Leonardo's SQL Cheat Sheet

Operators

Comparison operator	What does it mean?
=	Equal to
<>	Not equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE '%expression%'	Contains 'expression'
IN ('exp1', 'exp2', 'exp3')	Contains any of 'exp1', 'exp2', or 'exp3'

Data Dictionary

```
ROLE_SYS_PRIVS | ROLE_TAB_PRIVS |
USER_ROLE_PRIVS | USER_TAB_PRIVS_MADE
USER_TAB_PRIVS_RECD | USER_COL_PRIVS_MADE
USER_COL_PRIVS_RECD
```

Object Privileges

```
Object     Table View Sequence Procedure
ALTER        X          X
DELETE       X     X
EXECUTE                             X
INDEX        X
INSERT       X     X
REFERENCES   X
SELECT       X     X     X
UPDATE       X     X
```

Conversion Functions

```
TO_CHAR(number|date[,'fmt'])
TO_NUMBER(char[,'fmt'])
TO_DATE(char[,'fmt'])
NVL(expr1,expr2)
DECODE(col/expr,search1,result1
                  [,search2,result2,...,]
                  [,default])
```

Group Functions

```
AVG([DISTINCT|ALL]n)
COUNT(*|[DISTINCT|ALL]expr)
MAX([DISTINCT|ALL]expr)
MIN([DISTINCT|ALL]expr)
STDDEV([DISTINCT|ALL]n)
SUM([DISTINCT|ALL]n)
VARIANCE([DISTINCT|ALL]n)
```

Date Functions

```
MONTHS_BETWEEN(date1,date2)
ADD_MONTHS(date,n)
NEXT_DAY(date,'char')
LAST_DAY(date)
ROUND(date[,'fmt'])
TRUNC(date[,'fmt'])
Number Functions
MOD(m,n)
ROUND(column|expression,n)
TRUNC(column|expression,n)
SYSDATE();
```

Character Functions

```
LOWER(column|expression)
UPPER(column|expression)
INITCAP(column|expression)
INSTR(column|expression,m)
CONCAT(column1|expression1,column2|expression2}
SUBSTR(column|expression,m,[n])
LENGTH(column|expression)
LPAD(column|expression,n,'string')
```

Null Functions

```
NVL(test_value, if_null)
NVL2(test_value, if_not_null, if_null)
COALESCE (expr1, expr2, ..., exprn)
NULLIF (Expr1, expr2)
```

SQL REGEX

match_option:

```
- " c ":  Uses case-sensitive matching (default)
- " i ":  Uses non-case-sensitive matching
- " n ":  Allows match-any-character operator
- " m ":  Treats source string as multiple lines
```

```
Character Class Syntax        Meaning
[:alnum:] All alphanumeric characters
[:alpha:] All alphabetic characters
[:blank:] All blank space characters.
[:cntrl:] All control characters (nonprinting)
[:digit:] All numeric digits
[:graph:] All {[:punct:], [:upper:], [:lower:],
and [:digit:]} characters.
[:lower:] All lowercase alphabetic characters
[:print:] All printable characters
[:punct:] All punctuation characters
[:space:] All space characters (nonprinting)
[:upper:] All uppercase alphabetic characters
[:xdigit:]All valid hexadecimal characters
```

```
Metacharacter       Description
.         Find a single character, except
newline or line terminator

\d        Find a digit
\D        Find a non-digit character
\s        Find a whitespace character
\S        Find a non-whitespace character

[a..c]    Find any character since a to c
[^abc]    Find any character NOT between the
brackets
```

Syntax	Description
.	Matches any character in the supported character set, except NULL
+	Matches one or more occurrences
?	Matches zero or one occurrence
*	Matches zero or more occurrences of the preceding subexpression
{m}	Matches exactly *m* occurrences of the preceding expression
{m, }	Matches at least *m* occurrences of the preceding subexpression
{m,n}	Matches at least *m*, but not more than *n*, occurrences of the preceding subexpression
[…]	Matches any single character in the list within the brackets
|	Matches one of the alternatives
(...)	Treats the enclosed expression within the parentheses as a unit. The subexpression can be a string of literals or a complex expression containing operators.
^	Matches the beginning of a string
$	Matches the end of a string
\	Treats the subsequent metacharacter in the expression as a literal
\n	Matches the *n*th (1–9) preceding subexpression of whatever is grouped within parentheses. The parentheses cause an expression to be remembered; a backreference refers to it.
\d	A digit character
[:class:]	Matches any character belonging to the specified POSIX character class
[^:class:]	Matches any single character *not* in the list within the brackets

```
REGEXP_LIKE (source_string, pattern [,
match_parameter] )
REGEXP_INSTR (
  source_string, pattern [, position [,
occurrence
  [, return_option [, match_parameter [,
sub_expression ] ] ] ] ]
)
REGEXP_REPLACE (source_string, pattern
[, replace_string [, position [, occurrence [,
match_parameter ] ] ] ]
)
REGEXP_SUBSTR (
source_string, pattern [, position [, occurrence
[, match_parameter ] ] ]
)
REGEXP_COUNT (source_string, pattern [, position
[, match_parameter ] ] ] )
```

Select Statement

```
SELECT [DISNCT] {*, column [alias],...}
 FROM table
 [WHERE condition(s)]
 [ORDER BY {column, exp, alias} [ASC|DESC]]
```

Manipulating Data

```
CREATE TABLE [schema.]table
(column datatype [DEFAULT expr] [,...]) ;
INSERT Statement(one row)
INSERT INTO table [ (column [,column...])]
 VALUES (value [,value...]) ;
ALTER TABLE table
 ADD (column datatype [DEFAULT expr]
[, column datatype]...) ;
```

SQL Statements

CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control language (TCL)

Conditions

```
IF condition THEN
   Statement;
END IF;
IF condition THEN
   Statement1;
ELSE
   Statement2;
END IF;
IF(boolean_expression 1)THEN
   Statement1; -- Executes when the boolean
expression 1 is true
ELSIF( boolean_expression 2) THEN
   Statement2;  -- Executes when the boolean
expression 2 is true
ELSIF( boolean_expression 3) THEN
   Statement3; -- Executes when the boolean
expression 3 is true
ELSE
   Statement4; -- executes when the none of the
above condition is true
END IF;
CASE selector
   WHEN 'value1' THEN Statement1;
   WHEN 'value2' THEN Statement2;
   WHEN 'value3' THEN Statement3;
   ...
   ELSE Sn;  -- default case
END CASE;
CASE
   WHEN selector = 'value1' THEN Statement1;
   WHEN selector = 'value2' THEN Statement2;
   WHEN selector = 'value3' THEN Statement3;
   ...
   ELSE Sn;  -- default case
END CASE;
IF( boolean_expression 1)THEN
   -- executes when the boolean expression 1 is
true
   IF(boolean_expression 2) THEN
      -- executes when the boolean expression 2
is true
      sequence-of-statements;
   END IF;
ELSE
   -- executes when the boolean expression 1 is
not true
   else-statements;
END IF;
```

Function

```
CREATE OR REPLACE FUNCTION
function_name
(parameter_1 data_type,
Parameter_2 data_type)
RETURN data_type
{ IS | AS }
[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [function_name];
```

Procedures

```
Create [ or REPLACE ]
PROCEDURE procedure_name
(
parameter_name_1
data_type,
parameter_name_2 data_type
)
{ IS | AS }
END;
```

Loops

```
LOOP
   Sequence of statements;
END LOOP;
WHILE condition LOOP
   sequence_of_statements
END LOOP;
FOR counter IN initial_value .. final_value LOOP
   sequence_of_statements;
END LOOP;
LOOP
   Sequence of statements1
   LOOP
      Sequence of statements2
   END LOOP;
END LOOP;
FOR counter1 IN initial_value1 .. final_value1
LOOP
   sequence_of_statements1
   FOR counter2 IN initial_value2 ..
final_value2 LOOP
      sequence_of_statements2
   END LOOP;
END LOOP;
WHILE condition1 LOOP
   sequence_of_statements1
   WHILE condition2 LOOP
      sequence_of_statements2
   END LOOP;
END LOOP;
```

CURSORS

```
CURSOR cursor_name
IS
  SELECT_statement;
CURSOR cursor_name (parameter_list)
IS
  SELECT_statement;
CURSOR cursor_name
RETURN field%ROWTYPE
IS
   SELECT_statement;
FOR record_name IN cursor_name LOOP
    statement1;
    statement2;
. . .
END LOOP;
```

Indexing Tables of Records

```
DECLARE
  TYPE type_name IS TABLE OF DATA_TYPE
    INDEX BY PRIMARY_KEY_DATA_TYPE;
  identifier type_name;
BEGIN
   FOR record IN (SELECT column FROM table)
   LOOP
    identifier(primary_key):= record.column;
   END LOOP;
END;
```

Trapping Exceptions

```
EXCEPTION
 WHEN exception1 [OR exception2 ...] THEN
 statement1 ;
 statement2 ;
 ...
 [WHEN exception3 [OR exception4 ...] THEN
 statement1 ;
 statement2 ;
 ...]
 [WHEN OTHERS THEN
 statement1 ;
 statement2 ;
 ...]
```

EXAMPLES

```
-----------------------------------------------
CREATE OR REPLACE FUNCTION
get_RFC_F(p_parental_last_name VARCHAR2)
    return varchar2
IS
    last_name varchar2(999):=
initcap(p_parental_last_name);
    rfc VARCHAR2(2);
BEGIN
         rfc := upper(substr(last_name,1,1)) ||
upper(substr(last_name, (regexp_instr(last_name,
'[aeiou]')),1));
    return rfc;
END;
/
select last_name, get_RFC_F(last_name) rfc from
employees;
-----------------------------------------------
SET SERVEROUTPUT ON;
DECLARE
    CURSOR cursor_id_skipped IS
        SELECT department_id from departments;
    v_last_id departments.department_id%TYPE :=
0;
    v_current_id departments.department_id%TYPE;
BEGIN
    dbms_output.put_line('Id`s skipped');
    FOR v_dept IN cursor_id_skipped LOOP
        v_current_id := v_dept.department_id;
            WHILE v_current_id != v_last_id LOOP
                IF v_current_id != (v_last_id +
10)  THEN

DBMS_OUTPUT.PUT_LINE(v_last_id + 10);
                END IF;
                v_last_id := v_last_id + 10;
            END LOOP;
    END LOOP;
END;
-----------------------------------------------
set SEROUTPUT on;
CREATE OR REPLACE PROCEDURE
get_RFC_P(p_parental_last_name VARCHAR2)
IS
    last_name varchar2(100):=
initcap(p_parental_last_name);
    rfc VARCHAR2(2);
BEGIN
    rfc := upper(substr(last_name,1,1)) ||
upper(substr(last_name, (regexp_instr(last_name,
'[aeiou]')),1));
    dbms_output.put_line('The RFC from last name
' || upper(last_name) || ' is: ' || rfc);
END;
/
--llamada del procedimiento
call get_rfc_P('agUIleRA');
-----------------------------------------------
set SERVEROUTPUT on;
DECLARE
    CURSOR cur_reg is
        select * from regions;

    CURSOR cur_countries (p_reg number) is
        select * from countries
        where region_id = p_reg;
    CURSOR cur_loc (p_country VARCHAR2) is
        select * from locations
        where country_id = p_country;
begin
    for r_regions in cur_reg loop
        DBMS_OUTPUT.PUT_LINE(upper('Region: ' ||
r_regions.region_name));
        --DBMS_OUTPUT.PUT_LINE(upper('countries
from the region: '));
        FOR r_countries in cur_countries
(r_regions.region_id) loop

DBMS_OUTPUT.PUT_LINE(upper(r_countries.country_n
ame));
            DBMS_OUTPUT.PUT_LINE(upper(' cities
from the region: '));
            FOR r_locations in cur_loc
(r_countries.country_id) loop
                DBMS_OUTPUT.PUT_LINE(upper('  -'
|| r_locations.state_province));
            end loop;
            DBMS_OUTPUT.PUT_LINE('');
        end loop;
        DBMS_OUTPUT.PUT_LINE('');
    end loop;
end;
-----------------------------------------------
DECLARE
    strng VARCHAR2(400);
    j     NUMBER := 1;
    total VARCHAR2(100);
    CURSOR c_employees IS
          SELECT * FROM employees
          WHERE department_id = 60;
BEGIN
    FOR c_emp IN c_employees LOOP
        strng := 'Hola ' || c_emp.last_name || '
eres el N° ' || j;
        j := j + 1;
        dbms_output.put_line(strng);
    END LOOP;
dbms_output.put_line('_________________________');
    total := '            Total =' || ( j - 1 );
    dbms_output.put_line(total);
END;
-----------------------------------------------
CREATE OR REPLACE FUNCTION ageRange(p_date in
date)
    return varchar2
is
    v_sysdate date;
    v_age INTEGER;
    str varchar2(50);
begin
    select sysdate into v_sysdate from dual;
v_age := trunc((v_sysdate - p_date)/365);
 str :=
        case
            when v_age between 0 and 30 then
'1ra Edad'
            when v_age between 31 and 59 then
'2da Edad'
            when v_age between 60 and 90 then
'3ra Edad'
            else 'Horas Extra'
        end;
    return str;
end;
/
select ageRange(to_date('17 dec 1000', 'dd mon
yyyy')) from dual;

-----------------------------------------------
SET SERVEROUTPUT ON;
DECLARE
    CURSOR cursor_id_skipped IS
        SELECT department_id from departments;
    v_last_id departments.department_id%TYPE :=
0;
    v_current_id departments.department_id%TYPE;
BEGIN
    dbms_output.put_line('Id`s skipped');
    FOR v_dept IN cursor_id_skipped LOOP
        v_current_id := v_dept.department_id;
            WHILE v_current_id != v_last_id LOOP
                IF v_current_id != (v_last_id +
10)  THEN

DBMS_OUTPUT.PUT_LINE(v_last_id + 10);
                END IF;
                v_last_id := v_last_id + 10;
            END LOOP;
    END LOOP;
END;
```