) |
| MM | Two-digit value for the month |
| MONTH | Full name of the month |
| MON | Three-letter abbreviation of the month |
| DY | Three-letter abbreviation of the day of the week |
| DAY | Full name of the day of the week |
| DD | Numeric day of the month |

## Advanced element of the date format model

|  |  |
| --- | --- |
| **Element** | **Description** |
| SCC or CC | Century, server prefixes B.C date with - |
| Years in dates YYYY or SYYYY | Year, server prefixes B.C. date with - |
| YYY or YY or Y | Last three, two or one digit of the year |
| Y, YYY | Year with comma in this position |
| IYYY, IYY, IY, I | Four-, three-, two-, or one-digit year based on the ISO standard |
| SYEAR or YEAR | Year spelled out; server prefixed B.C. date with- |
| BC or AD | Indicates B.C. or A.D. year using periods |
| Q | Quarter of year |
| RM | Roman numeral month |
| WW or W | Week of the year or month |
| DDD or DD or D | Day of the year, month, or week |
| DY | Name of the day; thee-letter abbreviation |
| J | Julian day; the number of days since December 31, 4713 B.V. |
| IW | Weeks in the year from ISO standard |
| AM or PM | Meridian indicator |
| A.M. or P.M. | Meridian indicator with periods |
| HH or HH12 or HH24 | Hours of day, or hour (1-12) or hour (0-23) |
| MI | Minutes (0-59) |
| SS | Seconds (0-59) |
| SSSSS | Seconds past midnight(0-86399) |
| / . , | Punctuation is reproduced in the result |
| “of the” | Quoted string is reproduced in the result |
| TH | Ordinal number |
| SP | Spelled out number |
| SPTH or THSP | Spelled-our ordinal number |

## Using TO\_CHAR function with numbers

TO\_CHAR(number, ‘format\_model’)

|  |  |
| --- | --- |
| **Element** | **Result** |
| 9 | Represents a number |
| 0 | Forces a zero to be displayed |
| $ | Places a floating dollar sign |
| L | Uses the floating local currency symbol |
| . | Prints a decimal point |
| , | Prints a comma as a thousands indicator |
| D | Returns the decimal character in the specified position. The default is a period |
| G | Returns the group separator in the specified position. You can specify multiple group separators in a number format model |
| MI | Minus signs to right |
| PR | Parenthesize negative numbers |
| EEEE | Scientific notation |
| U | Returns in the specified position in the ‘Euro” dual currency |
| V | Multiply by 10 n times (n = number after 9s after V) |
| S | Returns the negative or positive value |
| B | Displays zero values as blank, not 0 |

## TO\_NUMBER and TO\_DATE

TO\_NUMER(char[, ‘format\_model’])  
 🡪Converts a character string to a number format

TO\_DATE(char[, ‘format\_model’])  
 🡪Converts a character string to a date format

These function have an fx modifier. This modifier specifies the exact match for the character argument and date format model of a TO\_DATE function

The fx modifier specifies the exact match for the character argument and date format model of a TO\_DATE function:

* Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model
* The character argument cannot have extra blanks. Without fx, the oracle server ignores extra blanks
* Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, the number in the character argument can omit leading zeros.

Single row functions can be nested to any dept. Nested functions are evaluated from the innermost level to the outermost level

# General functions

**NVL(expr1, expr2)**  
 converts a null value to an actual value  
 expr1 = is the source value or expression that may contain a null  
 expr2 = is the target value for converting null

**NVL2(expr1, expr2, expr3)** If expr1 is not null, NVL2 returns expr2, if expr1 is null, NVL2 returns expr3, the argument expr1 can have any data type  
 expr1 = is the source value or expression that may contain a null  
 expr2 = is the value that is returned if expr1 is not null  
 expr3 = is the value that is returned if expr1 is null

**NULLIF(expr1, expr2)** Compares two expressions and returns null if they are equal; return the first expression if they are not equal, you cannot specify the literal NULL for the first expression.

**COALESCE(expr1, expr2, …, exprn)** Returns the first non-null expression in the expression list  
 expr1 = returns this expression if it is not null  
 expr2 = returns this expression is the first expression is null and this expression is not null  
 expr3 = returns this expression if the preceding expression are null

# CASE expressions

In a simple CASE expression, the oracle server searches for the first WHEN … THEN pair for which expr is equal to copmparison\_expr and returns return\_expr.  
If none of the WHEN … THEN pairs meet this condition, and if an ELSE clause exists, then the oracle server returns else\_expr. Otherwise the oracle server returns a null. You cannot specify the literal NULL for all return\_exprs and the else\_expr

CASE expr WHEN comparison\_expr1 THEN return\_expr1  
 [WHEN comparison\_expr2 THEN return\_expr2  
 WHEN comparison\_exprn THEN return\_exprn  
 ELSE else\_expr]

END

The search occurs from left to right until an occurrence of the listed condition is found and then returns the return expression. If an ELSE clause exists, the return expression in the else clause is returned, otherwise, a NULL is returned

## DECODE function

DECODE(col|expression, search1, result1  
 [, search2, result2, …..,]  
 [, default])

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic that is used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.

Hoofdstuk6: Displaying Data from Multiple Tables

Data

Types of joints:

1. Natural joins
   1. NATURAL JOIN clause
   2. USING clause
   3. ON clause
2. OUTER joins
   1. LEFT OUTER JOIN
   2. RIGHT OUTER JOIN
   3. FULL OUTER JOIN
3. Cross joins

Example:

SELECT table1.column, table2.column  
 FROM table1  
 [NATURAL JOIN table2} |  
 [JOIN table2  
 ON (table1.column\_name = table2.column\_name)]|  
 [LEFT|RIGHT|FULL OUTER JOIN table2   
 ON (table1.column\_name = table2.column\_name)]|  
 [CROSS JOIN table2];

In the syntax:

Table1.column = denotes the table and the column from which data is retrieved  
 NATURAL JOIN = joins two tables based on the same column name  
 JOIN table2 USING column\_name = performs an equijoin based on the column name  
 JOIN table2 ON table1.column\_name = table2.column\_name = performs an equijoin based on the condition in the ON clause  
 LEFT/RIGHT/FULL OUTER = is used to perform OUTER join  
 CROS JOIN = returns a Cartesian product from the two tables

Qualifying ambiguous column names:

* Use table prefixed to qualify column names that are in multiple tables.
* Use table prefixed to improve performance
* Instead of full table name prefixes, use table aliases
* Table alias gives a table a shorter name:
  + Keeps sql code smaller, uses less memory
* Use column aliases to distinguish columns that have identical names, but reside in different tables

Guidelines:

* Table aliases can be up to 30 characters in length, but shorter are better
* If a table alias is used for a particular table name in the FROM clause, then that table alias must be substituted for the table name throughout the SELECT statement
* Table aliases should be meaningful
* The table alias is valid for only the current SELECT statement

## Creating natural joins

* The NATURAL JOIN clause is baes on all columns in the two tables that have the same name
* It selects rows from the two tables that have equal values in all matched columns
* If the columns having the same names have different data types, an error is returned

## Creating joins with the USING clause

* If several columns have the same names but the data types do not match, use the USING clause to specify the columns for the equijoin
* Use the USING clause to match only one column when more than one column matches
* The NATURAL JOIN and USING clauses are mutually exclusive
* The USING clause can be best used to specify only those columns that should be used for an equijoin

(equijoins are also called simple joins or inner joins)

Example:

SELECT employee\_id, last\_name, location\_id, department\_id  
 FROM employees JOIN departments  
 USING (department\_id)

Using table aliases with the USING clause example:

SELECT l.city, d.department\_name  
 FROM locations l JOIN departments d  
 USING (location\_id)  
 WHERE d.location\_id = 1400;

## Creating joins with the ON clause:

* The join condition for the natural join is basically an equijoin of all columns with the same name
* Use the ON clause to specify arbitrary conditions or specify columns to join
* The join condition is separated from other search conditions
* The ON clause makes code easy to understond

Retrieving records with the ON clause example:

SELECT e.employee\_id, e.last\_name, e.department\_id, d.department\_id, d.location\_id  
 FROM employees e JOIN departments d  
 ON (e.department\_id = d.department\_id)

You can also use the ON clause to join columns that have different names.

## Creating three-way joins with the ON clause = join of three tables

Example:

SELECT employee\_id, city, department\_name  
 FROM employees e  
 JOIN departments d  
 ON d.department\_id = e.department)id  
 JOIN locations l  
 ON d.location\_id = l.location\_id;

## SELF joining table

Sometimes you need to join a table to itself.  
Example: to find the name of each employee-s manager, ou need to join the employees table to itself, or perform a self-join. To find the name of Lorentz’s manager, you need to:

* Find Lorentz in the employees table by looking at the last\_name column
* Find the managers number of Lorentz
* Find the name of the manager with employee\_id

SELECT worker.last\_name emp, manager.last\_name mgr   
 FROM emploees worker JOIN employees manager  
 ON (worker.manager\_id = manager.employee\_id)

The ON clause can slo be used to join columns that have different names, within the same table or in a different table.

# Nonequijoins

A nonequijoin is a join condition containing something other than an equality operator

Example:

SELECT e.last\_name, e.salary, j.grade\_level  
 FROM employees e JOIN job\_grades j  
 ON e.salary  
 BETWEEN j.lowest\_sal AND j.highest\_sal;  
  
This example creates a nonequijoin to evaluate an employee’s salary grade. The salary must be between any pair of the low and high salary ranges

If a row does not satisfy a join condition, the row does not appear in the query result

# INNER versus OUTER joins

Joining tables with the NATURAL JOIN, USING or ON clauses results in an INNER join. Any unmatched rows are not displayed in the output. To return he unmatched rows, you can use an OUTER join.. it returns all rows that satisfy the join condition and also return some or all of those rows from one table for which no rows from the other table satisfy the join condition  
Types OUTER joins:

1. LEFT OUTER
2. RIGHT OUTER
3. FULL OUTER

## LEFT OUTER JOIN

Example:

SELECT e.last\_name, e.department\_id, d.department\_name  
 FROM employees e LEFT OUTER JOIN departments d  
 ON (e.department\_id = d.department\_id);

This query retrieves all the rows in the EMPLOYEES table, which is the left able, even if there is no match in the DEPARTMENTS table

## RIGHT OUTER JOIN

Example:

SELECT e.last\_name, e.department\_id, d.department\_name  
 FROM employees e LEFT OUTER JOIN departments d  
 ON (e.department\_id = d.department\_id);

This query retrieves all the rows in the DEPARTMENTS table, which is the table at the right, even if there is not match in the EMPLOYEES table

## FULL OUTER JOIN

Example:

SELECT e.last\_name, e.department\_id, d.department\_name  
 FROM employees e LEFT OUTER JOIN departments d  
 ON (e.department\_id = d.department\_id);

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMETNS table. It also retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

# Cartesian Products

Is formed when:

* A join is omitted
* A join condition is invalid
* All rows in the first table are joined to all rows in the second table

To avoid a cartesian product, always include a valid join condition. Cartesian is rarely usefull  
A cartesian product is generated if a join condition is omitted

The CROSS JOIN lause produces the cross-product of two table  
This is also called Cartesian product between the two tables  
Example:

SELECT last\_name, department\_name  
 FROM employees  
 CROSS JOIN departments

The example produces a cartesian product of the EMPLOYEES and DEPARTMENTS tables

Hoofdstuk9: Data manipulation

Data

A DML statement is executed when:

* Adds new rows to a table
* Modify existing rows in a table
* Remove existing rows from a table

A transaction consists of a collection of DML statements that form a logical unit of work

# INSERT statement

Add new rows to a table by using the INSERT statement

Example:

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

Table is the name of the table  
column is the name of the column in the table to populate  
value is the corresponding value for the column

Insert a new row containing values for each column  
List values in the default order of the columns in the table  
Enclose character and date values within single quotation marks

Inserting rows with Null values:

1. Implicit method: omit the column from the column list:  
   example:  
    INSERT INTO departments (department\_id, department\_name)  
    VALUES(30, ‘purchasing’);
2. Explicit method: specify the NULL keyword in te VALUES clause  
   example:  
    INSET INTO departments  
    VALUES(100, ‘finance’, NULL, NULL);

Date functions and substitution variables can be used in INSERT statements as well

## Copying rows from another table

* Write your INSERT statement with a subquery
* Do not use the VALUES clause
* Match the number of columns in the INSERT clause to those in the subquery
* Inserts all the rows returned by the subquery in the table

INSERT INTO table [column (, column)] subquery;

Table = name of the table  
column = name of the column in the table to populate  
subquery = the subquery that returns rows to the table

# Changing data in a table

## UPDATE statement

* Modify existing values in a table with the UPDATE statement  
  Example:  
   UPDATE table  
   SET column = value [, column = value, …]  
   WHERE condition;
* Update more than one row at a time (if required)
* Values for a specific row or rows are modified if you specify the WHERE clause
* Values for all the rows in the table are modified if you omit the WHERE clause
* Specify SET column\_name = NULL to update a column value to NULL

## Update two columns with a subquery

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries.  
Example:

UPDATE table  
 SET column = (SELECT column  
 FROM table  
 WHERE condition)  
 [,column = (SELECT column  
 FROM table  
 WHERE condition)]  
[WHERE condition] ;

Use the subqueries in the UPDATE statements to update row values in a table based on values from another table.

# Removing rows from a table

## DELETE statement

You a remove existing rows from a table by using the DELETE statement:  
 DELETE [FROM] table  
 [WHERE condition];

* Specific rows are deleted if you specify the WHERE clause:  
   DELETE FROM departments  
   WHERE department\_name = ‘finance’;
* If you don’t specify, all rows are deleted
* Use subqueries in the DELETE statements to remove rows from a table based on values from another table:  
   DELETE FROM employees  
   WHERE department\_id =   
   (SELECT department\_id  
   FROM departments  
   WHERE department\_name  
   LIKE ‘%Public%’)

## TRUNCATE statement

* Removes all rows from a table, leaving the table empty and the table structure intact
* Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
* Syntax:  
   TRUNCATE TABLE table\_name;

# Database transactions control using COMMIT, ROLLBACK and SAVEPOINT

The oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

|  |  |
| --- | --- |
| **Type** | **Description** |
| Data manipulation language (DML) | Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work |
| Data definition language (DDL) | Consists of only one DDL statement |
| Data control language (DCL) | Consists of only one DCL statement |

A DML statement is encountered and ends when one of the following occurs:

* A COMMIT or ROLLBACK statement is issued
* A DDL statement, such as CREATE, is issued
* A DCL statement is issued
* The user exists SQL Developer or SQL \*Plus
* A machine fails or the system crashes

After one transaction ends, the next executable SQL statement automatically starts the next transaction. A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

## Advantages of COMMIT and ROLLBACK statements

* Ensure data consistency
* Preview data changes before making changes permament
* Group logically-related operations

|  |  |
| --- | --- |
| **Statements** | **Description** |
| COMMIT | Ends the current transaction by making all pending data changes permanent |
| SAVEPOINT name | Marks a savepoint within the current transaction |
| ROLLBACK | Ends the current transaction by discarding all pending data changes |
| ROLLBACK TO SAVEPOINT name | Rolls back the current transaction to the specified point |

* Create a marker in the current transaction by using the SAVEPOINT statement  
   UPDATE…  
   SAVEPOINT update\_done;
* Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement  
   INSERT…  
   ROLLBACK TO update\_done;

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back.

## State of the data before COMMIT and ROLLBACK

* The previous state of the data can be recovered
* The current user can review the results of the DML operations by using the SELECT statement
* Other users cannot view the results of the DML statements issued by the current user
* The affected rows are locked; other users cannot change the data in the affected rows.
* To commit changes, just do COMMIT;

Discard all pending changes by using the ROLLBACK statement:

* Data changes are undone
* Previous state of data is restored
* Locks on the affected rows are released
* If a single DML statement fails during execution, only that statement is rolled back
* The oracle server implements an implicit savepoint
* All other changes are retained
* The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.

# Read consistency

Database users access the database in two ways:

1. Read operations (SELECT statements)
2. Write operations (INSERT, UPDATE, DELETE statements)

You need read consistency so that the following occur:

* The database reader and writer are ensured a consistent view of the data
* Readers do not view data that is in the process of being changes
* Writers are ensured that the changes to the database are done in a consistent manner
* Changes made by one writer do not disrupt or conflict with the changes being made by another writer.

# FOR UPDATE clause

You can use the FOR UPDATE of column\_name to qualify the column that you intend to change.  
The OF list of the FOR UPDATE clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state FOR UPDATE in the query and do not include one or more columns after the OF keyword, the database will lock all identified rows across all the tables listed in the FROM clause.

Hoofdstuk10: Using DDL statements to create and manage tables

# Database objects

The oracle database can contain multiple data structures. Each structure should be outlined in the database design so it can be created during the build stage of database development:

* **Table:** stores data
* **View:** subset of data from one or more tables
* **Sequence:** generates numeric values
* **Index:** improves the performance of some queries
* **Synonym:** gives alternative name to an object

**Oracle database structures:**

* Tables can be created at any time, even when users are using the database
* You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole
* Table structured can be modified online

Table names and column names:

1. Must begin with a letter
2. Must be 1-30 characters long
3. Must contain only A-Z, a-z, 0-9, \_, $ and #
4. Must not duplicate the name of another object owned by the same user
5. Must not be an oracle server-reserved word
6. Names are not case-sensitive

# CREATE TABLE statement

To create a table, a user must have the CREATE TABLE privilege and a storage are in which to create object. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.  
Example:

CREATE TABLE [schemas.] table  
 (column datatype [DEFAULT expr] [, …])

Schema is the same as the owner’s name  
table is the name of the table  
DEFAULT expr specifies a default value if a value is omitted in the INSERT statement  
column Is the name of the column  
datatype is the column’s data type and length

A schema is a collection of logical structures of data or schema objects. It is owned by a c=database user and has the same name as that user.  
SELECT \* FROM userb.employees

## DEFAULT option:

Specify a default value for a column during an insert

… hire\_date DATE DEFAULT SYSDATE, …

Literal values, expressions or SQL-functions are legal values  
Another column’s name or a pseudocolumn are illegal values  
The default data type must match the column data type

Confirm table creation with:  
DESCRIBE table;

# Data types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| VARCHAR2(size) | Variable-length character data ( a maximum size must be specified: minimum size is 1, maximum size is 4000 |
| CHAR [(size)] | Fixed-length character data of length size bytes (default and minimum size is 1, max size is 2000) |
| NUMBER[(p,s)] | Number having precision p and scale s (precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38 and scale can range from -84 to 127 |
| DATE | Date and time values to the nearest second between January 1 4712 B.C. and December 31, 9999 A.D. |
| LONG | Variable length character data (up to 2GB) |
| CLOB | Character data (up to 4GB) |
| RAW(size) | Raw binary data of length size (a maximum size must be specified: maximum size is 2000) |
| LONG RAW | Raw binary data of variable length (up to 2GB) |
| BLOB | Binary data (up to 4GB) |
| BFILE | Binary data stored in an external file (up to 4GB) |
| ROWID | A base-64 number system representing the unique address of a row in its table |

Guidelines:

1. A LONG column is not copied when a table is created using a subquery
2. A LONG column cannot be included in a GROUP BY or ORDER BY clause
3. Only one LONG column can be used per table
4. No constraints can be defined on a LONG column
5. You might want to use a CLOB column rather than a LONG column

## Datetime data types

|  |  |
| --- | --- |
| **Data type** | **Description** |
| TIMESTAMP | Date with fractional seconds |
| INTERVAL YEAR TO MONTH | Stored as an interval of years and months |
| INTERVAL DAY TO SECOND | Stored as an interval of days, hours, minutes and seconds |

# Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints

## Including constraint

* Constraints enforce rules at the table leven
* Constraint prevent the deletion of a table if there are dependencies
* The following constraint types are valid:
  + NOT NULL
  + UNIQUE
  + PRIMARY KEY
  + FOREING KEY
  + CHECK

Constraints are easy to reference if you give them a meaningful name. the names must follow the standard object-naming rules. Except it cannot be the same as another object owned by the same user. If you don’t oracle will random do things and make it harder for you, just name constraints idiot.

## Defining constraints

You can create constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. It must refer to the column or columns on which the constraint pertain in a set of parentheses. Otherwise, functionally a column-level constraint is the same as a table-level constraint .  
NOT NULL constraints must be defined at the column level

Example of column-level constraint:

CREATE TABLE employees(  
 employee\_id NUMBER(6)   
 CONSTRAINT emp\_emp\_id\_pk PRIMARY KEY,  
 first\_name VARCHAR2(20),  
 …);

Example of a table-level constraint:

CREATE TABLE employees(  
 employee\_id NUMBER(6),  
 first\_name VARCHAR2(20),  
 …  
 job\_id VARCHAR2(10) NOT NULL,  
 CONSTRAINT emp\_emp\_id\_pk  
 PRIMARY KEY (EMPLOYEE\_ID));

## UNIQUE constraint

A UNIQUE key integrity constraint requires that every value in a column or a set of columns (keys\_ be unique.  
The UNIQUE key constraint is called the unique key. If the UNIQUE constraint comprises more than one column, that group of columns is called a composite unique key.  
Unique constraints can be defined at column level or table level. You define the constraint at the table level when you want to create a composite unique key.

## PRIMARY KEY constraint

A PRIMAR KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or a set of columns that uniquely identifies each row in a table.

## FOREIGN KEY constraint

The FOREIGN KEY constraint (or referential integrity) constraint designates a column or a combination of columns as a foreign key and establishes a relationship with a primary key or a unique key in the same table or a different table.  
Guidelines:

1. A foreign key value must match an existing value in the parent table or be NULL
2. Foreign keys are based on data values and are purely logical, rather than physical, pointers.

A composite foreign key must be created using the table-level definition.

FOREIGN KEY constraint keywords:

* **FOREIGN KEY:** defines the column in the child table at the table-constraint level
* **REFERENCES:** Identifies the table and column in the parent table
* **ON DELETE CASCADE:** deletes the dependent rows in the child table when a row in the parent table is deleted
* **ON DELETE SET NULL:** converts dependent foreign key values to null

## CHECK constraint

Defines a condition that each row must satisfy  
The following expressions are not allowed:

* References to CURRVAL, NEXTVAL, LEVEL and ROWNUM pseudocolumns
* Calls to SYSDATE, UID, USER, and USERENV functions
* Queries that refer to other values in the rows

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column

## Violating constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

DELETE FROM departments  
 WHERE department\_id = 60;

# Creating tables in a subquery

Create a table and insert rows by combining the CREATE TABLE statement and the AS subquery option  
 CREATE TABLE table   
 [(column, column…)]  
 AS subquery

Match the number of specified columns to the number of subquery columns  
Define columns with column names and default values.

Guidelines:

* The table is created with the specifies column names, and the rows retrieved by the select statement are inserted into the table
* The column definition can contain only the column name and default value
* If column specifications are given, the number of columns must be equal to the number of columns in the subquery SELECT list
* If no column specifications are given, the column names of the table are the same as the column names in the subquery
* The column data type definitions and the NOT NULL constraint are passed to the new table.

# ALTER TABLE

Use the ALTER TABLE statement to:

* Add a new column
* Modify an existing column definition
* Define a default value for the new column
* Drop a column
* Rename a column
* Change table to read-only status

## Read-only tables

You can use the ALTER TABLE syntax to:  
 1) put a table into read-only mode, which prevents DDL or DML changes during table maintenance.  
 2) put the table ack into read/write mode

ALTER TABLE table READ ONLY;  
--- perform table maintenance and then  
--- return table back to read/write mode  
ALTER TABLE table READ WRITE:

You can specify READ ONLY to place a table in the read-only mode, you cannot issue any DML statements that affect the table or any SELECT …. FOR UPDATE statements.  
You cannot modify any data in the table  
Specify READ/WRITE to return a read-only table to the read/write mode

You can drop a table that is in READ ONLY mode. The DROP command is executed only in the data directory, so access to the table contents is not required

# DROP TABLE statement

* Moves a table to the recycle bin
* Removes the table and all its data entirely if the PURGE clause is specified
* Invalidates dependent objects and removes object privileges on the table

DROP TABLE table PURGE;

Guidelines:

* All the data is deleted from the table
* Any views and synonyms remain, but are invalid
* Any pending transactions are omitted
* Only the creator of the table or a user with the DROP ANY TABLE privilege can remove a table.

Use the FLASHBACK TABLE statement to restore a dropped table from the recycle bin.s