|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CONCEPT** | **DESCRIPTION** | **SYNTAX** | | | | | **EXAMPLE/EXECUTION** | | |
| PL/SQL BLOCK STRUCTURE | DECLARE-variables,cursors,user-defined exceptions  BEGIN-sql statements,pl/sql statements  EXCEPTION-actions to perform when errors occur  END; | DECLARE  BEGIN  Statements;  END; | | | | | DECLARE  v\_dname varchar2(15) := ‘Accounts';  BEGIN  INSERT into department\_master  VALUES (50, v\_dname);  END; | | |
| TYPES OF BLOCKS | ANONYMOUS,PROCEDURE,FUNCTION | DECLARE  BEGIN  --statements  EXCEPTION  END; | PROCEDURE name IS  BEGIN  ----statements  EXCEPTION  END; | | FUNCTION name RETURN datatype IS  BEGIN  --statements  RETURN value  EXCEPTION  END; | |  | | |
| TYPES OF VARIABLES | **PL/SQLVARIABLES:**  SCALAR,COMPOSITE,REFERENCE,LOB(LARGE OBJECTS)  **NON PL/SQL VARIABLES:** HOST,BIND | identifier [CONSTANT] datatype [NOT NULL]  [:= | DEFAULT expr]; | | | | | DECLARE  v\_hiredate DATE;  v\_deptno NUMBER(2) NOT NULL := 10;  v\_location VARCHAR2(13) := 'Atlanta';  c\_comm CONSTANT NUMBER := 1400; | | |
| DECLARING DATATYPE | %TYPE  %ROWTYPE | Variablename tablename.coulmnname %TYPE  Variablename tablename.coulmnname %rowtype | | | | | v\_name staff\_master.staff\_name%TYPE;  v\_balance NUMBER(7,2);  v\_min\_balance  v\_balance%TYPE := 10; | | DECLARE  nRecord staff\_master%rowtype;  BEGIN  SELECT \* into nrecord  FROM staff\_master  WHERE staff\_code = 100001;  UPDATE staff\_master  SET staff\_sal = staff\_sal + 101;  WHERE emp\_code = 100001;  END; |
| RECORD DATATYPE | * Define RECORD type,then declare records of that type. * Define in the declarative part of any block,subprograms,   packages. | TYPE type\_name IS RECORD (field\_declaration [,field\_ declaration] ...); | | | | | DECLARE  TYPE recname is RECORD  (customer\_id number,  customer\_name varchar2(20));  var\_rec recname;  BEGIN  var\_rec.customer\_id:=20;  var\_rec.customer\_name:=‘Smith';  dbms\_output.put\_line(var\_rec.customer\_id||' '||var\_rec.customer\_name);  END; | | |
| TABLE DATATYPE | PL/SQL table is like an array collection of structure. | TYPE type\_name is TABLE OF  {Column\_type | table.column%type} [NOT NULL]  INDEX BY BINARY\_INTEGER; | | | | | DECLARE  TYPE student\_table is table of student\_master.student\_name%type  INDEX BY BINARY\_INTEGER; | | |
| TYPES OF STATEMENTS | INSERT,DELETE,  UPDATE,SELECT | **SELECT** Column\_List INTO Variable\_List  FROM Table\_List  [WHERE expr1]  **CONNECT BY** expr2 [START WITH expr3]]  **GROUP BY** expr4] [HAVING expr5]  [UNION | INTERSECT | MINUS SELECT ...]  [ORDER BY expr | ASC | DESC]  [FOR UPDATE [OF Col1,...] [NOWAIT]]  INTO Variable\_List; | | | | | DECLARE  deptno number(10) := 30;  dname varchar2(15) ;  BEGIN  SELECT dept\_name INTO dname FROM department\_master  WHERE dept\_code = Block1. deptno;  DELETE FROM department\_master  WHERE dept\_code = Block1. deptno ;  END; | | |
| TYPES OF PROGRAMMATIC CONSTRUCTS | SELECTION,  ITERATION,  SEQUENCE | IF(selection construct):  IF condition  Then  Statements  Elsif condition  End if; | LOOP  (iteration construct)  LOOP  Statement  Exit when  (condition)  END LOOP | FOR(iteration construct)  FOR Variable IN [REVERSE] LOOP  PL/SQL\_Statements  END LOOP ; | | WHILE(sequence construct)  WHILE Condition  LOOP  PL/SQL  Statements;  END LOOP; | DECLARE  v\_counter number := 50 ;  BEGIN  LOOP  INSERT INTO department\_master  VALUES(v\_counter,'NEWDEPT');  DELETE FROM emp WHERE deptno = v\_counter;  v\_counter := v\_counter +10;  EXIT WHEN v\_counter>100 ;  END LOOP;  COMMIT ;  END ; | DECLARE  v\_counter number := 50 ;  BEGIN  FOR  Loop\_Counter IN 2..5  LOOP  INSERT INTO dept  VALUES(v\_counter ,'NEW DEPT') ;  v\_counter := v\_counter + 10 ;  END LOOP;  COMMIT ;  END ; | |

|  |  |  |  |
| --- | --- | --- | --- |
| **CONTENT** | **DESCRIPTION** | **SYNTAX** | **EXAMPLE** |
| Cursor | A Cursor is a “handle” or “pointer” to the context area. Using this cursor the PL/SQL program can control the context area, and thereby access the information stored in it.  Two types:   * + In an **EXPLICIT CURSOR**, a cursor name is explicitly assigned to a SELECT statement through CURSOR IS statement.   + An **IMPLICIT CURSOR** is used for all other SQL statements. | **EXPLICIT CURSOR:**  **OPEN** Cursor\_Name;  OPEN Cursor-name(param1, param2,..)//Parameterized Cursor  **FETCH** Cursor\_Name INTO List\_Of\_Variables;  **FETCH** Cursor\_Name INTO  PL/SQL \_Record;  **CLOSE** Cursor\_Name; | **IMPLICIT:**  BEGIN  UPDATE dept SET dname =‘Production’ WHERE deptno= 50;  IF SQL%NOTFOUND THEN  INSERT into department\_master VALUES ( 50, ‘Production');  END IF;  END; |
| Cursor Attributes &  REF CURSOR | **%ISOPEN,**  **%FOUND,**  **%NOTFOUND,**  **%ROWCOUNT**   * Ref\_type\_name is a type specifier used in subsequent declarations of cursor variables * Return\_type must represent a record or a row in a database table. | Cur\_Name%ISOPEN  cur\_Name%FOUND  cur\_Name%NOTFOUND  cur\_Name%ROWCOUNT  TYPE ref\_type\_name IS REF CURSOR RETURN return\_type; | **EXPLICIT WITH CURSOR ATTRIBUTE:**  DECLARE  **TYPE staffcurtyp is REF CURSOR RETURN staff\_master%rowtype;**  staff\_cv staffcurtyp; -- declare cursorvariable  staff\_cur staff\_master%rowtype;  BEGIN  **OPEN** staff\_cv for select \* from staff\_master;  LOOP  **EXIT** WHEN **staff\_cv%notfound**;  **FETCH** staff\_cv into staff\_cur;  INSERT into temp\_table  VALUES (staff\_cv.staff\_code,   staff\_cv.staff\_name,staff\_cv.staff\_sal);  END LOOP;  **CLOSE** staff\_cv; END; |
| **FOR** Loop | For all other loops we use OPEN, FETCH, AND CLOSE statements for a cursor. But the **CURSOR FOR Loop** implicitly handles the cursor processing. | **FOR** Variable in Cursor\_Name  LOOP  Process the variables  END LOOP; | DECLARE  CURSOR c\_staff is  SELECT staff\_code, staff\_master.design\_code   FROM staff\_master, designation\_master  WHERE design\_name = ‘Professor' and staff\_sal > 20000 and  staff\_master.design\_code =designation\_master.design\_code  **FOR UPDATE OF** design\_code  NOWAIT;  d\_code designation\_master.design\_code%type;  BEGIN  SELECT design\_code into d\_code  FROM designation\_master  WHERE design\_name=‘Director’;  **FOR** v\_rec in c\_staff  LOOP  UPDATE staff\_master  SET design\_code = d\_code  **WHERE current of** c\_staff;  END LOOP;  END; |
| Method of locking records | **Method of locking records** consists of two parts:  The **FOR UPDATE** clause in CURSOR declaration.  The **WHERE CURRENT OF** clause in an UPDATE or DELETE statement. | CURSOR Cursor\_Name IS  SELECT...FROM…WHERE…  ORDER BY  **FOR UPDATE**[OF column names]  [NOWAIT]  WHERE CURRENT OF Cursor\_Name |

|  |  |  |
| --- | --- | --- |
| CONCEPT | DESCRIPTION | EXAMPLE |
| **Error Handling:** | - A warning or error condition is called an “exception.  - Exceptions can be internally defined (by the run-time system) or user defined. |  |
| **1.Predefined exceptions:** | They are always available to the program. Hence there is no need to declare them. Automatically risen by oracle. | NO\_DATA\_FOUND,CURSOR\_ALREADY\_OPEN, PROGRAM\_ERROR |
| **2.Numbered exceptions:** | User defined exceptions can be named with error number between -20000 and -20999. | PRAGMA EXCEPTION\_INIT(Exception Name,Error\_Number); |
| **3.User defined exceptions:** | declared in the Declaration section, raised in the Executable section, and handled in the  Exception section. | -DECLARE  -RAISE  -HANDLE |
| **3.1.Raising Exceptions:** | Internal exceptions are raised implicitly by the run-time system, as are user-defined exceptions that are associated with an Oracle error number using EXCEPTION\_INIT. | RAISE Exception\_Name; |
| **3.2Control passing to Exception Handler :** | When an exception is raised, normal execution of your PL/SQL block or subprogram stops, and control passes to its exception-handling part.  To catch the raised exceptions, you write “exception handler”. |  |
| **4.OTHERS Exception Handler:** | The optional OTHERS exception handler, which is always the last handler in a block or subprogram, acts as the handler for all exceptions that are not specifically named in the Exception section. |  |
| **5.RAISE\_APPLICATION\_ERROR:** | The procedure RAISE\_APPLICATION\_ERROR lets you issue user-defined ORA- error messages from stored subprograms | RAISE\_APPLICATION\_ERROR( Error\_Number, Error\_Message); |

|  |  |  |  |
| --- | --- | --- | --- |
| **CONCEPT** | **DESCRIPTION** | **SYNTAX** | **EXAMPLE** |
| **DATABASE Triggers** | A Trigger defines an action the database should take when some database related event occurs. | CREATE TRIGGER Trg\_Name  {BEFORE | AFTER} {event} OF Column\_Names ON Table\_Name[FOR EACH ROW][WHEN restriction]  BEGIN  PL/SQL statements;  END Trg\_Name ; | CREATE OR REPLACE TRIGGER display\_salary\_changes  BEFORE DELETE OR INSERT OR UPDATE ON customers  FOR EACH ROW  WHEN (NEW.ID > 0)  DECLARE  sal\_diff number;  BEGIN  sal\_diff := :NEW.salary - :OLD.salary;  dbms\_output.put\_line('Old salary: ' || :OLD.salary);  dbms\_output.put\_line('New salary: ' || :NEW.salary);  dbms\_output.put\_line('Salary difference: ' || sal\_diff);  END; |
| **There are three types of triggers:**  Statement based triggers  Timing based triggers  Level based triggers |
| Triggers can be fired:  before or after the operation  on row or statement operations  Trigger can be fired for more than one type of triggering statement |
| **Disabling and Dropping Triggers** | ALTER TRIGGER Trigger\_Name DISABLE/ENABLE  DROP TRIGGER Trigger\_Name |
| **LOCKS** | **DML Locks** Row Level Locks, Table Level Locks.  **DDL Locks** Exclusive DDL Locks, Shared DDL Locks, Breakable Parse Locks. | LOCK TABLE tablename IN lockmode MODE NOWAIT; | LOCK TABLE staff\_master IN SHARE UPDATE mode nowait;  LOCK TABLE student\_master IN EXCLUSIVE mode; |

**DATABASE TRIGGERS**

**PROCEDURES, FUNCTIONS, AND PACKAGES**

|  |  |  |  |
| --- | --- | --- | --- |
| Procedure | A procedure is a subprogram used to perform a specific action. It contains two parts:   * Specification * Body | CREATE PROCEDURE Proc\_Name  (Parameter {IN | OUT | IN OUT}  datatype := value,...) AS  Variable\_Declaration ;  Cursor\_Declaration ; Exception\_Declaration ;  BEGIN  PL/SQL\_Statements ;  EXCEPTION  Exception\_Definition ;  END Proc\_Name ; | CREATE OR REPLACE PROCEDURE get\_details  (s\_code IN number, s\_name OUT varchar2,s\_sal OUT number ) IS  BEGIN  SELECT staff\_name, staff\_sal INTO s\_name, s\_sal  FROM staff\_master WHERE staff\_code=s\_code;  EXCEPTION  WHEN no\_data\_found THEN  INSERT into auditstaff  VALUES( ‘No employee with id ' || s\_code);  s\_name := null;  s\_sal := null;  END get\_details ;  **EXECUTION:**  variable salary number  variable name varchar2(20)  EXECUTE get\_details(100003,:Salary, :Name)  print salary  print name |
| Function | A function is similar to a Procedure. | CREATE FUNCTION Func\_Name  (Parameter datatype := value,..)  RETURN datatype1 AS  Variable\_Declaration ;  Cursor\_Declaration ;  Exception\_Declaration ;  BEGIN PL/SQL\_Statements ;  RETURN Variable\_Or\_Value\_Of\_Type\_Datatype1;  EXCEPTION  Exception\_Definition ;  END Func\_Name ; | CREATE FUNCTION crt\_dept  (dno number, dname varchar2)  RETURN number AS  BEGIN  INSERT into department\_master  VALUES (dno,dname) ;  return 1 ;  EXCEPTION  WHEN others THEN  return 0 ;  END crt\_dept ;  **EXECUTION:**  variable flag number  EXECUTE :flag:=crt\_dept(60,‘Production');  PRINT flag; |
| Package | Packages are PL/SQL constructs that allow related objects to be stored together. It consists of two parts, namely   * PackageSpecification * Package Body | CREATE **PACKAGE** package\_name AS  variable\_declaration ;  cursor\_declaration ;    FUNCTION func\_name  (Parameter datatype ,..)  return datatype1 ;    PROCEDURE proc\_name  (Parameter {in| out| in out} datatype...);  END package\_name ;  CREATE **PACKAGE BODY** package\_name AS  variable\_declaration ; cursor\_declaration ;  PROCEDURE proc\_name(parameter  {IN|OUT|INOUT} datatype,...} IS  BEGIN  PL/SQL\_Statements ;  END proc\_name ;  FUNCTION func\_name(parameter datatype,...) is  BEGIN  PL/SQL\_Statements ;  END func\_name ;  END package\_name; | CREATE OR REPLACE PACKAGE pack1 AS  PROCEDURE proc1;  FUNCTION fun1 return varchar2;  END pack1;  CREATE OR REPLACE PACKAGE BODY pack1 AS  PROCEDURE proc1 is  BEGIN  dbms\_output.put\_line(‘hello from proc1’);  END proc1;  function fun1 return varchar2 is  BEGIN  return (‘hello from fun1’);  END fun1;  END pack1;  **EXECUTION:**  EXEC pack1.proc1  SELECT pack1.fun1 FROM dual; |

**SQL\* PLUS REPORTS**

There are six types of SQL\*Plus commands:

* + Commands that initiate the SQL\*Plus environment
  + SQL\*Plus execute commands - /, HELP, HOST, RUN, TIMING
  + SQL\*Plus editing commands – used to load, save, manipulate contents
  + SQL\*Plus formatting commands - BREAK ON, BTITLE(BOLD, CENTER, COL, FORMAT, LEFT, RIGHT, SKIP,TAB), COLUMN(FORMAT, HEADING, JUSTIFY, NEWLINE, NEW\_VALUE, NOPRINT)
  + Miscellaneous commands – ACCEPT, DEFINE, DESC, PAUSE, PROMPT, REMARK, SET, SPOOL, UNDEFINE
  + Access commands for various databases

**USING COMMANDS:**

COLUMN LAST\_NAME HEADING 'LAST NAME'

COLUMN SALARY HEADING 'MONTHLY SALARY'

COLUMN COMMISSION\_PCT

HEADING COMMISSION SELECT LAST\_NAME, SALARY, COMMISSION\_PCT

FROM EMP\_DETAILS\_VIEW

WHERE JOB\_ID='SA\_MAN';

**SQL\* LOADER**

SQL\*Loader

SQL\*Loader is a bulk loader utility used for moving data from external files into the Oracle database.

**Data**

* + 10001,"Scott Tiger", 1000, 40
  + 10002,"Frank Naude", 500, 20

**Control file**

The SQL\*Loader Control file is the text file which is a key to any load process.

Eg: load data

infile 'c:\data\mydata.csv'

into table emp fields terminated by ","

optionally enclosed by '"' ( empno, empname, sal, deptno )

**Log file**

The log file is a record of SQL\*Loader’s activities during a load session.

**Bad file**

Whenever SQL\*Loader encounters a database error while trying to load a record, it writes that record to a file known as the “Bad file”.

**Discard file**

Records that do not meet the specified criteria are not loaded, and are instead written to a file known as the “Discard file”.

**Invoking SQL\*Loader**

–sqlldr

–sqlldr keyword=value [keyword=value ...]

–sqlldr value [value ...]

Eg: C:\sqlldr scott/tiger@oracle9i control=d:\custcontrol.ctl

**BULK COLLECT & BULK BIND**

**ADVANCE PL/SQL:**

**Context Switching:**

PL/SQL: Embeded in RDBMS accepts any valid PL/SQL block

PL/SQL engine- PL/SQL block-procedural statement executor-sql-sql statement executor(data)

**Loop Overhead:**

Context switch between PL/SQL and SQl engines adds to the overhead.

**Binding:**

Binding is known as assigining of values to PL/SQL variables in SQL statements(In-Bind, Out-Bind)

**Bulk Binding:**

Bulk Binding is known as binding of entire collection at once.

**FORALL Statements:**

Instructs PL/SQL engine to bulk-bind input collections before sending them to SQl engine with an INSERT/DELETE/UPDATE.

**Bulk Collect Clause:**

It tells the SQL engine to bulk bind output collections before returning them to PL/SQl statements.

**CODING STANDARDS**

**Coding Standards:** coding principles, coding with handlers, coding with tiggers, coding with SQl, coding with PL/SQL.

**Basic Of Coding Standards:**

Naming conventions in coding are important for standard rules for indentation, code formatting, comments and white space.

**Coding Principles:**

-redable, avoid using other coding languages.

-fast performance over the web is critical, avoid platform-specific code, reusable object should be employee.

**Coding With Handlers:**

**-**Uses groups of packaged procedures to develop, maintain and debug**.**

**Coding With Triggers:**

-Triggers names must be unique eg: a table and a trigger can have same name.

Note: triggers name do not need to be unique with respective to other schema objects – tables, views and procedures.

**Coding with SQL:**

-While creating SQL code-CREATE, ALTER, SELECT, WHERE and use “\_”notation for columns and fields.

**Coding with PL/SQL:**

**-**Similar to SQL it makes sense to combine the standards for both in one place.