ORACLE PL/SQL Cheat Sheet prepared by Haradhan Pal (Haradhan Automation Library)	
YouTube Channel Link	https://www.youtube.com/c/HaradhanAutomationLibrary?sub_confirmation=1
SQL	SQL stands for Structured Query Language. As the name suggests, it is a structured language via which you can query the database for performing various tasks such as Storing, Manipulating, and retrieving data from a database. SQL is the standard language when it comes to communicating with powerful relational databases such as Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Anything related to data in relational databases such as creating tables, limiting access to data, sorting, filtering, grouping, etc. is achieved using SQL.
RDBMS	RDBMS Stands for Relational DataBase Management System and is a collection of tools that allow users to organize, manipulate, and visualize databases. RDMBS follows some standards that allow for the fastest response from a database and make it easier for humans to interact with a database.
Oracle SQL	It is the world's most widely used database management system. It is used to store and retrieve information. Oracle database is designed for enterprise grid computing.
PostgreSQL	It is an open-source, powerful and advanced version of SQL that supports different functions of SQL including, foreign keys, subqueries, functions, and user-defined types.
Scalar	Single values with no internal components, such as a NUMBER, DATE, or BOOLEAN.
Large Object (LOB)	Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.
Composite	Data items that have internal components that can be accessed individually. For example, collections and records.
Reference	Pointers to other data items.
Scalar, Numeric, Date/Time & Character Data Types	
Date Type	**Description**
Numeric	Numeric values on which arithmetic operations are performed.
Character	Alphanumeric values that represent single characters or strings of characters.
Boolean	Logical values on which logical operations are performed.
Datetime	Dates and times.
PLS_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
BINARY_INTEGER	Signed integer in range -2,147,483,648 through 2,147,483,647, represented in 32 bits
BINARY_FLOAT	Single-precision IEEE 754-format floating-point number
BINARY_DOUBLE	Double-precision IEEE 754-format floating-point number
NUMBER(prec, scale)	Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0
DEC(prec, scale)	ANSI specific fixed-point type with maximum precision of 38 decimal digits
DECIMAL(prec, scale)	IBM specific fixed-point type with maximum precision of 38 decimal digits
NUMERIC(pre, secale)	Floating type with maximum precision of 38 decimal digits
DOUBLE PRECISION	ANSI specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
FLOAT	ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)
INT	ANSI specific integer type with maximum precision of 38 decimal digits
INTEGER	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
SMALLINT	ANSI and IBM specific integer type with maximum precision of 38 decimal digits
REAL	Floating-point type with maximum precision of 63 binary digits (approximately 18 decimal digits)

DATE	A simple date in YYYY-MM–DD format, supporting a range from ‘1000-01-01’ to ‘9999-12-31’.
TIME(fsp)	A time in hh:mm:ss format, with a supported range from ‘-838:59:59’ to ‘838:59:59’
DATETIME(fsp)	A date and time combination in YYYY-MM-DD hh:mm:ss format. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP(fsp)	A Unix Timestamp, which is a value relative to the number of seconds since the Unix epoch (‘1970-01-01 00:00:00’ UTC). This has a supported range from ‘1970-01-01 00:00:01’ UTC to ‘2038-01-09 03:14:07’ UTC.
YEAR	A year in four-digit format with the range as - 1901 to 2155
CHAR	A fixed-length character string with a maximum size of 32,767 bytes
VARCHAR2	A variable-length character string with a maximum size of 32,767 bytes
RAW	A variable-length binary or byte string with a maximum size of 32,767 bytes, not interpreted by PL/SQL
NCHAR	A fixed-length national character string with a maximum size of 32,767 bytes
NVARCHAR2	A variable-length national character string with a maximum size of 32,767 bytes
LONG	A variable-length character string with a maximum size of 32,760 bytes
LONG RAW	A variable-length binary or byte string with a maximum size of 32,760 bytes, not interpreted by PL/SQL
ROWID	Physical row identifier, the address of a row in an ordinary table
UROWID	Universal row identifier (physical, logical, or foreign row identifier)
PL/SQL Operators	
Operator	**Description**
+	Adds two operands
–	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
**	Exponentiation operator, raises one operand to the power of other
+=	Add Equals
-=	Subtract Equals
*=	Multiply Equals
/=	Divide Equals
%=	Modulo Equals
&=	Bitwise AND Equals
^-=	Bitwise Exclusive Equals
|*=	Bitwise OR Equals
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=<>~=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records

IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition
Collection Methods	
EXISTS(n)	Returns TRUE if the nth element in a collection exists; otherwise returns FALSE.
COUNT	Returns the number of elements that a collection currently contains.
LIMIT	Checks the maximum size of a collection.
FIRST	Returns the first (smallest) index numbers in a collection that uses the integer subscripts.
LAST	Returns the last (largest) index numbers in a collection that uses the integer subscripts.
PRIOR(n)	Returns the index number that precedes index n in a collection.
NEXT(n)	Returns the index number that succeeds index n.
EXTEND	Appends one null element to a collection.
EXTEND(n)	Appends n null elements to a collection.
EXTEND(n,i)	Appends n copies of the ith element to a collection.
TRIM	Removes one element from the end of a collection.
TRIM(n)	Removes n elements from the end of a collection.
DELETE	Removes all elements from a collection, setting COUNT to 0.
DELETE(n)	Removes the nth element from an associative array with a numeric key or a nested table. If the associative array has a string key, the element corresponding to the key value is deleted. If n is null, DELETE(n) does nothing.
DELETE(m,n)	Removes all elements in the range m..n from an associative array or nested table. If m is larger than n or if m or n is null, DELETE(m,n) does nothing.
CRUD Operations with SQL	CRUD is an acronym that stands for Create, Read, Update, and Delete. These are the most fundamental operations that one can perform on any database. For creating any application, these 4 types of operations are crucial. They are:- INSERT (Create) SELECT (Read) UPDATE (Update) DELETE (Delete)
List of useful SQL Keywords and their Description	
Keyword	**Description**
ADD	Add a new column to an existing table. Eg: ALTER TABLE customers ADD email_address VARCHAR(255);
ALTER TABLE	Adds, deletes, or edits columns/constraints in a table. Eg: ALTER TABLE customers DROP COLUMN email_address;
ALTER COLUMN	Changes the data type of a table's column. Eg: ALTER TABLE customers ALTER COLUMN phone varchar(50)
AS	Renames a table or column with an alias value that only exists for the duration of the query. Eg: SELECT name AS customer_name, phone, postalCode FROM customers;
ASC	Used with ORDER BY to return the data in ascending order.
CHECK	Adds a constraint that limits the value which can be added to a column. Eg: CREATE TABLE Users(firstName varchar(255),age INT, CHECK(age>10));
CREATE DATABASE	Creates a new database. Eg: CREATE DATABASE my website;
CREATE TABLE	Creates a new table. Eg: CREATE TABLE users (id int,firsr_name varchar(255), surname varchar(255), address varchar(255), contact_number int);

DEFAULT	Set the default value for a column. Eg: CREATE TABLE products(ID int, name varchar(255) DEFAULT 'Username', from date DEFAULT GETDATE());
DELETE	Delete values from a table. DELETE FROM users WHERE user_id= 674;
DESC	Used with ORDER BY to return the data in descending order.
DROP COLUMN	Deletes a column from a table. ALTER TABLE users DROP COLUMN first_name;
DROP DATABASE	Deletes a complete database along with all the tables and data inside. Eg: DROP DATABASE my website;
DROP DEFAULT	Removes a default value for a column. Eg: ALTER TABLE products ALTER COLUMN name DROP DEFAULT;
DROP TABLE	Delete a table from a database. Eg: DROP TABLE customers;
FROM	Specifies which table to select or delete data from. Eg: SELECT * FROM customers;
IN	Used with a WHERE clause as a shorthand for multiple OR conditions. Eg: SELECT * FROM users WHERE country IN('USA', 'United Kingdom','Russia');
IS NULL	Tests for empty (NULL) values. Eg: SELECT * FROM users WHERE phone IS NULL;
IS NOT NULL	Opposite of IS NULL. Tests for values that are not null.
LIKE	Returns true if the operand value matches a pattern. SELECT * FROM users WHERE first_name LIKE '%son';
ORDER BY	Used to sort the resultant data in ascending (default) or descending order.
SELECT DISTINCT	Same as SELECT, except duplicate values are excluded. Eg: SELECT DISTINCT postalCode from customers;
TOP	Used alongside SELECT to return a set number of records from a table. Eg: SELECT TOP 5 * FROM customers;
VALUES	Used alongside the INSERT INTO keyword to add new values to a table. Eg: INSERT INTO cars (name, model, year) VALUES ('Ford', 'Fiesta', 2010);
WHERE	Filters result only includes data that meets the given condition. SELECT * FROM orders WHERE quantity > 1;
PL/SQL Statemement	**Sample Query**
SELECT	SELECT * FROM beverages WHERE field1 = 'Kona' AND field2 = 'coffee' AND field3 = 122;
SELECT INTO	SELECT name,address,phone_number INTO v_employee_name,v_employee_address,v_employee_phone_number FROM employee WHERE employee_id = 6;
INSERT	INSERT INTO table_name VALUES ('Value1', 'Value2', ...); INSERT INTO table_name(Column1, Column2, ...) VALUES ('Value1', 'Value2', ...);
DELETE	DELETE FROM table_name WHERE some_column=some_value DELETE FROM customer WHERE sold = 0;
UPDATE	UPDATE customer SET name='Joe' WHERE customer_id=10; UPDATE movies SET invoice='paid' WHERE paid > 0;
SEQUENCES	CREATE SEQUENCE sequence_name MINVALUE value MAXVALUE value START WITH value INCREMENT BY value CACHE value;
ALTER SEQUENCE	ALTER SEQUENCE <sequence_name> MAXVALUE <integer>; ALTER SEQUENCE seq_maxval MAXVALUE 10;
Create table	CREATE TABLE employee (id int, name varchar(20));

Add column	ALTER TABLE employee ADD (id int)
Modify column	ALTER TABLE employee MODIFY(sickHours s float);
Drop column	ALTER TABLE employee DROP COLUMN vacationPay;
Create an index	CREATE INDEX customer_idx ON customer (customer_name);
Rename an Index	ALTER INDEX customer_id RENAME TO new_customer_id;
Drop an index	DROP INDEX customer_idx;
Creating a user	CREATE USER username IDENTIFIED BY password;
Change password	ALTER USER username IDENTIFIED BY password;
DISTINCT	SELECT DISTINCT expression FROM table_name WHERE [condition];
GROUP BY	SELECT col_name FROM table_name WHERE condition GROUP BY col_name(s)
ORDER BY	SELECT * FROM table_name WHERE condition ORDER BY expression [ASC|DESC];
UNION	SELECT exp_1,..exp_n FROM table_1 WHERE condition UNION SELECT exp_1,..exp_n FROM table_2 WHERE condition;
INTERSECT	SELECT exp_1,..exp_n FROM table_1 WHERE condition INTERSECT SELECT exp_1,..exp_n FROM table_2 WHERE condition;
INNER JOIN	SELECT col_1,..col_n from table_1 INNER JOIN ON table_1.col = table2.col
LEFT OUTER JOIN	SELECT table1.col1, table1.col2, table2.col1,… FROM table1 LEFT JOIN table2 ON condition;
RIGHT OUTER JOIN	SELECT table1.col1, table1.col2, table2.col1,… FROM table1 RIGHT JOIN table2 ON condition;
FULL OUTER JOIN	SELECT table1.col1, table1.col2, table2.col1,… FROM table1 FULL JOIN table2 ON condition;
SEMI JOINS	SELECT col1, col2 FROM table1 WHERE id IN (SELECT table1_id FROM table2 WHERE condition)
Conditions and Loop	

IF-THEN Statement	IF condition THEN
	Statement;
	END IF;

IF-THEN-ELSE Statement	IF condition THEN
	Statement1;
	ELSE
	Statement2;
	END IF;

IF-THEN-ELSIF Statement	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 Statement	CASE selector
	  WHEN 'value1' THEN Statement1;
	  WHEN 'value2' THEN Statement2;
	  WHEN 'value3' THEN Statement3;
	  ...
	  ELSE Sn; -- default case
	END CASE;

Nested IF-THEN-ELSE Statements	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;

Basic Loop	LOOP Sequence of statements; END LOOP;
WHILE Loop	WHILE condition LOOP sequence_of_statements END LOOP;
FOR Loop	FOR counter IN initial_value .. final_value LOOP sequence_of_statements; END LOOP;

Nested FOR 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;

Nested WHILE LOOP	WHILE condition1 LOOP
	  sequence_of_statements1
	  WHILE condition2 LOOP
	  sequence_of_statements2
	  END LOOP;
	END LOOP;

SQL KEYS	
KEYS	KEYS in the database helps you to identify a tuple(row) in a relation(table). It allows you to establish a relationship between tables and also identify the relationships between tables. There are no. of keys in database like **super key, primary key, foreign key, composite key, unique key, alternate key, etc.
PRIMARY KEY	A column of a table is said to be a primary key if it uniquely identifies each row in that table.
FOREIGN KEY	A foreign key is a column that is used to link two tables together.
COMPOSITE KEY	It is the combination of two or more columns in a table that can be used to uniquely identify each row in the table.
UNIQUE KEY	It is a set of one or more than one column of a table that uniquely identifies a record in a database table.