★★★★★ PL/SQL Cheatsheet ★★★★★

SELECT Statement
```
SELECT [DISNCT] {*, column [alias],...}
   FROM table
   [WHERE condition(s)]
   [ORDER BY {column, exp, alias} [ASC|DESC]]
```
Cartesian Product
```
SELECT table1.*, table2.*,[...]
   FROM table1,table2[,...]
```
Equijoin(Simple joins or inner join)
```
SELECT table1.*,table2.*
   FROM  table1,table2
   WHERE table1.column = table2.column
```
Non-Equijoins
```
SELECT table1.*, table2.*
   FROM table1, table2
   WHERE table1.column
   BETWEEN table2.column1 AND table2.column2
```
Outer joins
```
SELECT table1.*,table2.*
   FROM  table1,table2
   WHERE table1.column(+) = table2.column
SELECT table1.*,table2.*
   FROM  table1,table2
   WHERE table1.column = table2.column(+)
```
Self joins
```
SELECT alias1.*,alias2.*
   FROM  table1 alias1,table1 alias2
   WHERE alias1.column = alias2.column
```
Aggregation Selecting
```
SELECT [column,] group_function(column)
   FROM table
   [WHERE condition]
   [GROUP BY group_by_expression]
   [HAVING group_condition]
   [ORDER BY column] ;
```
Group function
```
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)
```
Subquery
```
SELECT select_list
   FROM table
   WHERE expr operator(SELECT select_list FROM table);
single-row comparison operators
       =   >   >=  <   <=   <>
multiple-row comparison operators
        IN   ANY   ALL
```
Multiple-column Subqueries
```
SELECT column, column, ...
   FROM table
   WHERE (column, column, ...) IN
          (SELECT column, column, ...
           FROM table
            WHERE condition) ;
```

Manipulating Data

INSERT Statement(one row)
```
INSERT INTO table [ (column [,column...])]
  VALUES        (value [,value...]) ;
```
INSERT Statement with Subquery
```
INSERT INTO table [ column(, column) ]
       subquery ;
```
UPDATE Statement
```
UPDATE table
   SET column = value [, column = value,...]
   [WHERE condition] ;
```
Updating with Multiple-column Subquery
```
UPDATE table
   SET (column, column,...) =
        (SELECT column, column,...
         FROM table
         WHERE condition)
   WHERE condition ;
```
Deleting Rows with DELETE Statement
```
DELETE [FROM] table
   [WHERE conditon] ;
```
Deleting Rows Based on Another Table
```
DELETE FROM table
   WHERE column = (SELECT column
                   FROM table
                   WHERE condtion) ;
```
Transaction Control Statements
```
COMMIT ;
SAVEPOINT name ;
ROLLBACK [TO SAVEPOINT name] ;
```
CREATE TABLE Statement
```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr] [,...]) ;
```
CREATE TABLE Statement with Subquery
```
CREATE TABLE [schema.]table
        [(column, column...)]
   AS subquery
```
Datatype

VARCHAR2(size)	CHAR(size)	NUMBER(p,s)	DATE
LONG	CLOB	RAW	LONG RAW
BLOB	BFILE		

ALTER TABLE Statement (Add columns)
```
ALTER TABLE table
   ADD  (column datatype [DEFAULT expr]
        [, column datatype]...) ;
```
Changing a column's type, size and default of a Table
```
ALTER TABLE table
   MODIFY  (column datatype [DEFAULT expr]
            [, column datatype]...) ;
```
Dropping a Table
```
DROP TABLE table ;
```
Changing the Name of an Object
```
RENAME old_name TO new_name ;
```
Trancating a Table
```
TRUNCATE TABLE table ;
```
Adding Comments to a Table
```
COMMENT ON TABLE table | COLUMN table.column
     IS ‘text’ ;
```

Dropping a comment from a table
```
COMMENT ON TABLE table | COLUMN table.column IS ‘’ ;
```
Data Dictionary

ALL_OBJECTS	USER_OBJECTS
ALL_TABLES	USER_TABLES
ALL_CATALOG	USER_CATALOG or CAT
ALL_COL_COMMENTS	USER_COL_COMMENTS
ALL_TAB_COMMENTS	USER_TAB_COMMENTS

Defining Constraints
```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][NOT NULL]
        [column_constraint],...
        [table_constraint][,...]) ;
```
Column constraint level
```
column [CONSTRAINT constraint_name] constraint_type,
```
Constraint_type
```
PRIMARY KEY    REFERENCES table(column)      UNIQUE
CHECK (codition)
```
Table constraint level(except NOT NULL)
```
column,...,[CONSTRAINT constraint_name]
    constraint_type (column,...),
```
NOT NULL Constraint (Only Column Level)
```
CONSTRAINT table[_column...]_nn NOT NULL ...
```
UNIQUE Key Constraint
```
CONSTRAINT table[_column..]_uk UNIQUE (column[,...])
```
PRIMARY Key Constraint
```
CONSTRAINT table[_column..]_pk PRIMARY (column[,...])
```
FOREIGN Key Constraint
```
CONSTRAINT table[_column..]_fk
    FOREIGN KEY (column[,...])
    REFERENCES table (column[,...])[ON DELETE CASCADE]
```
CHECK constraint
```
CONSTRAINT table[_column..]_ck CHECK (condition)
```
Adding a Constraint(except NOT NULL)
```
ALTER TABLE table
   ADD [CONSTRAINT constraint_name ] type (column) ;
```
Adding a NOT NULL constraint
```
ALTER TABLE table
   MODIFY (column datatype [DEFAULT expr]
   [CONSTRAINT constraint_name_nn] NOT NULL) ;
```
Dropping a Constraint
```
ALTER TABLE table
   DROP CONSTRAINT constraint_name ;
ALTER TABLE table
   DROP PRIMARY KEY | UNIQUE (column) |
   CONSTRAINT constraint_name [CASCADE] ;
```
Disabling Constraints
```
ALTER TABLE table
   DISABLE CONSTRAINT constraint_name [CASCADE] ;
```
Enabing Constraints
```
ALTER TABLE table
   ENABLE CONSTRAINT constraint_name ;
```
Data Dictionary

ALL_CONSTRAINTS	USER_CONSTRAINTS
ALL_CONS_COLUMNS	USER_CONS_COLUMNS

Creating a View
```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
```

```
       [(alias[, alias]...)]
   AS subquery
   [WITH CHECK OPTION [CONSTRAINT constraint_name]]
   [WITH READ ONLY] ;
```
Removing a View
```
DROP VIEW view ;
```
CREATE SEQUENCE Statement
```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n| NOMAXVALUE}]
       [{MINVALUE n| NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE [n|20]| NOCACHE}] ;
```
Pseudocolumns
```
sequence.NEXTVAL      sequence.CURRVAL
```
Modifying a Sequence (No START WITH option)
```
ALTER SEQUENCE sequence
       [INCREMENT BY n]
       [{MAXVALUE n| NOMAXVALUE}]
       [{MINVALUE n| NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE [n|20]| NOCACHE}] ;
```
Removing a Sequence
```
DROP SEQUENCE sequence ;
```
Creating an Index
```
CREATE INDEX index
   ON TABLE (column[,column]...) ;
```
Removing an Index
```
DROP INDEX index ;
```
Synoyms
```
CREATE [PUBLIC] SYNONYM synonym FOR object ;
```
Removing Synonyms
```
DROP SYNONYM synonym ;
```
Data Dictionary

ALL_VIEWS	USER_VIEWS
ALL_SEQUENCES	USER_SEQUENCES
ALL_INDEXES	USER_INDEXES
ALL_IND_COLUMNS	USER_IND_COLUMNS

System Privileges(DBA)	**User System Privileges**
CREATE USER	CREATE SESION
DROP USER	CREATE TABLE
DROP ANY TABLE	CREATE SEQUENCE
BACKUP ANY TABLE	CREATE VIEW
	CREATE PROCEDURE

Creating Users
```
CREATE USER user
   IDENTIFIED BY password ;
```
Creating Roles
```
CREATE ROLE role ;
```
Granting System Privileges
```
GRANT privelges[,...] TO user[,...] ;
GRANT privelges TO role ;
GRANT role TO user[,...] ;
```
Changing Password
```
ALTER USER user IDENTIFIED BY password ;
```
Dropping Users
```
DROP USER user [CASCADE] ;
```
Dropping Roles
```
DROP ROLE role ;
```
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		

Object Privileges
```
GRAND object_priv [(column)]
   ON object
   TO  {user|role|PUBLIC}
   [WITH GRANT OPTION] ;
```
Revoking Object Privileges
```
REVOKE {privilege [,privilege...] | ALL}
   ON  object
   FROM {user[,user...]|role|PUBLIC}
   [CASCADE CONSTRAINTS] ;
```
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

PL/SQL Block Structure
```
DECLARE --Optional
 --Variables, Cursors, User-defined exceptions
BEGIN --Mandatory
 --SQL statements
 --PL/SQL statements
EXCEPTION --Optional
 --Actions to perform when errors occur
END ; --Mandatory
```
PL/SQL Block Type
```
Anonymous      Procedure          Function
[DECLARE]      PROCEDURE name     FUNCTION name
               IS                 RETURN datatype IS
               [DECLARE]          [DECLARE]
BEGIN          BEGIN              BEGIN
--statements   --statements       --statements
[EXCEPTION]    [EXCEPTION]        [EXCEPTION]
END ;          END ;              END ;
```
Declaring PL/SQL Variables
```
identifier [CONSTANT] datatype [NOT NULL]
  [:=|DEFAULT expr] ;
```
Assigning Values to Variables
```
identifier := expr ;
```
Base Scalar Datatypes
```
VARCHAR2(n)    NUMBER(p,s)    DATE      CHAR(n)
LONG           LONG RAW       BOOLEAN
BINARY_INTEGER PLS_INTEGER
```
The %TYPE Attribute
```
table_name.column_name%TYPE ;
variable_name%TYPE ;
```
Composite Datatypes
```
TABLE          RECORD         NESTED TABLE   VARRAY
```
LOB Datatypes
```
CLOB           BLOB           BFILE          NCLOB
```
Creating Bind Variables
```
VARIABLE variable_name dataype
```
Displaying Bind Variables
```
PRINT [variable_name]
```
Commenting Code
```
--prefix single-line comments with two dashes
/* Place muti-line comment between the symbols */
```
SELECT Statements in PL/SQL
```
SELECT {column_list|*}
INTO {variable_name[,variable_name]...
      |record_name}
FROM table
WHERE condition
```
Implicit Cursor Attributes for DML statements
```
SQL%ROWCOUNT
SQL%FOUND
SQL%NOTFOUND
SQL%ISOPEN
```
Constrol Structures
```
IF Statement                 Basic Loop
IF condition THEN            LOOP
   statements ;                statements;
[ELSIF condition THEN          ...
   statements ;]               EXIT [WHEN condition];
[ELSE                        END LOOP
   statements;]
END IF ;
FOR Loop                     WHILE Loop
FOR conter in [REVERSE]      WHILE condition LOOP
  lower..upper LOOP             statement1;
  statement1;                   statement2;
  statement2;                   ...
  ...                        END LOOP ;
END LOOP;
```
Creating a PL/SQL Record
```
TYPE record_name_type IS RECORD
     (field_declaration[,field_declaration]...) ;
record_name record_name_type ;
```
Where field_declaration is
```
field_name {field_type|variable%TYPE|
                 table.column%TYPE|table%ROWTYPE}
                 [[NOT NULL] {:=|DEFAULT} expr]
```
Referencing Fields in the Record record_name.field_name

Declaring Records with the %ROWTYPE Attribute
```
DECLARE
       record_name    reference%ROWTYPE
```
Creating a PL/SQL Table
```
TYPE type_name IS TABLE OF
    {column_scalr_type|variable%TYPE|table.column%TYPE
    |variable%ROWTYPE} [NOT NULL]
    [INDEX BY BINARY_INTEGER];
identifier type_name ;
```
Referencing a PL/SQL table
```
pl_sql_table_name(primary_key_value)
```

Using PL/SQL Table Method
```
table_name.method_name[(parameters)]
```
PL/SQL Table Methods
```
EXITS(n)       COUNT   FIRST   LAST    PRIOR(n)
NEXT(n)        EXTEND(n,i)     TRIM    DELETE
```
PL/SQL Table of Records
```
TYPE table_name_type IS TABLE OF table_name%ROWTYPE
     INDEX BY BINARY_INTEGER ;
table_name table_name_type ;
```
Referencing a Table of Records
```
table_name(index).field
```
Declaring the Cursor in Declaration Section
```
CURSOR cursor_name IS select_statement ;
record_name cursor_name%ROWTYPE ;
```
Opening and Closing the Cursor
```
OPEN cursor_name ;
CLOSE cursor_name ;
```
Fetching Data from the Cursor
```
FETCH cursor_name
INTO [variable1(,variable2,...)
               |record_name] ;
```
Explicit Cusor Attributes
```
cursor_name%ISOPEN
cursor_name%NOTFOUND
cursor_name%FOUND
cursor_name%ROWCOUNT
```
Cursor FOR Loops
```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  ...
END LOOP;
```
Cursor FOR Loops Using Subqueries
```
FOR record_name IN (subqueries) LOOP
  statement1
  ...
END LOOP ;
```
Cursors with Parameters
```
CURSOR cursor_name [(cursor_parameter_name datatype
[,...])]
IS select_statement
[FOR UPDATE [OF column_reference][NOWAIT]];
```
Parameter Name
```
cursor_parameter_name [IN] datatype [{:=|DEFAULT}expr]
```
Openning with Parameters
```
OPEN cursor_name(cursor_parameter_name[,...]);
```
Cursor FOR Loops with parameters
```
FOR record_name IN cursor_name(cursor_parameter_name
[,...]) LOOP
  statement1;
  statement2;
  ...
END LOOP;
```
WHERE CURRENT OF clause
```
UPDATE|DELETE ... WHERE CURRENT OF cursor_name ;
```
Predefined Exceptions
```
NO_DATA_FOUND
TOO_MANY_ROWS
INVALID_CURSOR
ZERO_DIVIDE
DUP_VAL_ON_INDEX
```
Trapping Exceptions
```
EXCEPTION
   WHEN exception1 [OR exception2 ...] THEN
      statement1 ;
      statement2 ;
      ...
  [WHEN exception3 [OR exception4 ...] THEN
      statement1 ;
      statement2 ;
      ...]
  [WHEN OTHERS THEN
      statement1 ;
      statement2 ;
      ...]
```
Declaring Non-Predefined Oracle Sever Exception
```
DECLARE
   exception EXCEPTION ;
   PRAGMA EXCEPTION_INIT(exception, error_number) ;
```
Referencing the declared Non-predefined execption
```
BEGIN
   ...
EXCEPTION
   WHEN exception THEN
      statement1 ;
      ...
END ;
```
Trapping User-Defined Exceptions
```
DECLARE
   exception EXCEPTION ;
BEGIN
   ...
   IF SQL%NOTFOUND THEN
      RAISE exception ;
   END IF ;
   ...
EXCEPTION
   WHEN exception THEN
      statement1 ;
      ...
END ;
```
Functions for Trapping Exceptions

SQLCODE	return error code
SQLERRM	return error message

RAISE_APPLICATION_ERROR procedure(Executable/Exception Section)
```
RAISE_APPLICATION_ERROR ( error_number,
                          message [, {TRUE|FALSE}]) ;
```

error_number	between -20000 to -20999
message	string up to 2,048 bytes long
TRUE	placed on the stack of previous errors.
FALSE	replaces all previous errors

Single-Row Functions

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

Comparison	= > >= < <= <>
	BETWEEN..AND, IN, LIKE, IS NULL
Logical	AND OR NOT

Order of Operations

Operator	Operation
**,NOT	Exponentiation, logical negation
+,-	Identity, negation
*,/	Muliplication, division
+,-,||	Addition, subtraction, concatenation =,!
=,<,>,<=	Comparison
>=,IS NULL,LIKE	
BETEEN,IN	
AND	Conjunction
OR	Inclusion

"If you like our course and if you have an extra minute and wouldn't mind leaving a 5-star review, we would greatly appreciate it!"

Best Wishes,
(Oracle Masters Team)