PL/SQL

Procedural Language/Structured Query Language

- PL/SQL is an extension of the SQL language.
- It eliminates many restrictions of the SQL Language.
- PL/SQL extends SQL by adding conditional statements, looping statements, user defined functions and procedures, exceptions etc. found in the other procedural languages.
- Procedural constructs blend seamlessly with SQL, resulting in a structured, powerful language.

PL/SQL combines the SQL's language's ease of data manipulation and the procedural language's ease of programming.

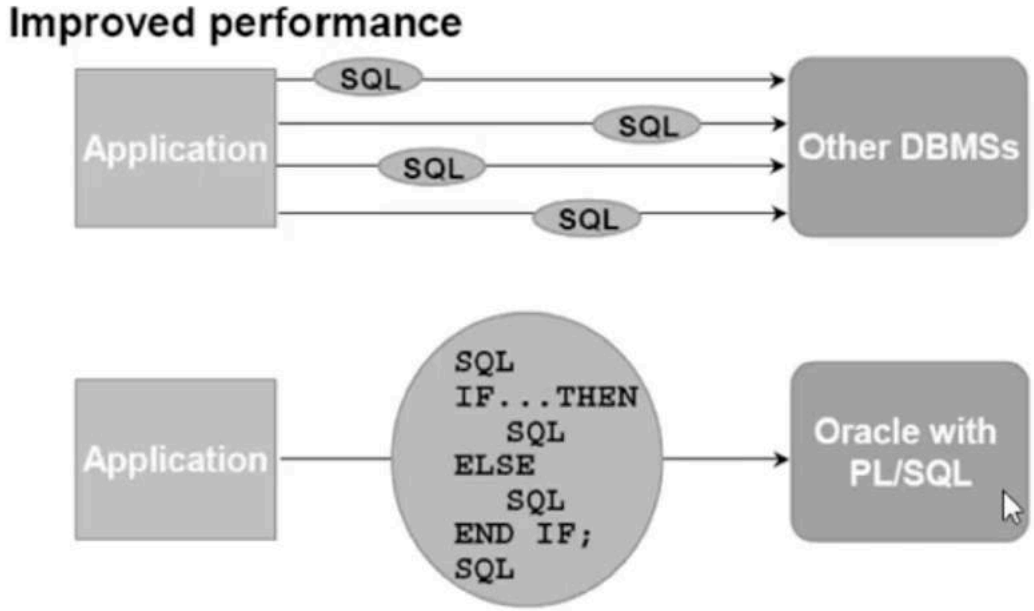
Parteek Bhatia: Simplifying...
SUBSCRIBED

Benefits of PL/SQL

- Improved Performance

  In PL/SQL it sends the bundle of SQL statements to the oracle server in the form of BLOCK and hence calling the server resources only once.

  After processing all the statements in a block server closes the resources results in faster execution.



Benefits of PL/SQL

Modularize program development

```
DECLARE
  • • •
BEGIN
  • • •
EXCEPTION
  • • •
END;
```

Features of PL/SQL

- Supports the declaration and manipulation of variables and constants.
- Exception Handling
- Allows the creation and calling of user defined functions and procedures.

Features of PL/SQL

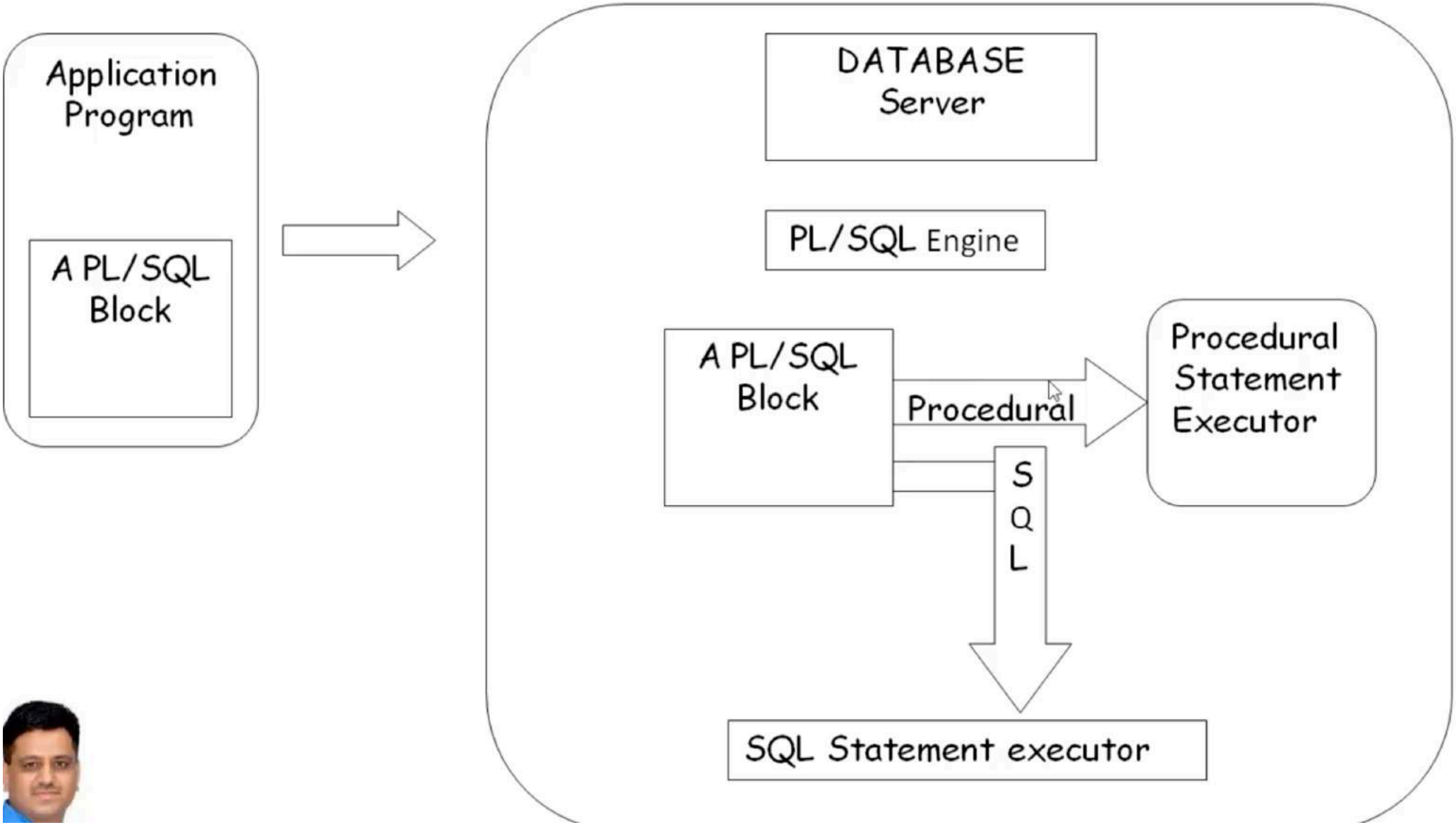
- Triggers (A code which automatically got executed when some event occurs)
- Cursors (Used to process set of rows)
- Support for SQL
- Support for Object-Oriented Programming

SQL Vs. PL/SQL

SQL	PL/SQL
SQL does not have any procedural capabilities. It means that there is no provision of conditional checking, looping and branching, which are very essential for filtration of data before entering it into database.	PL/SQL has procedural capabilities to support data filtration.
SQL statements are passed to oracle engine one at a time and hence generating network traffic resulting in slow processing.	In PL/SQL it sends the bundle of SQL statements to the oracle server in the form of BLOCK and hence calling the server resources only once and results in faster execution of SQL statements in a PL/SQL block.

SQL Vs. PL/SQL

SQL	PL/SQL
There is no provision of handling errors and exceptions, which means that if any SQL statement fails to execute, then it gives its own error messages which may not be user friendly.	In PL/SQL, we can program the block of statements to handle the errors using exception block in such a way that if any of the statement fails to execute then we can display user-friendly appropriate messages.
SQL does not support PL/SQL statements.	PL/SQL supports SQL statements in its block.
It does not support variables and nstants, that means we cannot store intermediate results of a query in iables.	PL/SQL supports declaring the variables so as to store intermediate results of query for later use.


Application Program
A PL/SQL Block
DATABASE Server
PL/SQL Engine
A PL/SQL Block
Procedural
Procedural Statement Executor
SQL
SQL Statement executor

PL/SQL Language Elements

- Identifiers
- Literals
- Operators
- Comments
- Expressions and comparisons
- Data types and Declarations

Identifiers

- Identifiers are named conventions used for variables, constants and oracle objects like tables, procedure, functions, packages, trigger and cursors etc.

Literals

- Literals specify an exact value in a program
- It is an explicit numeric, character, string or boolean value not represented by an identifier.
- Literals are used to initialize constants or variables.

Example:

first_number:=3;

Here, first_number is identifier and 3 is literal;

Operators

Arithmetic Operators	Example	Description
+	2+5; x+y;	Adds two operands
-	8-2; x-y	Subtracts the second operand from the first
*	2*5; x*y	Multiplies two operands
/	12/2; x/y;	Divides the first operand by the second
**	3**2;x**2;	Raises the first operand to the exponent of the second

Operators

Operators	Example	Description
:=	x:=y;a:=b*c; a:=3;	Assigns the value of the operand or expression on the right to the variable or constant on the left.
..	1..5; 4..d ; 4..d-1;	Defines an integer range
||	'kit'||'kat'; x||y	Concatenates two or more strings.

Comments

- Comments can be single line or multiple lines.

Single line comment:

- Begins with -- can appear within a statement, at end of line.

Example:

x number; --variable declaration

Multi line comment

- Begin with /* and end with an asterisk-slash (*/).

Example

/* Statements to select rate and quantity into variables and calculate value */

Expressions and Comparisons

- Expressions are constructed using operands and operators. An operand is a variable, constant, literal or function call that contributes a value to an expression.
- In PL/SQL operators can be unary that operate on one operand and binary that operate on two or more operand. An expression always produces a single value of a specific data type.

Data Types

- NUMBER
- CHAR
- VARCHAR
- DATE
- BOOLEAN

Structure of PL/SQL Block

By
Dr. Parteek Bhatia
Associate Professor
Department of Computer Science and Engineering
Thapar Institute of Engineering & Technology,
Patiala, India

PL/SQL Block

Modularize program development

```
DECLARE
  • • •
BEGIN
  • • •
EXCEPTION
  • • •
END;
```

PL/SQL Block

Declare

-used to declare variables and constants

-is also used to declare user defined procedures and functions

-is an optional section

Begin

Exception

---

d;

PL/SQL Block

Declare

Begin

-is the executable section containing the code, the whole logic is written in this section

-is compulsory section

Exception

---

d;

PL/SQL Block

Declare

Begin

Exception

-code contained in this section is executed only when an error occurs

-handles exceptions occurring during processing

-used to place predefined or user defined exceptions

-is an optional section

d;

PL/SQL Block

Declare

-Declare variables and constants and optional

Begin

-Process PL/SQL statements and compulsory section

Exception

-Error handlers and optional

End;

Declaration of Variables and Constants

variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]

Declaration of Variables

```
declare
age   number (4) ;
done boolean ;
begin
------
end;
```

Variable Value Assignment

Three ways:

i) With Assignment operator

```
a : = b * c ;
increase : = sal * 1.5 ;
OK : = false ;
```

Variable Value Assignment

ii) With substitute variables

a:=&first_number;

The system will prompt to enter the value of substitution variable first_number and its value will be substituted.

enter the value of first_number:3

a:=3;

b:=&b;

--Not supported in OracleLive only allowed in iSQLPLUS

Variable Value Assignment

iii) From database

SELECT INTO statement …

For example:

SELECT ENAME INTO E FROM EMP WHERE ENO=100;

Constant Declaration

- Constants are named elements whose values do not change.
- For example, pi can be declared as a constant whose value 3.14 is never changed in the PL/SQL block.

Constant Declaration

- The declaration of a constant is similar to that of declaration of a variable.
- To declare the constant we must make use of the keyword constant. This keyword must precede the data type as shown below.
- Example

pi constant number : = 3.14;

Variable Value Assignment

iii) From database

SELECT INTO statement …

For example:

SELECT ENAME INTO E FROM EMP WHERE ENO=100;

Inbox (26,214) - parteek.bha | Inbox (38,571) - parteek.bha | Oracle Live SQL - SQL V

https://livesql.oracle.com/apex/f?p=590:1:16750434857570::NO:::

Live SQL | Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
insert into emp values(1,'ram','prof',300000)
insert into emp values(2,'sham', 'assoc prof', 200000)
select * from emp
declare
n char(30);
j char(20);
begin
select name,job into n,j from emp where eno=2;
dbms_output.put_line('Welcome to PL/SQL');
dbms_output.put_line('Name is '||n);
dbms_output.put_line('Job is '||j);
end;
```

```
Welcome to PL/SQL
Name is sham
Job is assoc prof
```

https://livesql.oracle.com/apex/f?p=590:1:16750434857570::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
select * from emp
declare
n char(30);
j char(20);
begin
select name,job into n,j from emp where eno=20;
dbms_output.put_line('Welcome to PL/SQL');
dbms_output.put_line('Name is '||n);
dbms_output.put_line('Job is '||j);
end;
```

```
ORA-01403: no data found ORA-06512: at line 5
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
4  select * from emp
5  declare
6  n char(30);
7  j char(20);
8  begin
9  select name,job into n,j from emp ;
10 dbms_output.put_line('Welcome to PL/SQL');
11 dbms_output.put_line('Name is '||n);
12 dbms_output.put_line('Job is '||j);
13 end;
14
15
```

```
ORA-01422: exact fetch returns more than requested number of rows ORA-06512: at line 5
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Variable Value Assignment

Note:

- If SELECT INTO statement return no row then PL/SQL block will return an error no data found.
- If SELECT INTO statement return more than one row then PL/SQL block will return an error too many rows.
- So, SELECT INTO statement should return only one row.

Variable Attributes

Attributes allow us to refer to data types and objects from the database. PL/SQL variables and constants can have attributes.

- %TYPE
- %ROWTYPE

https://livesql.oracle.com/apex/f?p=590:1:16750434857570::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
select * from emp

alter table emp modify (name char(60));

declare
n emp.name%type;
j emp.job%type;
begin
select name,job into n,j from emp where eno=1;
dbms_output.put_line('Welcome to PL/SQL');
dbms_output.put_line('Name is '||n);
dbms_output.put_line('Job is '||j);
end;
```

```
Welcome to PL/SQL
Name is ram
Job is prof
```

Variable Attributes

Attributes allow us to refer to data types and objects from the database. PL/SQL variables and constants can have attributes.

- %TYPE
- %ROWTYPE

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
e emp.eno%type;
n emp.name%type;
j emp.job%type;
s emp.sal%type;
begin
select eno,name,job,sal into e,n,j,s from emp where eno=1;
dbms_output.put_line('Welcome to PL/SQL');
dbms_output.put_line('Eno is '||e);
dbms_output.put_line('Name is '||n);
dbms_output.put_line('Job is '||j);
dbms_output.put_line('Sal is '||s);
end;
```

```
Name is ram
Job is prof
Sal is 300000
```

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

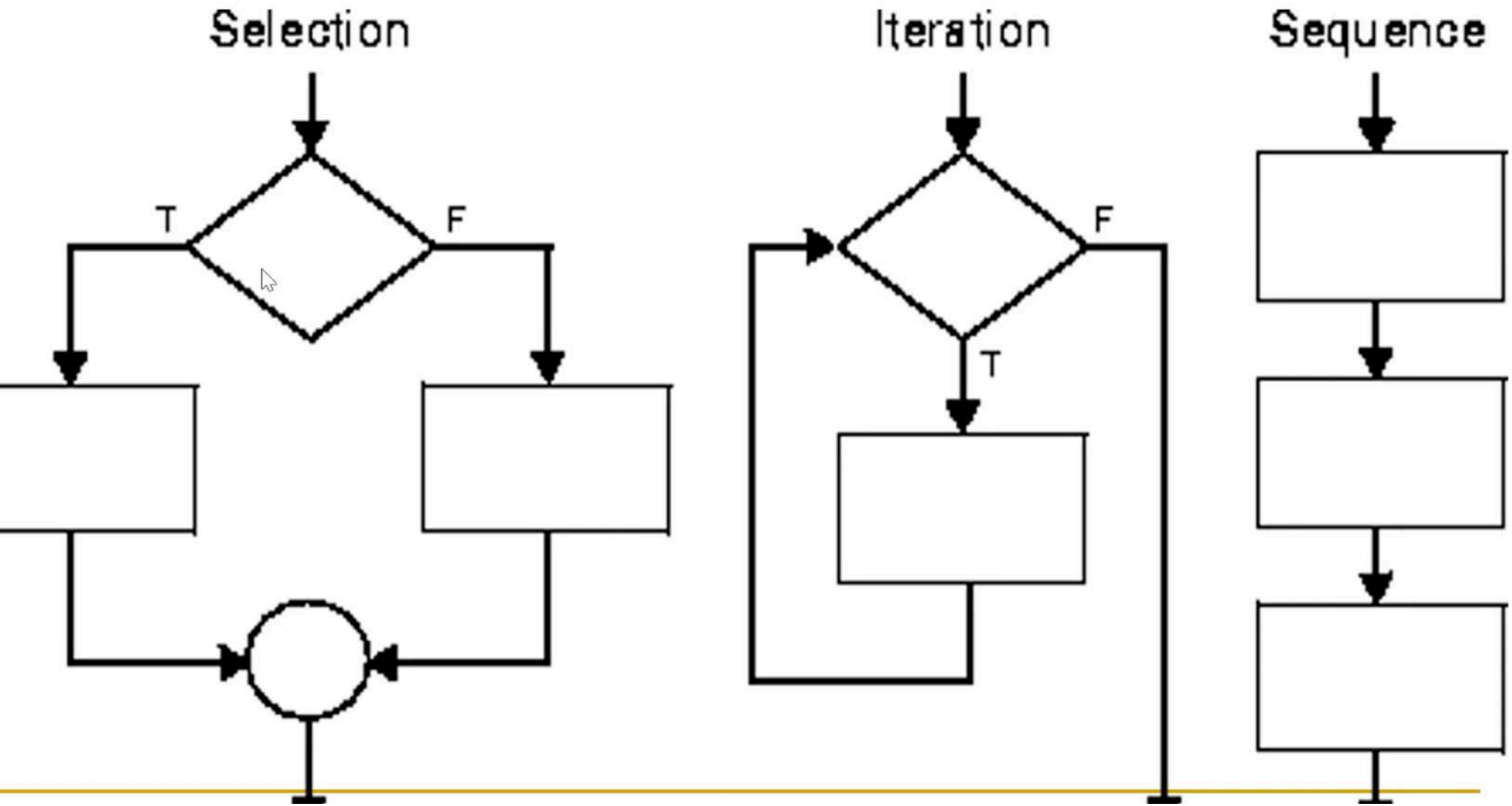
```
alter table emp modify (name char(60));

declare
rec emp%rowtype;
begin
select * into rec from emp where eno=1;
dbms_output.put_line('Welcome to PL/SQL');
dbms_output.put_line('Eno is '||rec.eno);
dbms_output.put_line('Name is '||rec.name);
dbms_output.put_line('Job is '||rec.job);
dbms_output.put_line('Sal is '||rec.sal);
end;
```

```
Name is ram
Job is prof
Sal is 300000
```

19:36

Control structures of PL/SQL

- Conditional Control
- Iterative Control
- Sequential Control


Selection
T
F
Iteration
F
T
Sequence

Conditional Control

- IF-THEN END IF;STATEMENT
- IF- THEN – ELSE – END IF; STATEMENT
- IF-THEN-ELSIF STATEMENT (LADDER IF)

Conditional Control

```
IF a > b THEN
  dbms_ouput.put_line('a is greater');
END IF;

IF A > B THEN
DBMS_OUTPUT.PUT_LINE(' A IS GREATER ');
ELSE
DBMS_OUTPUT.PUT_LINE(' B IS GREATER ');
END IF;
```

IF-THEN-ELSIF STATEMENT (LADDER IF)

```
IF CONDITION THEN
-----
ELSIF CONDITION  THEN
-----
ELSIF CONDITION  THEN
-----
ELSIF CONDITION  THEN
-----
ELSE
-----
END IF;
```

SQL Worksheet

Clear Find Actions

```
DECLARE
        A       NUMBER := 3;
        B       NUMBER := 9;
BEGIN
        IF A > B THEN
                DBMS_OUTPUT.PUT_LINE(' A IS GREATER ');
        ELSE
                DBMS_OUTPUT.PUT_LINE(' B IS GREATER ');
        END IF;
END;
```

```
Statement processed.
 A IS GREATER
```

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
        B          NUMBER := 3;
BEGIN
        IF A=B THEN
                DBMS_OUTPUT.PUT_LINE(' NUMBERS ARE EQUAL ');
        ELSE
        IF A > B THEN
                DBMS_OUTPUT.PUT_LINE(' A IS GREATER ');
        ELSE
                DBMS_OUTPUT.PUT_LINE(' B IS GREATER ');
        END IF;
        END IF;
END;
```

```
Statement processed.
 NUMBERS ARE EQUAL
```

Live SQL

Feedback Help parteek.b

SQL Worksheet

Clear Find Actions

```
DECLARE
A NUMBER:=3;
B NUMBER:=2;
C NUMBER;
X NUMBER;
BEGIN
X:=2;
IF X=1 THEN
    C:=A+B;
ELSIF X=2 THEN
C:=A-B;
ELSIF X=3 THEN
C:=A*B;
ELSIF X=4 THEN
```

```
Statement processed.
RESULT IS 1
```

SQL Worksheet

Clear Find Actions

```
DECLARE
A NUMBER:=3;
B NUMBER:=2;
C NUMBER;
X NUMBER;
BEGIN
X:=41;
IF X=1 THEN
    C:=A+B;
ELSIF X=2 THEN
C:=A-B;
ELSIF X=3 THEN
C:=A*B;
ELSIF X=4 THEN
```

```
Statement processed.
NOT A VALID OPTION
RESULT IS
```

Iterative Control

- Simple Loop Statement
- While Loop Statement
- For Loop Statement

EXIT and EXIT WHEN Statements

- We can use an EXIT statement to complete the loop.

There are two forms of EXIT statements:

- EXIT
- EXIT WHEN

EXIT Statement

It is commonly used with EXIST statement.

Example:

```
LOOP
   ...
            IF c> 5 THEN
                  EXIT;  -- exit loop immediately
            END IF;
END LOOP;
-- Control resumes here
```

EXIT-WHEN Statement

- The EXIT-WHEN statement allows a loop to complete conditionally. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition yields TRUE, the loop completes and control passes to the next statement after the loop.

Example

- An example follows:

```
LOOP
        ------
        EXIT WHEN c>5;  -- exit loop if condition is true   ...
END LOOP;
```

Example

IF c > 5 THEN	EXIT WHEN c > 5;
EXIT;	
END IF;	

WAP TO PRINT NUM FROM 1 TO N

SQL Worksheet

Clear Find Actions Save Run

```
DECLARE
        VAR1    NUMBER  :=1;
        N       NUMBER  :=10;
BEGIN
        LOOP
            DBMS_OUTPUT.PUT_LINE( VAR1);
            VAR1 := VAR1+1;
            EXIT WHEN VAR1 > N ;
        END LOOP;
END ;
```

```
8
9
10
```

WHILE-LOOP

```
WHILE condition LOOP
  sequence_of_statements;
END LOOP;
```

WHILE-LOOP

```
WHILE i <= 10 LOOP
   --
   i:=i+1;
END LOOP;
```

WAP TO PRINT NUM FROM 1 TO N

SQL Worksheet

Clear | Find | Actions | Save | Run

```
DECLARE
        VAR1    NUMBER  :=1;
        N       NUMBER  :=10;
BEGIN
        WHILE VAR1<=N LOOP
            DBMS_OUTPUT.PUT_LINE( VAR1);
            VAR1 := VAR1+1;
        END LOOP;
END ;
```

Multiplication table of any Number

```
DECLARE
        TABLE_OF NUMBER:=5;
        C NUMBER := 1;
        RESULT NUMBER;
BEGIN
        WHILE C <= 10 LOOP
                RESULT := TABLE_OF * C ;
                DBMS_OUTPUT.PUT_LINE(TABLE_OF||' * '|| C ||'='||RESULT);
                C := C +1 ;
        END LOOP;
END;
```

FOR-LOOP Statement

```
FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements;
END LOOP;
```

FOR-LOOP Statement

```
FOR i IN 1..3 LOOP  -- assign the values 1,2,3 to i
   sequence_of_statements;  -- executes three times
END LOOP;
```

Features of For Loop

- No need to declare counter variable in declare section, it automatically declares counter variable of number type.

```
FOR i IN 1..3 LOOP
   ---
   ---
END LOOP;
```

Features of For Loop

- The scope of this counter variable is limited to body of For Loop

```
FOR i IN 1..3 LOOP
    ---
    ---
END LOOP;
```

Features of For Loop

- There is no need to increment or decrement this counter variable, it is automatically incremented or decremented by 1 after completion of a cycle.

```
FOR i IN 1..3 LOOP
   ---
   ---
END LOOP;
```

Features of For Loop

- We cannot change the value of this counter variable, but we can use its value.

```
FOR i IN 1..3 LOOP
   ---
    i:=i+1;--not allowed
    i:=2;--not allowed
    s:=i+2;--allowed
END LOOP;
```

What will the output in this case

```
FOR C IN 3..3 LOOP
 DBMS_OUTPUT.PUT_LINE(C);
END LOOP;
```

Output is: 3

```
FOR C IN REVERSE 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE(C);
END LOOP;
```

Output is:
3
2
1

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
DECLARE
    TABLE_OF NUMBER:=8;
    RESULT NUMBER;
BEGIN
        FOR C IN 1..10 LOOP
        RESULT := TABLE_OF * C ;
        DBMS_OUTPUT.PUT_LINE(TABLE_OF||' * '|| C ||' = '||RESULT);
        END LOOP;
END;
```

```
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

16:52

To design a for loop that execute for multiple of n

For example

Code that run for 5, 10, 15, 20, 25 and so on

Or

Code that run for 3, 6, 9, 12, 15 and so on

Live SQL

Feedback Help parteek.bhatia@gmail.

SQL Worksheet

Clear Find Actions Save

```
BEGIN
        FOR j IN 5..45 LOOP
            IF MOD(j, 5) = 0 THEN       -- pass multiples of 5       sequence_of_statem
            DBMS_OUTPUT.PUT_LINE(J);

            END IF;
        END LOOP;
END;
```

```
Statement processed.
5
10
15
```

Sequential Control

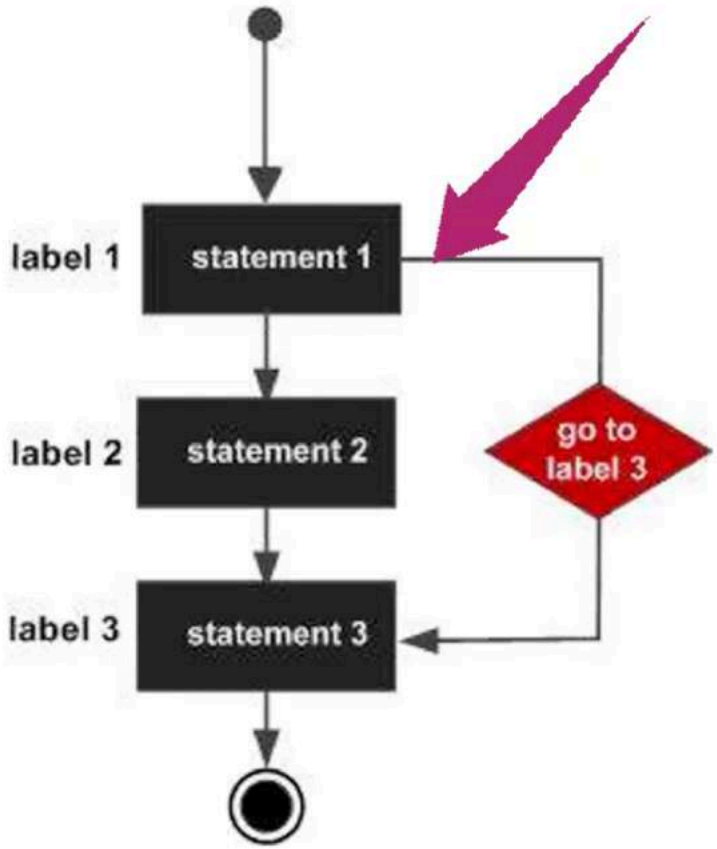
- GOTO statement
- NULL statement

GOTO statement

- The GOTO statement branches to a label unconditionally.
- When executed, the GOTO statement transfers control to the labeled statement.

- Syntax

```
GOTO label;
..
..
<< label >>
statement;
```



The label must be unique within its scope.

Sequential Control

GOTO Statement

```
BEGIN
...
GOTO insert_row;
.............
<<insert_row>>
INSERT INTO emp VALUES
...
END;
```

Restriction with GOTO statement

```
BEGIN
GOTO ABC;
        IF A>B THEN
        ..
        <<ABC>>
        ..
        END IF;
END;
```

GOTO statement cannot branch into IF statement.

Restriction with GOTO statement

```
BEGIN
...
        IF A>B THEN
        ..
        GOTO ABC;
        ...
        ELSE
        ..
        <<ABC>>
        ..
        END IF;
END;
```

GOTO statement cannot branch into IF statement one clause to another.

Restriction with GOTO statement

```
BEGIN
GOTO ABC;
        WHILE A<10 LOOP
        ..
        <<ABC>>
        ..
        END LOOP;
END;
```

GOTO statement cannot branch into LOOP statement.

Restriction with GOTO statement

```
LOOP
....
IF done THEN
GOTO end_loop;
END IF;
--
<<end_loop>>  -- illegal
END LOOP;  -- not an
   --executable statement
```

- The GOTO statement must be followed by an executable statement.

Null Statement

```
LOOP
…
IF done THEN
GOTO end_loop;
END IF;
--
<<end_loop>>  -- illegal
END LOOP;  -- not an executable
    --statement
END;
```

Solution is:

```
LOOP
…
IF done THEN
GOTO end_loop;
END IF;
--
<<end_loop>>  -- illegal
NULL;  -- an executable statement
END LOOP;  -- not an executable statement
END;
```

Null Statement

```
BEGIN
   IF condition THEN
    ........
    ......
  ELSE
   NULL;
  END IF;
END;
```

- Also used to improve readability of code.

Live SQL

Feedback Help parteek.bhatia@gmail.com

Worksheet

Clear Find Actions Save Run

```
F X=1 THEN
    C:=A+B;
LSIF X=2 THEN
:=A-B;
LSIF X=3 THEN
:=A*B;
LSIF X=4 THEN
:=A/B;
LSE
BMS_OUTPUT.PUT_LINE('NOT A VALID OPTION');
OTO SKIP;
ND IF;
BMS_OUTPUT.PUT_LINE('RESULT IS ' || C);
<SKIP>>
ND;
```

NO NULL

```
0: line 22, column 1:
3: Encountered the symbol "END" when expecting one of the following:
```

Live SQL

Feedback Help parteek.bhatia@gma

SQL Worksheet

Clear Find Actions Save

```
IF X=1 THEN
    C:=A+B;
ELSIF X=2 THEN
C:=A-B;
ELSIF X=3 THEN
C:=A*B;
ELSIF X=4 THEN
C:=A/B;
ELSE
DBMS_OUTPUT.PUT_LINE('NOT A VALID OPTION');
GOTO SKIP;
END IF;
DBMS_OUTPUT.PUT_LINE('RESULT IS ' || C);
<<SKIP>>
NULL;
END;
```

NULL

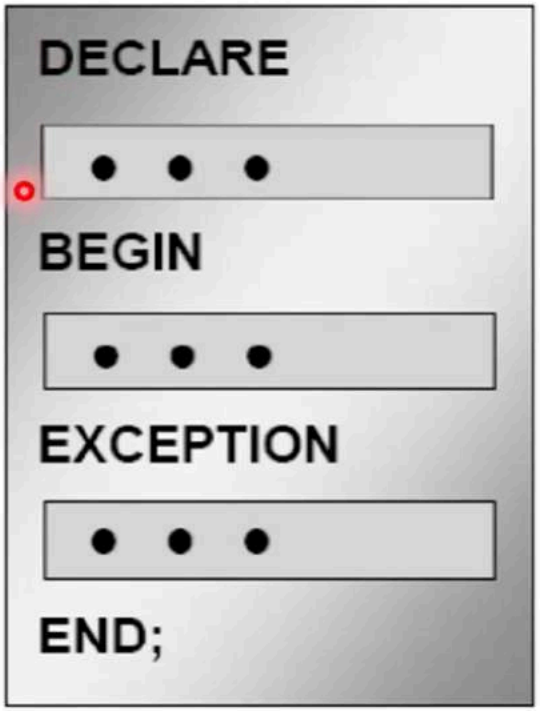
```
Statement processed.
NOT A VALID OPTION
```

Exception Handling

- There are numbers of reasons due to which run time errors may be raised during the execution of a PL/SQL block.
- Although we cannot anticipate all possible errors, but we can plan to handle certain kinds of errors meaningful to our PL/SQL program.
- With PL/SQL, a mechanism called exception handling allows us to handle run times errors smoothly without abruptly terminating the code or without througing error to user or calling environment.

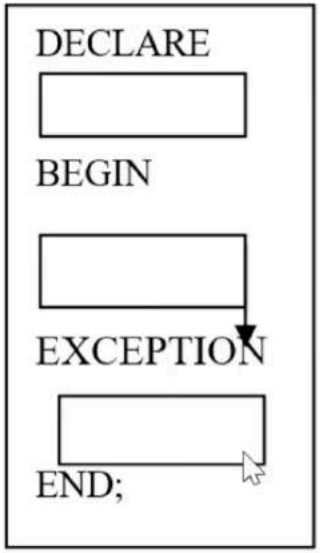
Exception Handling

- When an error occurs, an exception is raised; normal execution is stopped and control is transferred to an exception-handling section of PL/SQL program.
- A specific section can be defined in the program to handle exceptions.



Trapping of the Exception

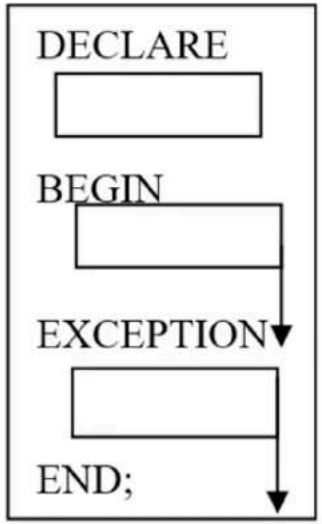
Trap the exception



Exception is raised

Exception is trapped

Propagate the execption



Exception is raised

Exception is not trapped

Exception propagates to calling environment

Syntax of Handling Exceptions

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

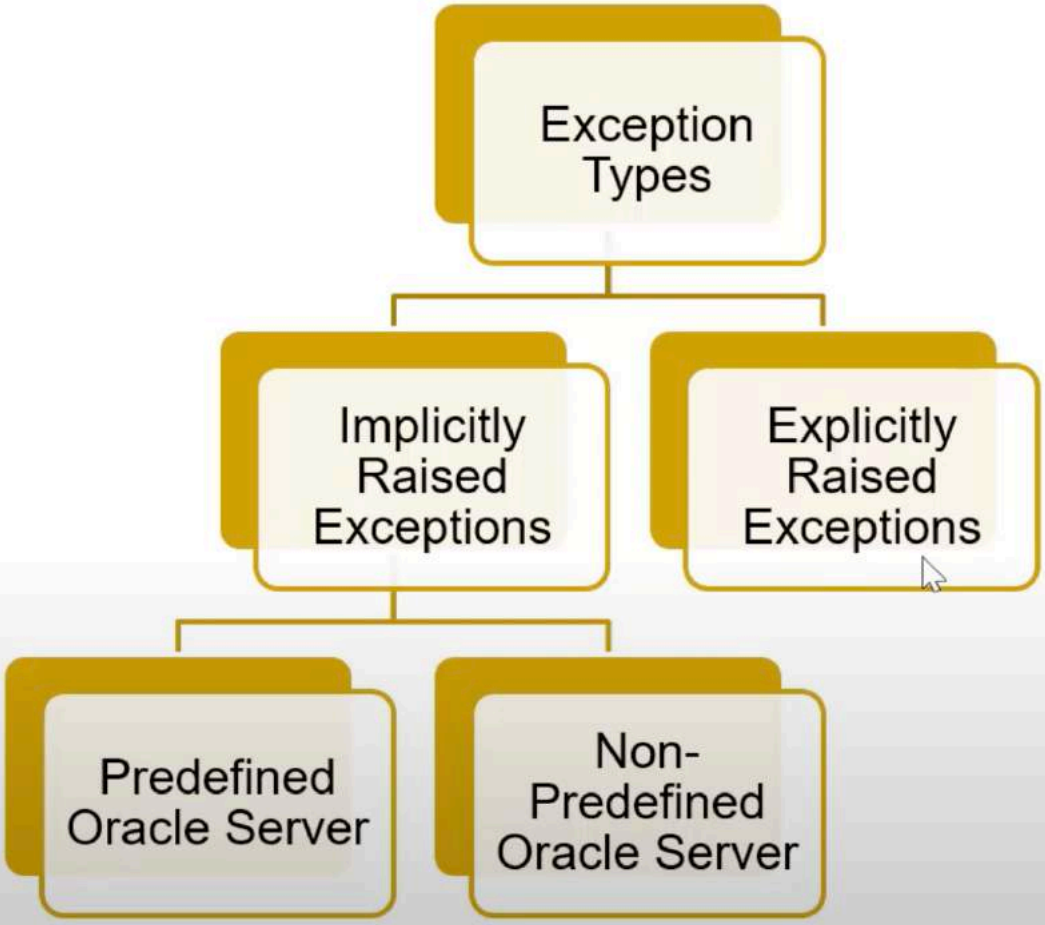
Syntax of Handling Exceptions

```
EXCEPTION
WHEN  exception1 THEN
statement1;
statement2;
WHEN  exception2 THEN
statement1;
statement2;
...............
WHEN OTHERS THEN
statement1;
statement2;
        ........
END;
```

Syntax of Handling Exceptions

```
EXCEPTION
WHEN OTHERS THEN
statement1;
statement2;
        ........
END;
```

Exception Types



Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
create table emp(eno number(3) primary key, name char(30), job char(20), sal number(5));
insert into emp values(1,'ram','prof',90000);
insert into emp values(2,'sham','assoc prof',70000);
DECLARE
N EMP.NAME%TYPE;
BEGIN
SELECT NAME INTO N FROM EMP WHERE ENO=11;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
END;
```

```
ORA-01403: no data found ORA-06512: at line 4
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Some of Predefined Oracle Server Errors

Exception	Error	Raised if
DUP_VAL_ON_INDEX	ORA-00001	Attempt is made to INSERT or UPDATE duplicate values in a UNIQUE database column.
INVALID_NUMBER	ORA-01722	The conversion of a character string to a number fails in a SQL statement.
NO_DATA_FOUND	ORA-01403	A SELECT INTO returns no rows, or you refer to an uninitialized row in a PL/SQL table.
TOO_MANY_ROWS	ORA-01422	A SELECT INTO returns more than one row.
VALUE_ERROR	ORA-06502	The conversion of a character string to a number fails in a procedural statement, or an arithmetic, conversion, truncation, or constraint error occurs.
ZERO_DIVIDE	ORA-01476	An attempt is made to divide a number by zero.

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.

SQL Worksheet

Clear Find Actions Save

```
create table emp(eno number(3) primary key, name char(30), job char(20), sal number(5));
insert into emp values(1,'ram','prof',90000);
insert into emp values(2,'sham','assoc prof',70000);
DECLARE
N EMP.NAME%TYPE;
BEGIN
SELECT NAME INTO N FROM EMP ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
END;
```

s://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

```
ORA-01422: exact fetch returns more than requested number of rows ORA-06512: at line 4
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
insert into emp values(2,'sham','assoc prof',70000);
DECLARE
N EMP.NAME%TYPE;
BEGIN
SELECT NAME INTO N FROM EMP WHERE ENO=3 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
END;
```

```
Statement processed.
SOME RUN TIME ERROR OCCURS PLEASE CHECK
```

s://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@

SQL Worksheet

Clear Find Actions Save

```
N EMP.NAME%TYPE;
A NUMBER(3);
BEGIN
A:=3/0;
SELECT NAME INTO N FROM EMP WHERE ENO=1 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
END;
```

tps://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

```
Statement processed.
SOME RUN TIME ERROR OCCURS PLEASE CHECK
```

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@g

SQL Worksheet

Clear Find Actions Save

```
BEGIN
A:=3/0;
SELECT NAME INTO N FROM EMP WHERE ENO=1 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('ZERO DIVIDE OCCURS');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
END;
```

s://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

```
Statement processed.
ZERO DIVIDE OCCURS
```

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
DECLARE
N EMP.NAME%TYPE;
A NUMBER(3);
BEGIN
insert into emp values(2,'sham','assoc prof',70000);
--A:=TO_NUMBER('A2');
SELECT NAME INTO N FROM EMP WHERE ENO=1 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
```

```
ORA-00001: unique constraint (SQL_PBTQEPFHMWGDEZHJTHSFBJIAV.SYS_C0026971788) violated ORA-06512: at line 5
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

https://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

Oracle Live SQL - SQL V | Inbox (26,273) - parteek.bha | PL/SQL - Exceptions - Tutor | Web Filter Block Override

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL | Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
insert into emp values(2,'sham','assoc prof',70000);
DECLARE
N EMP.NAME%TYPE;
A NUMBER(3);
BEGIN
insert into emp values(3,'sham','assoc prof','70000');
--A:=TO_NUMBER('A2');
SELECT NAME INTO N FROM EMP WHERE ENO=1 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
```

ps://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

```
Statement processed.
INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT
```

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
insert into emp values(2,'sham','assoc prof',70000);
DECLARE
N EMP.NAME%TYPE;
A NUMBER(3);
BEGIN
insert into emp values(3,'sham','assoc prof','70000');
--A:=TO_NUMBER('A2');
SELECT NAME INTO N FROM EMP WHERE ENO=1 ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
```

```
Statement processed.
NAME IS ram
```

s://livesql.oracle.com/apex/f?p=590:1000:14555295760576::::

https://livesql.oracle.com/apex/f?p=590:1:14555295760576::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
DBMS_OUTPUT.PUT_LINE('INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
WHEN ZERO_DIVIDE THEN
DBMS_OUTPUT.PUT_LINE('ZERO DIVIDE OCCURS');
WHEN VALUE_ERROR THEN
DBMS_OUTPUT.PUT_LINE('CONVERSION TO NUMBER IS FAILED IN ASSIGNMENT');
WHEN INVALID_NUMBER THEN
DBMS_OUTPUT.PUT_LINE('Conversion of string to number failed');
--WHEN OTHERS THEN
--DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
END;
```

```
Statement processed.
Conversion of string to number failed
```

Trapping Predefined Oracle Server errors

- PL/SQL has a set of exceptions that are predefined for the common Oracle system errors. Predefined exceptions are used to detect and handle Oracle system errors that occur internally at program run time.

- Predefined Oracle Server errors are handled by name.

Example-1

```
DECLARE
        N EMP.NAME%TYPE;
BEGIN
        SELECT NAME INTO N FROM EMP WHERE ENO=1;
        DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO RECORD FOUND FOR ENO’);
        WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('QUERY RETURN MORE THAN ONE RECORD');
END;
```

Example-2

```
BEGIN
    INSERT INTO EMP VALUES(95,'sham','assoc prof',70000);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('INSERT STATEMENT IS VIOLATING UNIQUE CONSTRAINT’);
END;
```

Example-3

```
BEGIN
        insert into emp values(95,'sham','assoc prof','70000A');
EXCEPTION
        WHEN INVALID_NUMBER THEN
        DBMS_OUTPUT.PUT_LINE('Conversion of string to number failed');
END;
```

Example-4

```
DECLARE
        A NUMBER(3);
BEGIN
        A:=TO_NUMBER('A34’);
EXCEPTION
        WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('CONVERSION TO NUMBER IS
        FAILED IN ASSIGNMENT’);
END;
```

Example-5

```
DECLARE
        A NUMBER(3);
BEGIN
        A:=3/0;
EXCEPTION
        WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('ZERO DIVIDE    OCCURS');
END;
```

Trapping of Non-Predefined Oracle Server errors

- PL/SQL provides a limited set of predefined exceptions for the internal oracle system errors, but it provides a mechanism to declare user-defined exceptions to handle the other unnamed Oracle System errors.
- An exception variable can be defined and linked to the error using the PRAGMA EXCEPTION_INIT statement in a block's declarative section.

SIMPLIFIED APPROACH TO DBMS

Syntax

Declare

Exception_name EXCEPTION;

PRAGMA EXCEPTION_INIT (exception_name, error_number);

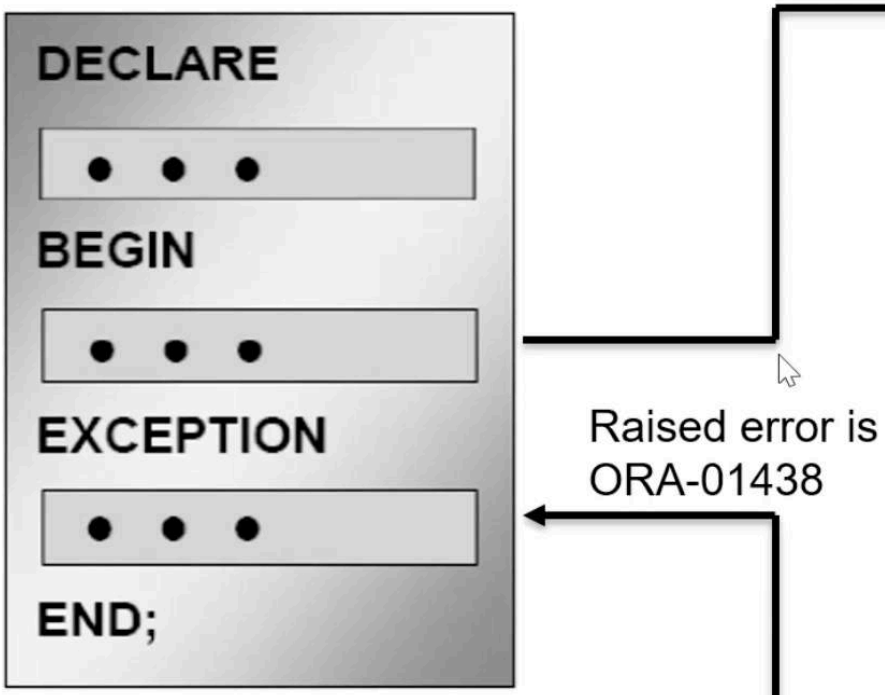
Trapping of Non-Predefined Oracle Server errors

Exception	Error
DUP_VAL_ON_INDEX	ORA-00001
INVALID_NUMBER	ORA-01722
NO_DATA_FOUND	ORA-01403
TOO_MANY_ROWS	ORA-01422
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476
LARGE_VALUE	ORA-01438

It is a Compiler Directive Statement
It defines specified exception name for given oracle error number.
It will applicable only for the PL/SQL block in which it is specified

```
PRAGMA EXCEPTION_INIT(LARGE_VALUE,-01438);
```

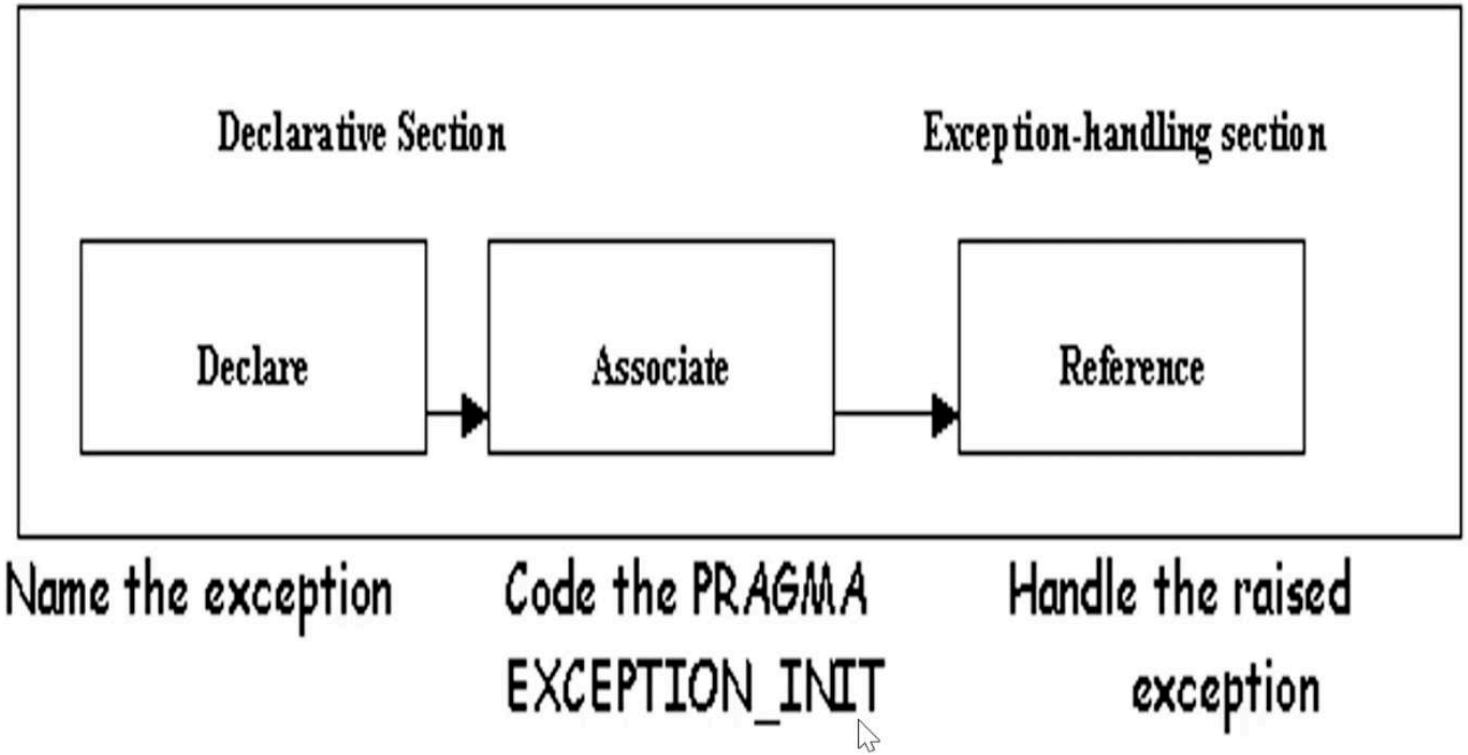
Trapping of Non-Predefined Oracle Server errors



Raised error is
ORA-01438

Exception	Error
DUP_VAL_ON_INDEX	ORA-00001
INVALID_NUMBER	ORA-01722
NO_DATA_FOUND	ORA-01403
TOO_MANY_ROWS	ORA-01422
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476
LARGE_VALUE	ORA-01438

Trapping of Non-predefined Oracle Server



Example

```
create table emp(eno number(3) primary key, name char(30), job char(20), sal
number(5));

BEGIN
insert into emp values(1,'ram','prof',900000);
END;
```

https://livesql.oracle.com/apex/f?p=590:1:1434179244480::NO:::

Live SQL

Feedback Help parteek.bhatia@gma

SQL Worksheet

Clear Find Actions Save

```
create table emp(eno number(3) primary key, name char(30), job char(20), sal number(5));
DECLARE
LARGE_VALUE EXCEPTION;
PRAGMA EXCEPTION_INIT(LARGE_VALUE,-01438);
BEGIN
insert into emp values(1,'ram','prof',900000);
EXCEPTION
WHEN LARGE_VALUE THEN
DBMS_OUTPUT.PUT_LINE('VALUE TOO BIG FOR COLUMN');
END;
```

```
Statement processed.
VALUE TOO BIG FOR COLUMN
```

Inbox (26,293) - parteek.bl | Inbox (38,752) - parteek.bl | Oracle Live SQL - SQL | Online Learning — An ant | Redirecting

https://livesql.oracle.com/apex/f?p=590:1:1434179244480::NO:::

Live SQL

Feedback | Help | parteek.bhatia@g

SQL Worksheet

Clear | Find | Actions | Save

```
PRAGMA EXCEPTION_INIT(LARGE_VALUE,-01438);
NUMBER_CONVERSION EXCEPTION;
PRAGMA EXCEPTION_INIT(NUMBER_CONVERSION,-06502);
A NUMBER(3);
BEGIN
A:=TO_NUMBER('45A');
insert into emp values(1,'ram','prof',900000);
EXCEPTION
WHEN NUMBER_CONVERSION THEN
DBMS_OUTPUT.PUT_LINE('CONVERSION TO NUMBER FAILS');
WHEN LARGE_VALUE THEN
DBMS_OUTPUT.PUT_LINE('VALUE TOO BIG FOR COLUMN');
END;
```

```
Statement processed.
CONVERSION TO NUMBER FAILS
```

Trapping User defined Exceptions

- User defined exceptions are used to handle error conditions specific to an application program.
- It is used to raise the run time error in case of violation of a business logic, so Oracle will not raise error in this case.
- Programmer has to raise it explicitly so it is called as Explicit Exception or User Defined Exception.

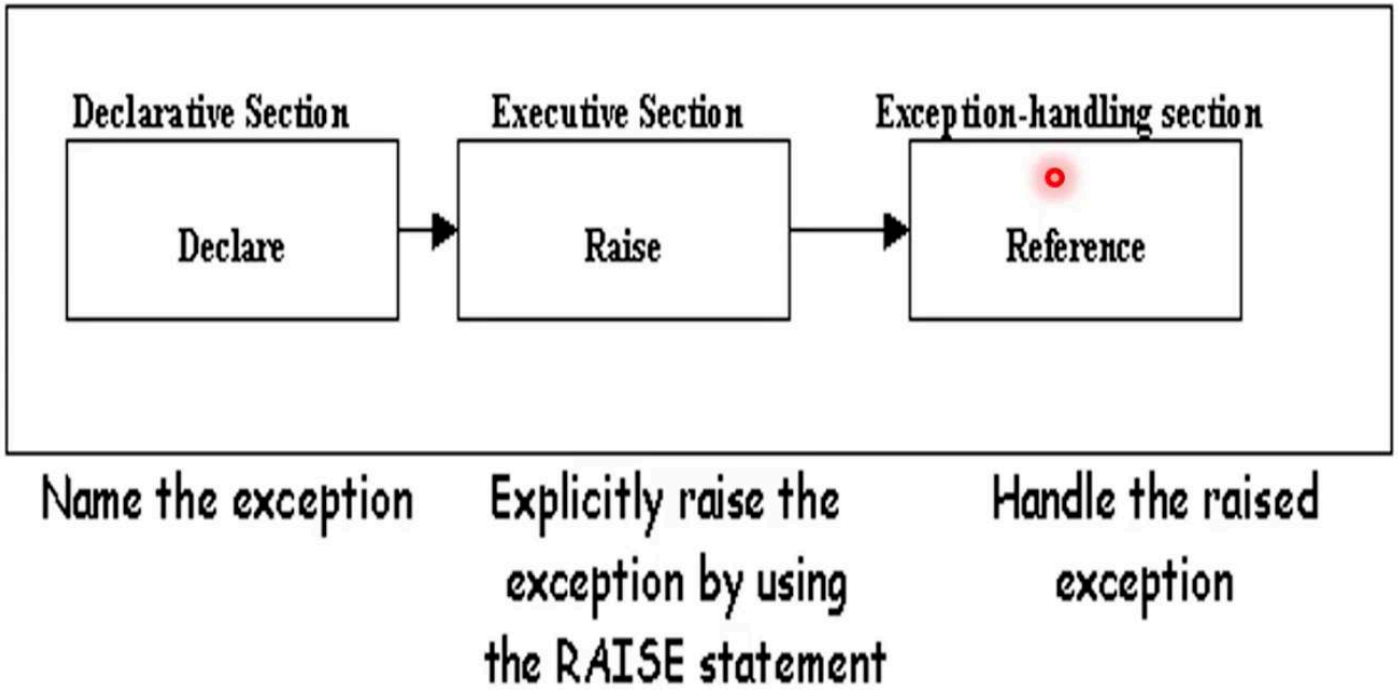
Trapping User defined Exceptions

- For example in case of Library system the date of issue of a book cannot be more than date of return, if that happens then an error must be raised explicitly by the application programmer and that facility is developed with the help of User defined exceptions.

Syntax

```
Declare
    Exception_name EXCEPTION;
Begin
    ...
    If condition Then
            RAISE  exception_name;
    End if;
    …
Exception
    When exception_name then
    ….
End;
```

Trapping of Explicitly raised or User defined exceptions



Example

Consider following tables

```
create table item
(item_no number(3) primary key, name char(30), rate number(3), qty_in_stock
number(5));

create table item_order
(order_no number(3) primary key, item_no number(3), name char(30), qty
number(3));
```

Example

Raise an exception when the quantity in stock is less than the threshold value=5

```
DECLARE
        out_of_stock   EXCEPTION;
        rec item%rowtype;
BEGIN
        select * into rec from item where item_no=1;
        IF rec. qty_in_stock < 5 THEN
                RAISE out_of_stock;
        END IF;
        DBMS_OUTPUT.PUT_LINE('qty_in_stock '|| rec. qty_in_stock );
EXCEPTION
        WHEN out_of_stock THEN
        insert into item_order values(1, rec.item_no, rec.item_name, 100);
        DBMS_OUTPUT.PUT_LINE('Order has been placed successfully');
END;
```

https://livesql.oracle.com/apex/f?p=590:1:10851832123804::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
BEGIN
    select * into rec from item where item_no=2;
            IF rec. qty_in_stock < 5 THEN
                    RAISE out_of_stock;
            END IF;
    DBMS_OUTPUT.PUT_LINE('qty_in_stock '|| rec. qty_in_stock );
EXCEPTION
    WHEN out_of_stock THEN
    insert into item_order values(1, rec.item_no, rec.name, 100);
    DBMS_OUTPUT.PUT_LINE('Order has been placed successfully');
END;
```

When exception part runs then body after exception doesn't run

```
Statement processed.
Order has been placed successfully
```

Error Trapping Functions

- In an exception handler, you can use the functions SQLCODE and SQLERRM to find out which error occurred and to get the associated error message.

Error Trapping Functions

Function	Description
SQLCODE	Returns the numeric value for the error code (You can assign it to a NUMBER variable)
SQLERRM	Returns character data containing the message associated with the error number

Possible Values of SQLCODE Function

SQLCODE	Description
0	No exception encountered
1	User-defined exception
+100	NO_DATA_FOUND exception
Negative number	Another Oracle Server error number

Error Trapping Functions

- The maximum length of an Oracle error message is 512 characters.
- SQLCODE or SQLERRM cannot be directly used in a SQL statement. Instead, their values must be assigned to local variables, which can be used in the SQL statement.

Example

```
DECLARE
        err_num NUMBER;
        err_msg CHAR(100);
BEGIN
  ……
EXCEPTION
  ……..
        WHEN OTHERS THEN
        err_num:=SQLCODE;
        err_msg:=SUBSTR(SQLERRM,1,100);
        INSERT INTO errors VALUES (err_num, err_msg);
END;
```

Live SQL

Feedback Help parteek.bhatia@gr

SQL Worksheet

Clear Find Actions Save

```
create table errors(err_num number(6), err_msg varchar(100));
drop table errors
select * from errors
DECLARE
N EMP.NAME%TYPE;
err_num NUMBER;
BEGIN
err_num:=to_number('a');
SELECT NAME INTO N FROM EMP ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN OTHERS THEN
    err_num:=SQLCODE;
    err_msg:=SUBSTR(SQLERRM,1,100);
    INSERT INTO errors VALUES (err_num, err_msg);
END;
```

```
Table dropped.
```

SQL Worksheet

Clear Find Actions

```
create table errors(err_num number(6), err_msg varchar(100));
drop table errors
select * from errors
DECLARE
N EMP.NAME%TYPE;
err_msg CHAR(100);
err_num NUMBER;
BEGIN
SELECT NAME INTO N FROM EMP  ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN OTHERS THEN
    err_num:=SQLCODE;
    err_msg:=SUBSTR(SQLERRM,1,100);
    INSERT INTO errors VALUES (err_num, err_msg);
    DBMS_OUTPUT.PUT_LINE('EXCEPTION HAS BEEN RAISED');
END;
```

ERR_NUM	ERR_MSG
100	ORA-01403: no data found

SQL Worksheet

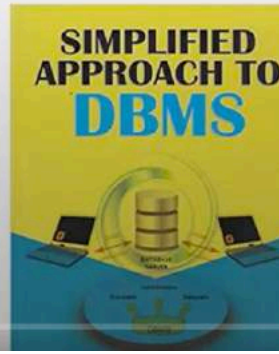
Clear Find Actions

```
create table errors(err_num number(6), err_msg varchar(100));
drop table errors
select * from errors
DECLARE
N EMP.NAME%TYPE;
err_msg CHAR(100);
err_num NUMBER;
BEGIN
SELECT NAME INTO N FROM EMP  ;
DBMS_OUTPUT.PUT_LINE('NAME IS '||N );
EXCEPTION
WHEN OTHERS THEN
    err_num:=SQLCODE;
    err_msg:=SUBSTR(SQLERRM,1,100);
    INSERT INTO errors VALUES (err_num, err_msg);
    DBMS_OUTPUT.PUT_LINE('EXCEPTION HAS BEEN RAISED');
END;
```

ERR_NUM	ERR_MSG
-1422	ORA-01422: exact fetch returns more than requested number of rows

Handing of Errors Raised in Declare Section and Propagation of Exceptions

By
Dr. Parteek Bhatia
Associate Professor
Department of Computer Science and Engineering
Thapar Institute of Engineering & Technology,
Patiala, India



Inbox (26,339) - parteek.bha | Oracle Live SQL - SQL V

https://livesql.oracle.com/apex/f?p=590:1:15923138244675::NO:::

Live SQL | Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
DECLARE
A NUMBER(3):=TO_NUMBER('AB');
BEGIN
DBMS_OUTPUT.PUT_LINE('NUMBER IS '||A );
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
END;
```

EXCEPTION HANDLER handles exceptions only after BEGIN and not of segment DECLARE

```
ORA-06502: PL/SQL: numeric or value error: character to number conversion error ORA-06512: at line 2
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Inbox (26,339) - parteek.bha | Oracle Live SQL - SQL W

https://livesql.oracle.com/apex/f?p=590:1:15923138244675::NO:::

Live SQL | Feedback | Help | parteek.bhatia@g

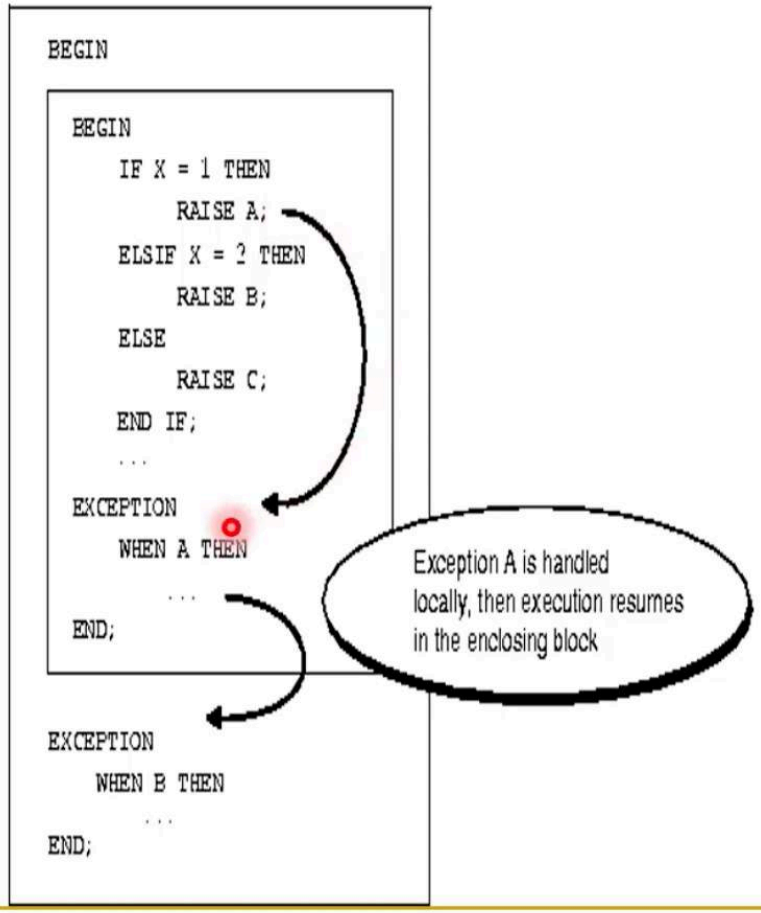
SQL Worksheet

Clear | Find | Actions | Save

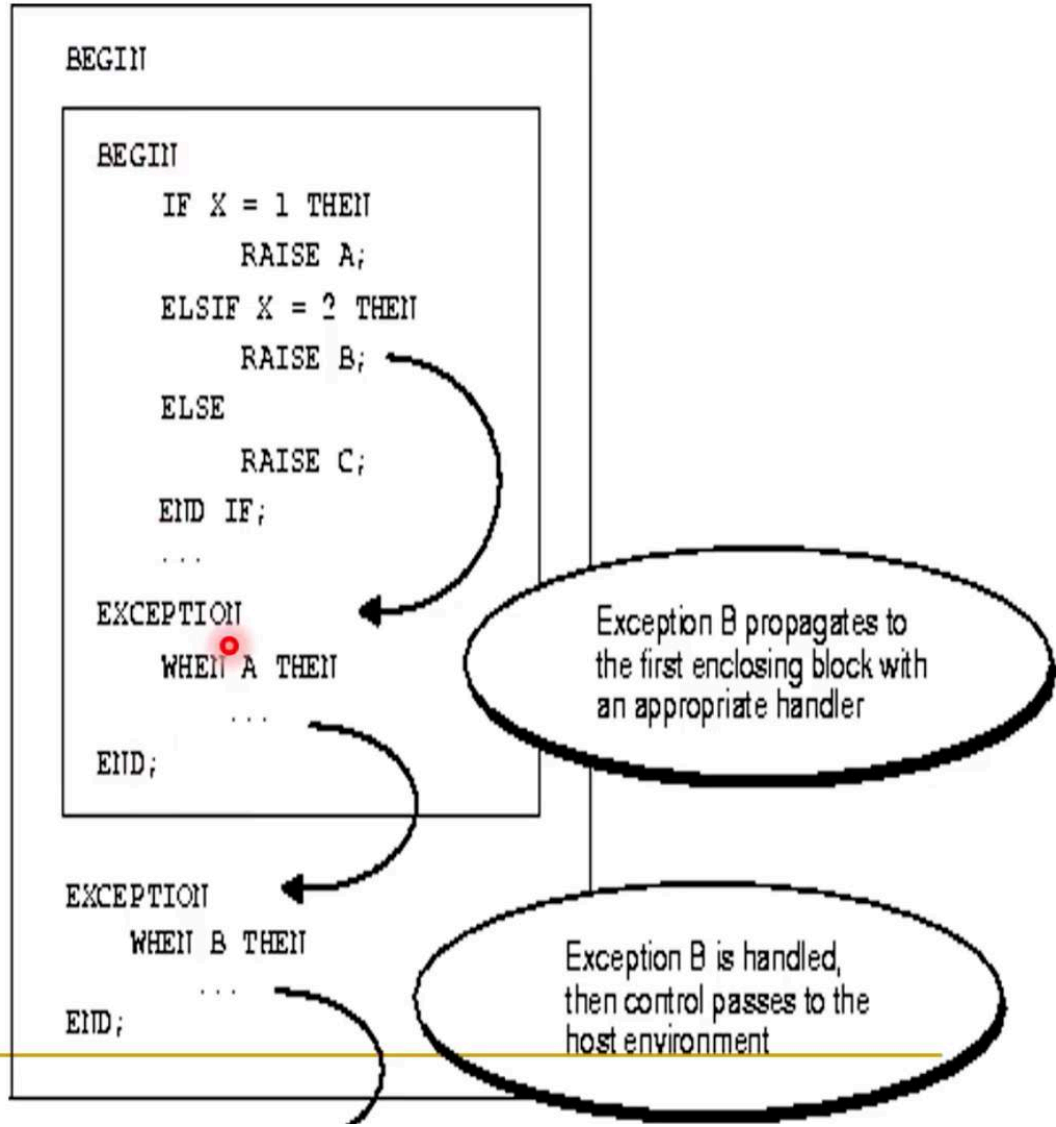
```
BEGIN
    DECLARE
    A NUMBER(3):=TO_NUMBER('AB');
    BEGIN
    DBMS_OUTPUT.PUT_LINE('NUMBER IS '||A );
    EXCEPTION
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('SOME RUN TIME ERROR OCCURS PLEASE CHECK');
    END;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('ERROR IS HANDLED');
```

```
Statement processed.
ERROR IS HANDLED
```

Propagating of Exceptions



Propagating of Exceptions



Propagating of Exceptions

```
BEGIN
  BEGIN
      IF X = 1 THEN
           RAISE A;
      ELSIF X = 2 THEN
           RAISE B;
      ELSE
           RAISE C;
      END IF;
      ...
  EXCEPTION
      WHEN A THEN
           ...
  END;

EXCEPTION
    WHEN B THEN
         ...
END;
```

Exception C has no handler, so an unhandled exception is returned to the host environment

Re-raising of an Exception

- Sometimes, we want to re-raise an exception and for this RAISE statement is use.

Example

```
DECLARE
        out_of_balance  EXCEPTION;
BEGIN
        ...
        BEGIN  ---------- sub-block begins
                IF ... THEN
                        RAISE out_of_balance;  -- raise the exception
                END IF;
        EXCEPTION
                WHEN out_of_balance THEN
                -- handle the error
         END;  ------------ sub-block ends
--These statements will be executed
EXCEPTION
        WHEN out_of_balance THEN
       -- handle the error differently
END;
```

Re-raising of an Exception

```
DECLARE
        out_of_balance  EXCEPTION;
BEGIN
         ...
         BEGIN  ---------- sub-block begins
                IF ... THEN
                      RAISE out_of_balance;  -- raise the exception
                END IF;
         EXCEPTION
                WHEN out_of_balance THEN
                -- handle the error
                RAISE;  -- reraise the current exception
          END;  ------------ sub-block ends
--Now These statements will not be executed
EXCEPTION
        WHEN out_of_balance THEN
       -- handle the error differently
END;
```

Assignment

Consider the following example:

```
EXCEPTION
        WHEN INVALID_NUMBER THEN
        INSERT INTO ...
        -- might raise DUP_VAL_ON_INDEX
        WHEN DUP_VAL_ON_INDEX THEN
        ******
END;
```

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
3  DECLARE
4  A NUMBER(3);
5  BEGIN
6  A:=TO_NUMBER('AB');
7  DBMS_OUTPUT.PUT_LINE('NUMBER IS '||A );
8  EXCEPTION
9  WHEN VALUE_ERROR THEN
10 insert into test values(1);
11 DBMS_OUTPUT.PUT_LINE('INSERTED SUCCESSFULLY');
12 WHEN DUP_VAL_ON_INDEX THEN
13 DBMS_OUTPUT.PUT_LINE('EXCEPTION HANDLED');
14 END;
```

```
ORA-00001: unique constraint (SQL_HIARNRZLHSCWSZWOANVCBJFKQ.SYS_C0027465170) violated ORA-06512: at line
8
```

Inbox (26,341) - parteek.bha | Oracle Live SQL - SQL W

https://livesql.oracle.com/apex/f?p=590:1:5718681419993::NO:::

Live SQL | Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet

Clear | Find | Actions | Save | Run

```
BEGIN
    DECLARE
    A NUMBER(3);
    BEGIN
    A:=TO_NUMBER('AB');
    DBMS_OUTPUT.PUT_LINE('NUMBER IS '||A );
    EXCEPTION
    WHEN VALUE_ERROR THEN
    insert into test values(1);
    DBMS_OUTPUT.PUT_LINE('INSERTED SUCCESSFULLY');
    WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('EXCEPTION HANDLED');
    END;
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
DBMS_OUTPUT.PUT_LINE('EXCEPTION HANDLED IN ENCLOSING BLOCK');
END;
```

```
Statement processed.
EXCEPTION HANDLED IN ENCLOSING BLOCK
```

Cursor: Definition

- Cursor is a memory (work) area that a oracle engine uses for its internal processing for executing and storing the results of SQL statement, and this work area is reserved for SQL's operations.
- Its is also called Oracle's Private area or CURSOR .
- For Example:

  When a user fires a select statement as:

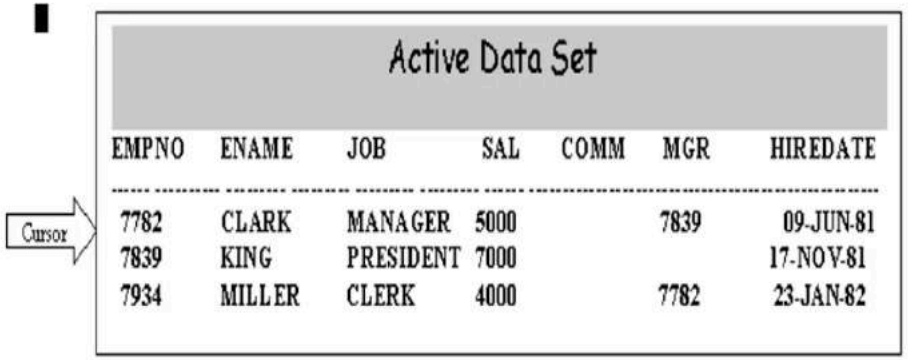
  SELECT empno, ename, job, sal FROM emp WHERE deptno = 10;

  All the rows returned by the query are stored in the cursor at the Server and will be as displayed at the client end.

Cursor: Concept

SQL > SELECT empno,ename,job,sal,comm,mgr,hiredate FROM emp WHERE deptno =10 ;

- When oracle engine executes the above query as a result some memory area is reserved and is fed with the result of the SQL statement.



EMPNO	ENAME	JOB	SAL	COMM	MGR	HIREDATE
7782	CLARK	MANAGER	5000		7839	09-JUN-81
7839	KING	PRESIDENT	7000			17-NOV-81
7934	MILLER	CLERK	4000		7782	23-JAN-82

Types of Cursor

- Implicit Cursors
- Explicit Cursors

Implicit Cursor

- Implicit Cursors are declared by PL/SQL implicitly for all SQL statements. They are opened and managed by Oracle engine internally.
- So, there is no need to open and manage by the programmer, these are operations are performed automatically.
- The name of Implicit Cursor is “SQL”

Implicit Cursor

- Implicit cursors attributes can be used to access information about the status of last insert, update, delete or single-row select statements.
- This can be done by preceding the implicit cursor attribute with the cursor name (i.e. SQL).

General Cursor Attributes

Attributes	Description
%ISOPEN	It returns TRUE if cursor is open, FALSE otherwise.
%FOUND	It returns TRUE if record was fetched successfully from the opened cursor, and FALSE otherwise.
%NOTFOUND	It returns TRUE if record was not fetched successfully and FALSE otherwise.
%ROWCOUNT	It returns number of records processed from cursor.

Implicit Cursor Attributes

Automatically
Opened Processed Closed

Attributes	Description
SQL%ISOPEN	It is always FALSE because Oracle automatically closes an Implicit cursor after executing its SQL statement.
SQL%FOUND	It is TRUE if the most recently executed DML statement was successful.
SQL%NOTFOUND	It is TRUE if the most recently executed DML statement was not successful.
SQL%ROWCOUNT	It returns the number of rows affected by an INSERT, UPDATE, DELETE, or single row SELECT statement.

SQL Statement

After Processing SQL Statement Implict cursor is always in closed state

https://livesql.oracle.com/apex/f?p=590:1:17287862005695::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
create table emp(eno number(3) primary key, name char(30), job char(20), sal number(5),deptno number(2));
insert into emp values(1,'ram','prof',90000,10);
insert into emp values(2,'sham','assoc prof',70000,20);
insert into emp values(3,'raj','prof',95000,10);
select * from emp;
```

```
1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

Work - parteek.bhatia@thai | Oracle Live SQL - SQL W

https://livesql.oracle.com/apex/f?p=590:1:17287862005695::NO:::

Live SQL

SQL Worksheet

```
1  update emp set sal=30000 ;
```

```
3 row(s) updated.
```

Work - parteek.bhatia@thai | Oracle Live SQL - SQL W

https://livesql.oracle.com/apex/f?p=590:1:17287862005695::NO:::

Live SQL

SQL Worksheet

```
1  update emp set sal=30000 where deptno=30;
```

```
0 row(s) updated.
```

Surplus Librar | Get ready to | Temporary Er | Oracle Liv | (9) Anupam S | Online Learni | (19) CSR (Cor | Channel mon

https://livesql.oracle.com/apex/f?p=590:1:5977194329771::LEVEL1:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
begin
update emp set sal=30000;
if sql%isopen then
    dbms_output.put_line('Implicit Cursor is in open state');
else
    dbms_output.put_line('Implicit Cursor is in close state');
end if;
if sql%found then
    dbms_output.put_line('Updation has been done. '||sql%rowcount||' rows have been updated');
else
    dbms_output.put_line('Updation has not been done');
end if;
end;
```

```
Statement processed.
Implicit Cursor is in close state
Updation has been done. 3 rows have been updated
```

Use of SQL%NOTFOUND Attribute

Consider a PL/SQL code to display a message to check whether the record is deleted or not.

```
BEGIN
DELETE EMP WHERE ENO=1;
IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('RECORD NOT DELETED');
ELSE
        DBMS_OUTPUT.PUT_LINE('RECORD DELETED');
END IF;
END
```

Explicit Cursor Handling

Steps:

- Declare the cursor

CURSOR <cursor-name> IS <select statement>;

- Open the cursor

OPEN <cursor-name>;

- Fetch data from the cursor one row at a time into memory variables.

FETCH <cursor-name> INTO <variables>;

Process the data of all the rows of cursors by storing its values in the memory variables one row at a time by using a loop.

Exit from the loop after processing is complete.

- Close the cursor

Close <cursor-name>;

Need of Explicit Cursor

Write a PL/SQL block to display name of all employees of EMP table.

```
DECLARE
N EMP.ENAME%TYPE;
BEGIN
SELECT ENAME INTO N FROM EMP;
END;
```

NOT RIGHT IF WE WANT TO DISPLAY MULTIPLE ROWS

Consider a PL/SQL code to display the empno, ename, job of employees of department number 10.

```
DECLARE
        CURSOR C1 IS SELECT EMPNO, ENAME, JOB FROM EMP WHERE DEPTNO=10;
        REC C1%ROWTYPE;  --rec is a row type variable for cursor c1 record, containing empno, ename and job
BEGIN
        OPEN C1;
        LOOP
                FETCH C1 INTO REC;
                EXIT WHEN C1%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO);
                DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
                DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
        END LOOP;
        CLOSE C1;
END;
```

Inbox (26,370) - parteek.bha | Oracle Live SQL - SQL W | dsfdsf - Bing

https://livesql.oracle.com/apex/f?p=590:1:11061360547405::NO:::

Live SQL | Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet | Clear | Find | Actions | Save | Run

```
DECLARE
    CURSOR C1 IS SELECT EMPNO, ENAME, JOB FROM EMP WHERE DEPTNO=10;
    REC C1%ROWTYPE;  --rec is a row type variable for cursor c1 record, containing empno, e
BEGIN
    OPEN C1;
    LOOP
        FETCH C1 INTO REC;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO);
                DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
        DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
```

3	raj	prof	95000	10

Download CSV

Cursor FOR Loop: Syntax

```
FOR <variable name> IN <cursor_name> LOOP
    <statements>;
END LOOP;
```

- No need to declare this <variable name> as it automatically declares it of cursor_name%rowtype.
- No need to open the cursor.
- No need to use Fetch statement to fetch a row from the cursor.
- No need to close the cursor when all rows have been processed

A PL/SQL code to display the empno, ename, job of employees of department number 10 with CURSOR FOR Loop statement.

```
DECLARE
        CURSOR C1 IS SELECT * FROM EMP WHERE DEPTNO=10;
BEGIN
FOR REC IN C1 LOOP
DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO ||'ENAME ' ||REC.ENAME|| 'JOB '||REC.JOB);
END LOOP;
END;
```

(33) Parteek Bhatia: Simplify | Video sessions of Cursor Ha | Oracle Live SQL - SQL V

https://livesql.oracle.com/apex/f?p=590:1:9252845581443::NO:::

Live SQL

Feedback | Help | parteek.bhatia@gmail.c

- Home
- SQL Worksheet
- My Session
- Schema
- Quick SQL
- My Scripts
- My Tutorials
- Code Library

SQL Worksheet

Clear | Find | Actions | Save | Ru

```
DECLARE
    CURSOR C1 IS SELECT * FROM EMP WHERE DEPTNO=10;
BEGIN
FOR REC IN C1 LOOP
DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO ||'ENAME ' ||REC.ENAME|| 'JOB '||REC.JOB);
END LOOP;
END;
```

```
Statement processed.
EMPNO 1ENAME Raj                    JOB Prof
EMPNO 3ENAME Rahat                  JOB Prof
```

ORACLE | Integrated Cloud Applications & Platform Services

© 2020 Oracle Corporation · Privacy · Terms of Use

Oracle Learning Library · Ask Tom · Dev Gym · Database Doc 19c , 18c , 12c · Follow on Twitter

Live SQL 20.1.1, running Oracle Database 19c Enterprise Edition - 19.5.0.0.0

Built with ♥ using Oracle APEX

A PL/SQL code to display the empno, ename, job of employees of department number 10 with CURSOR FOR Loop statement.

With Cursor For Loop

```
DECLARE
        CURSOR C1 IS SELECT * FROM
        EMP WHERE DEPTNO=10;
BEGIN
   FOR REC IN C1 LOOP
        DBMS_OUTPUT.PUT_LINE('EMPNO
        '||REC.EMPNO ||'ENAME '
        ||REC.ENAME|| 'JOB '||REC.JOB);
   END LOOP;
END;
```

Without Cursor For Loop

```
DECLARE
        CURSOR C1 IS SELECT * FROM   EMP
         WHERE DEPTNO=10;
        REC C1%ROWTYPE;
BEGIN
 OPEN C1;
        LOOP
        FETCH C1 INTO REC;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('EMPNO
        '||REC.EMPNO ||'ENAME '
        ||REC.ENAME|| 'JOB '||REC.JOB);
        END LOOP;
 CLOSE C1;
END;
```

Cursors with Parameters

Syntax:

To declare

```
CURSOR cursor_name (variable_name datatype) IS
<SELECT statement…>
```

To Open

```
OPEN cursor_name (value/variable/expression);
```

Consider a PL/SQL code to display the empno, ename, job of employees of a particular department number whose value is passed as a parameter.

```
DECLARE
        CURSOR C1 (d number)  IS
        SELECT * FROM EMP WHERE DEPTNO=d;
        REC C1%ROWTYPE;
BEGIN
         OPEN C1(10); -- OPEN C1(&d);
        LOOP
                FETCH C1 INTO REC;
                EXIT WHEN C1%NOTFOUND;
                DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.ENO);
                DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
                DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
        END LOOP;
        CLOSE C1;
END;
```

https://livesql.oracle.com/apex/f?p=590:1:9969241912836::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
DECLARE
    CURSOR C1 (d number)  IS
    SELECT * FROM EMP WHERE DEPTNO=d;
    REC C1%ROWTYPE;
BEGIN
     OPEN C1(10); -- OPEN C1(&d);
    LOOP
        FETCH C1 INTO REC;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO);
                DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
        DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
    END LOOP;
    CLOSE C1;
```

```
EMPNO 1
ENAME Raj
JOB Prof
EMPNO 3
ENAME Rahat
```

Consider a PL/SQL code to display the empno, ename, job of employees of a particular department number whose value is passed as a parameter.

```
DECLARE
        CURSOR C1(d number) IS
        SELECT * FROM EMP WHERE DEPTNO=d;
BEGIN
FOR REC IN C1(20) LOOP --FOR REC IN C1(&d) LOOP
        DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.ENO);
        DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
        DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
END LOOP;
END;
```

https://livesql.oracle.com/apex/f?p=590:1:9969241912836::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail

SQL Worksheet

Clear Find Actions Save

```
DECLARE
    CURSOR C1(d number) IS
    SELECT * FROM EMP WHERE DEPTNO=d;
BEGIN
FOR REC IN C1(30) LOOP --FOR REC IN C1(&d) LOOP
    DBMS_OUTPUT.PUT_LINE('EMPNO '||REC.EMPNO);
    DBMS_OUTPUT.PUT_LINE('ENAME '||REC.ENAME);
    DBMS_OUTPUT.PUT_LINE('JOB '||REC.JOB);
END LOOP;
END;
```

```
ENAME Raj
JOB Prof
EMPNO 3
ENAME Rahat
JOB Prof
```

Cursor Assignment-2

- Let us consider CUST table with columns
- Cno PK, MeterNo Unique, Prev_Reading, Current_Reading, Units, Bill_Amount
- Units=Current_Reading-Prev_Reading
- Bill will be charged according to following rate:
- First 100 Units 0.5 Rs per unit and after 0.75 Rs for every unit consumed.

Cno	MeterNo	Prev_Reading	Current_Reading	Units	Bill_Amount
89	100	300	800		
67	101	790	1000		
90	200	800	1200		
62	789	200	800		
70	899	3200	8700		
66	500	3000	9800		

Cursor Assignment-2

- Write a PL/SQL code to update units and bill amount of customer number 89 in CUST table
- Write a PL/SQL code to update units and bill amount of all the customers in CUST table

Cno	MeterNo	Prev_Reading	Current_Reading	Units	Bill_Amount
89	100	300	800		
67	101	790	1000		
90	200	800	1200		
62	789	200	800		
70	899	3200	8700		
66	500	3000	9800		

Write a PL/SQL code to update units and bill amount of all the customers in CUST table

Cno	MeterNo	Prev_Reading	Current_Reading	Units	Bill_Amount
89	100	300	800		
67	101	790	1000		
90	200	800	1200		
62	789	200	800		
70	899	3200	8700		
66	500	3000	9800		

```
DECLARE
        CURSOR C1 IS SELECT * FROM CUST;
        B NUMBER;
        U NUMBER;
        EXTRA NUMBER;
BEGIN
        FOR REC IN C1 LOOP
                U:=REC.CURRENT_READING-REC.PREV_READING;
                IF U<100 THEN
                        B:=U*0.5;
                ELSE
                        EXTRA:=U-100;
                        B:=100*0.5+EXTRA*0.75;
                END IF;
        UPDATE CUST SET UNITS=U, BILL_AMOUNT=B WHERE CNO=REC.CNO;
        END LOOP;
END;
```

Google AdSense | Inbox (26,458) - parteek.bha | Oracle Live SQL - SQL V

https://livesql.oracle.com/apex/f?p=590:1:6249020552455::NO:::

Live SQL

Feedback | Help | parteek.bhatia@gmail.com

SQL Worksheet

Clear | Find | Actions | Save | Run

```
Create table Cust(
Cno number(4) primary key,
MeterNo number(4) unique,
Prev_Reading number(5),
Current_Reading number(5),
Units number(5),
Bill_Amount number(6));
insert into cust values(89, 100, 300, 800, null, null);
insert into cust values(67,101,790, 1000, null,null);
insert into cust values(90,200, 800,1200, null,null);
insert into cust values(62,789,200,800, null,null);
insert into cust values(70,899,3200,8700, null,null);
insert into cust values(66,500,3000,9800, null,null);
select * from cust;
```

SQL Statement Output

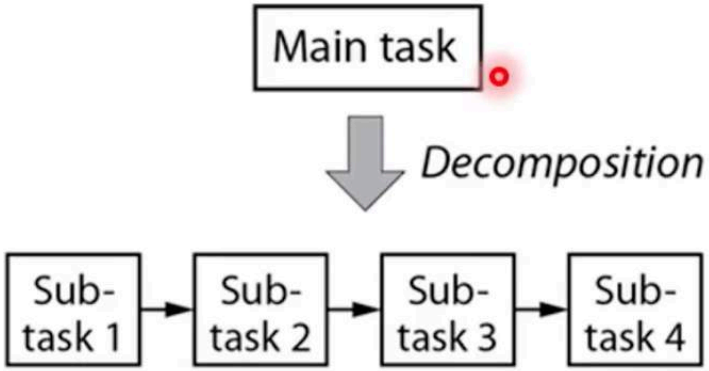
Write a PL/SQL code to update the bill of customer number 89 in CUST table

```
DECLARE
        REC CUST%ROWTYPE;
        B NUMBER;
        U NUMBER;
        EXTRA NUMBER;
BEGIN
        SELECT * INTO REC FROM CUST WHERE CNO=89;
        U:=REC.CURRENT_READING-REC.PREV_READING;
                IF U<100 THEN
                        B:=U*0.5;
                ELSE
                        EXTRA:=U-100;
                        B:=100*0.5+EXTRA*0.75;
                END IF;
        UPDATE CUST SET UNITS=U, BILL_AMOUNT=B WHERE CNO=89;
END;
```

PL/SQL 2

Subprograms

- Subprograms allow decomposition of a program into logical units.



- It is used to develop User defined Functions and Procedures in Oracle.
- Subprograms are named PL/SQL blocks that can take parameters and can be invoked.

Subprograms

PL/SQL has two types of subprograms:

- Functions
- Procedures

They can be further classified as:

- Local subprograms
- Stored Subprograms

Local and Stored Subprogram

Local Subprogram

- Local Subprogram is local to the PL/SQL module, which contain it.
- It can be created in the declarative section of PL/SQL module.
- It can only be invoked in PL/SQL block in which it is created.

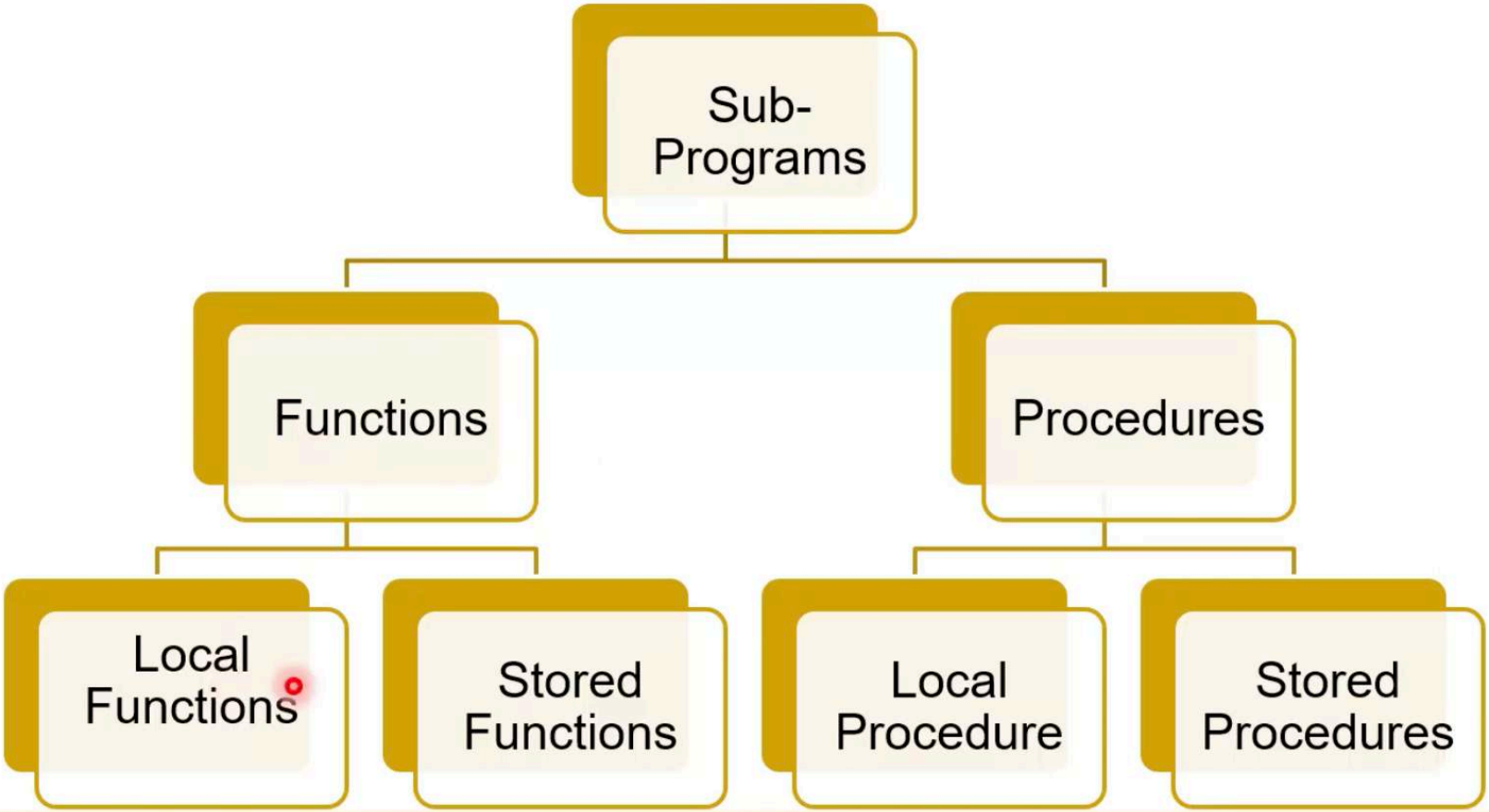
Local and Stored Subprogram

Stored Subprogram

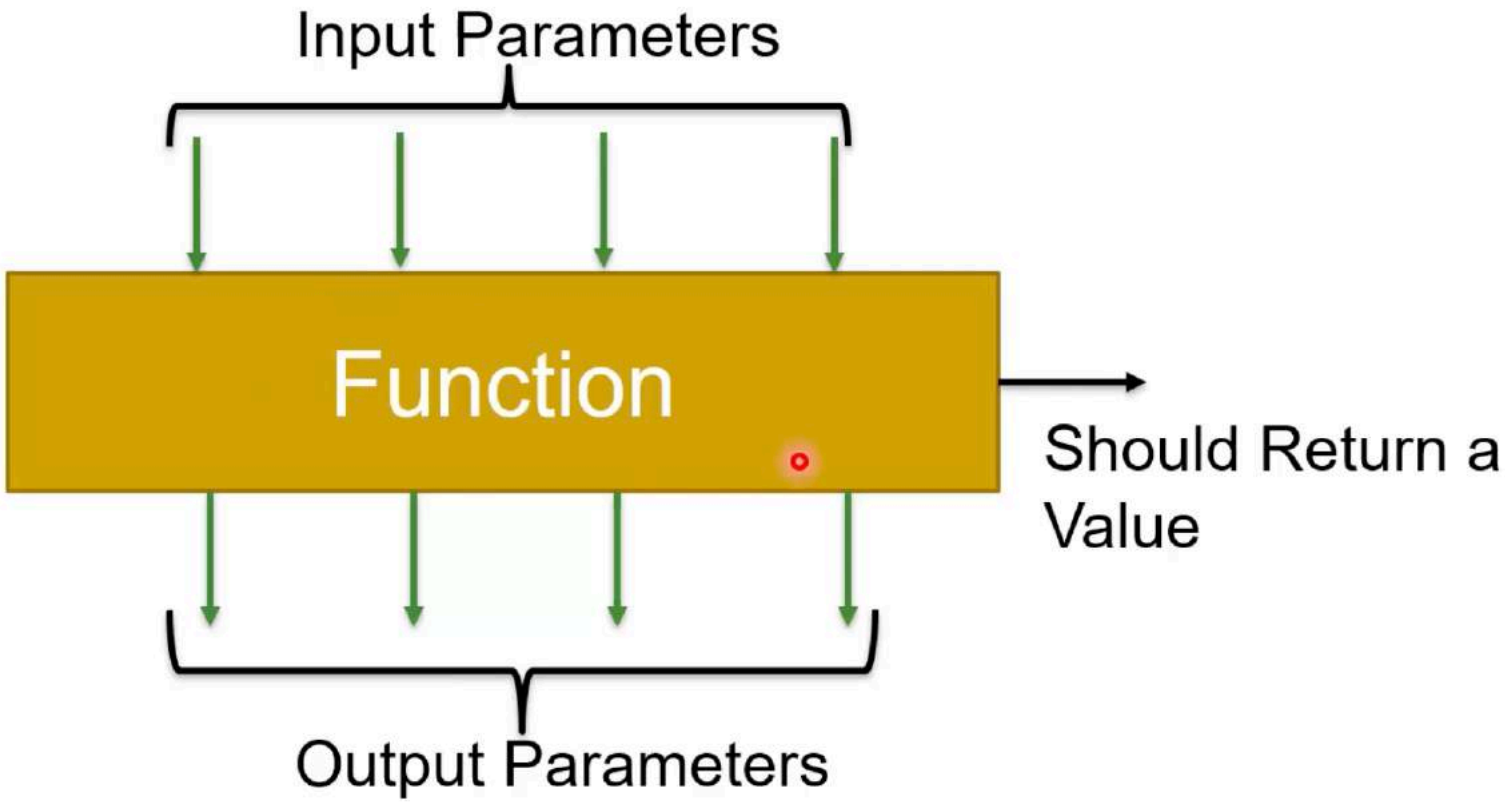
Stored subprogram is a named PL/SQL block that has been compiled and stored in Oracle database.

- It is stored as database object in database as tables are stored.
- They are invoked or called by any the PL/SQL block.

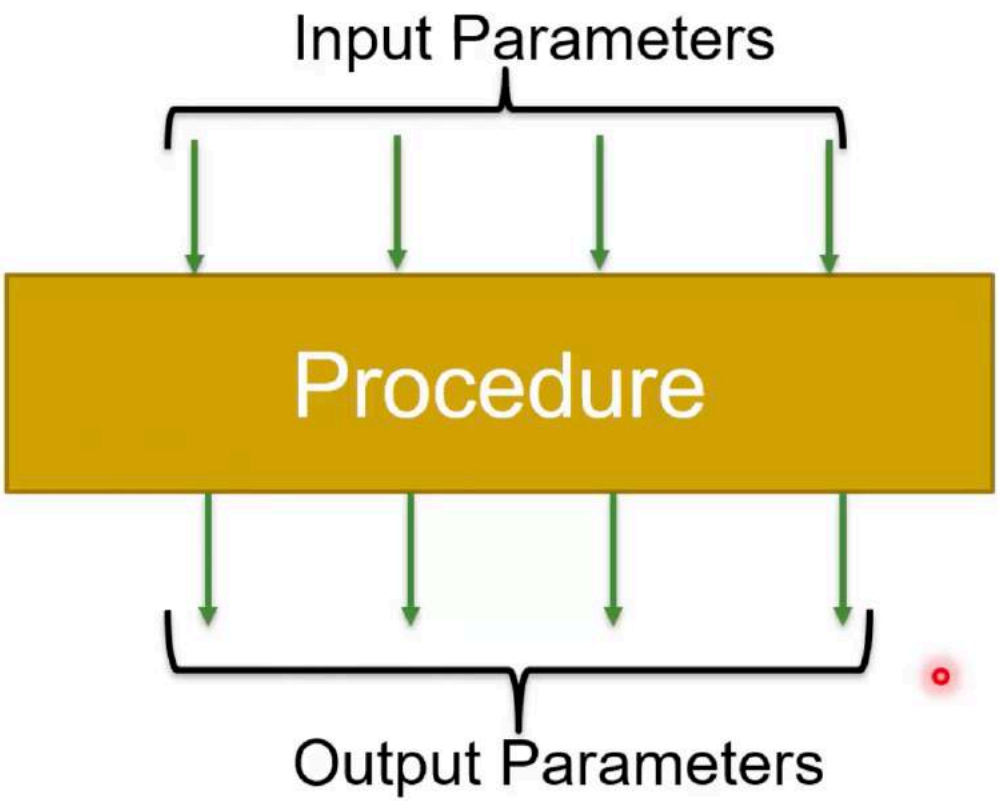
Types of Subprograms



Function


Input Parameters
Function
Should Return a Value
Output Parameters

Procedure



Difference between Procedure and Function

Procedure	Function
A Procedure is a subprogram that can accept parameters, perform an action and return parameters.	A Function is a subprogram that can accept parameters, compute a value and return that value to caller.
A procedure may return no value.	A Function must return only one value.
A procedure cannot return a value as direct output.	A Function can return a value as direct output of the function call.
Generally, we use a procedure to perform an action.	Generally, we use a function to compute a value.

Local and Stored Subprogram

Local Subprogram

- Local Subprogram is local to the PL/SQL module, which contain it.
- It can be created in the declarative section of PL/SQL module.
- It can only be invoked in PL/SQL block in which it is created.

Local Function: Syntax

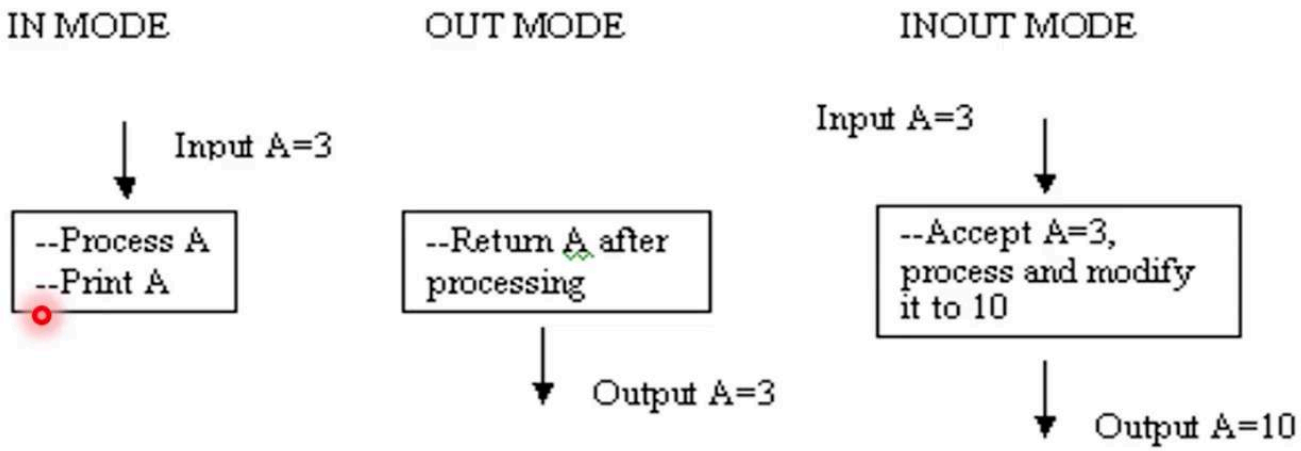
```
DECLARE
        FUNCTION function_name
        [(argument {IN|OUT|INOUT} data type, ...)] RETURN datatype {IS | AS}
        [local declarations]
        BEGIN
        PL/SQL subprogram body;
        [EXCEPTION
        exception handlers]
        END [function_name ];
BEGIN
--PL/SQL code to call function
[EXCEPTION
--calling program exception handler]
END;
```

Local Function: Syntax

There is a space in IN OUT

```
DECLARE
        FUNCTION function_name
        [(argument {IN|OUT|INOUT} data type, ...)] RETURN datatype {IS | AS}
        [local declarations]
        BEGIN
        PL/SQL subprogram body;
        [EXCEPTION
        exception handlers]
        END [function_name ];
BEGIN
--PL/SQL code to call function
[EXCEPTION
--calling program exception handler]
END;
```

Types of Arguments



Types of Arguments

IN	OUT	IN OUT
It is the default.	It must be specified.	It must be specified.
It passes values to a subprogram.	It returns values to the caller.	It passes initial values to a subprogram and returns updated values to the caller.
It is a formal parameter acts like a constant.	It is a formal parameter acts like an uninitialized variable.	It is a formal parameter acts like an initialized variable.
It is a formal parameter cannot be assigned a value.	It is a formal parameter cannot be used in an expression and must be assigned a value.	It is a formal parameter should be assigned a value.
It can be a constant, initialized variable, literal, or expression.	It must be a variable.	It must be a variable.

A PL/SQL code to write local function to add two numbers.

```
DECLARE
        A NUMBER;
        B NUMBER;
        RES NUMBER;
        FUNCTION ADDN ( FIRST IN NUMBER, SECOND IN NUMBER) RETURN NUMBER IS
        BEGIN
        RES:=FIRST+SECOND;
        RETURN(RES);
        END;
BEGIN
        A:=3;    --A:=&FIRSTNUMBER;
        B:=2;    --B:=&SECONDNUMBER;
        RES:=ADDN(A,B);
        DBMS_OUTPUT.PUT_LINE('ADDITION IS '||RES);
END;
```

Write a PL/SQL code for local function to return balance of the customer on the basis of account number which is passed to this local function as its parameter.
CUST(acc_no, name, balance)

```
DECLARE
acc_no NUMBER;
bal NUMBER;
        FUNCTION balance (acct_id NUMBER) RETURN NUMBER IS
        acct_bal CUST.BALANCE%TYPE;
        BEGIN
        SELECT BALANCE INTO acct_bal FROM CUST WHERE acc_no = acct_id;
        RETURN acct_bal;
        END balance;
BEGIN
         acc_no:=100;      --acc_no:=&acc_no;
        bal:=balance(acc_no);
        DBMS_OUTPUT.PUT_LINE('BALANCE IS '|| bal);
END;
```

https://livesql.oracle.com/apex/f?p=590:1:7275001288351::NO:::

Live SQL

Feedback Help parteek.bhatia

orksheet

ssion

a

SQL

ipts

torials

Library

SQL Worksheet

Clear Find Actions Save

```
    exception
    when no_data_found then
    return -100;
    DBMS_OUTPUT.PUT_LINE('no record for account number');
    END balance;
BEGIN
    acc_no:=105;    --acc_no:=&acc_no;
    bal:=balance(acc_no);
    if bal=-100 then
    DBMS_OUTPUT.PUT_LINE('no record for account number '||acc_no);
    else
    DBMS_OUTPUT.PUT_LINE('BALANCE IS '|| bal);
    end if;
END;
```

```
Statement processed.
no record for account number 105
```

© 2020 Oracle Corporation · Privacy · Terms of Use
Oracle Learning Library · Ask Tom · Dev Gym · Database Doc 19c , 18c , 12c · Follow on Twitter
ORACLE | Integrated Cloud

A PL/SQL code to write local function to return total number of employees in EMP table.

```
DECLARE
        FUNCTION TOTAL_EMP RETURN NUMBER IS
        TOTAL NUMBER;
        BEGIN
        SELECT COUNT(*) INTO TOTAL FROM EMP;
        RETURN(TOTAL);
        END;
BEGIN
DBMS_OUTPUT.PUT_LINE('TOTAL NUMBER OF EMPLYEES '||TOTAL_EMP());
END;
```

Local and Stored Subprogram

Stored Subprogram

Stored subprogram is a named PL/SQL block that has been compiled and stored in Oracle database.

- It is stored as database object in database as tables are stored.
- They are invoked or called by any the PL/SQL block.

Stored Function: Syntax

```
CREATE OR REPLACE FUNCTION function_name
        [(argument {IN|OUT|INOUT} DATATYPE,…)]
        RETURN datatype {IS | AS}
        [local declarations]
BEGIN
        PL/SQL subprogram body;
[EXCEPTION
        exception PL/SQL block;]
END;
```

Compilation of Stored Functions and Stored Procedures

- Stored Functions and Stored Procedures are always created after compilation, if there is no error then subprogram is created successfully otherwise invalid subprogram is created with errors. But of course subprogram will be created.
- So, to view the compilation errors of subprogram we can use following select statement:
- SELECT * FROM USER_ERRORS;
- Or
- Show errors;

Dropping of a Stored Function

- To drop a function, DROP FUNCTION command is used.

Syntax:

- DROP FUNCTION function_name;

Example:

- DROP FUNCTION ADDN;

A PL/SQL code to create a stored function to add two numbers.

```
CREATE OR REPLACE FUNCTION ADDN( A IN NUMBER, B IN NUMBER) RETURN NUMBER IS
C NUMBER;
BEGIN
        C:=A+B;
        RETURN(C);
END;
```

Code of PL/SQL block to call stored function ADDN( )

```
DECLARE
  A NUMBER;
  B NUMBER;
  C NUMBER;
BEGIN
  A:=3;        --A:=&FIRSTNUMBER;
  B:=2;        --B:=&SECONDNUMBER;
  C:=ADDN(A,B);
  DBMS_OUTPUT.PUT_LINE('ADDITION IS'||C);
END;
```

A PL/SQL code to create a stored function to find total salary paid to employees of a particular department number. The department number is passed as parameter to the stored function. EMP (eno, ename, job, salary, deptno)

```
CREATE OR REPLACE FUNCTION SALARY (dno NUMBER)
RETURN NUMBER IS
total_salary NUMBER;
BEGIN
Select sum(salary) into total_salary from emp where deptno=dno;
RETURN(total_salary);
END;
```

Code of PL/SQL block to call stored function SALARY ( )

```
DECLARE
        D NUMBER;
        TOTAL_SALARY NUMBER;
BEGIN
        D:=10;        --D:=&DEPTNO;
        TOTAL_SALARY :=SALARY(D);
        DBMS_OUTPUT.PUT_LINE('salary of department   number' ||
        d || 'is' || TOTAL_SALARY);
END;
```

Local and Stored Subprogram

Local Subprogram

- Local Subprogram is local to the PL/SQL module, which contain it.
- It can be created in the declarative section of PL/SQL module.
- It can only be invoked in PL/SQL block in which it is created.

Local Procedure: Syntax

```
DECLARE
        --Declaration of Global variables
        PROCEDURE name
         [(argument {IN, OUT, IN OUT} data type, ....)]  {IS | AS}
          [local declarations]
        BEGIN
            PL/SQL subprogram body;
        [EXCEPTION
          exception handlers]
        END [name];
BEGIN
            --Executable code
            --Code to call procedure
  [EXCEPTION
          exception handlers for main block]
END;
```

- Example 1: Write a PL/SQL code of a local procedure that accepts two numbers and return its addition, subtraction, multiplication and division or in other words write a procedure to return multiple values through parameters.

```
DECLARE
        FIRST_NUMBER NUMBER;
        SECOND_NUMBER NUMBER;
        ADD NUMBER;
        SUB NUMBER;
        MUL NUMBER;
        DIV NUMBER;
        PROCEDURE PROCESS( A IN NUMBER, B IN NUMBER, C OUT NUMBER, D OUT NUMBER,
  E OUT NUMBER, F OUT NUMBER) IS
        BEGIN
        C:=A+B;
        D:=A-B;
        E:=A*B;
        F:=A/B;
        END;
BEGIN
FIRST_NUMBER:=4;
SECOND_NUMBER:=2;
PROCESS(FIRST_NUMBER,SECOND_NUMBER,ADD,SUB,MUL,DIV);
DBMS_OUTPUT.PUT_LINE('ADDITION IS '||ADD || ' SUBTRACTION IS '||SUB ||' MUTIPLICATION IS '||MUL||' DIVISION IS '||DIV);
END;
```

- Example 2: Write a PL/SQL code of a local procedure raise_salary, which accepts two parameters empid and bonus. It increases the salary of the employee by passed value of bonus and update it to the database.

Example 2: Write a PL/SQL code of a local procedure raise_salary, which accepts two parameters empid and the amount to be added to salary of employee. It increases the salary of the employee and update it to the database.

```
DECLARE
        EMP_NO NUMBER;
        BONUS NUMBER;
        PROCEDURE RAISE_SALARY( EMPID IN NUMBER,  BONUS IN NUMBER) IS
        BEGIN
        UPDATE EMP SET SALARY=SALARY+BONUS WHERE ENO=EMPID;
        END;
BEGIN
EMP_NO:=1;-- EMP_NO:=&EMP_NO;
BONUS:=2000; -- BONUS:=&BONUS;
RAISE_SALARY(EMP_NO,BONUS);
END;
```

Local and Stored Subprogram

Stored Subprogram

Stored subprogram is a named PL/SQL block that has been compiled and stored in Oracle database.

- It is stored as database object in database as tables are stored.
- They are invoked or called by any the PL/SQL block.

SIMPLIFIE APPROACH

Stored Procedure

```
CREATE OR REPLACE PROCEDURE procedure_name
[(argument {IN| OUT | IN OUT} datatype,….)] {IS | AS}
[local declarations];
BEGIN
        PL/SQL subprogram body;
[EXCEPTION
        Exception PL/SQL block;]
END;
```

Dropping a Stored Procedure

- To drop a procedure, DROP PROCEDURE command is used.

Syntax:

- DROP PROCEDURE procedure_name;

Example:

- DROP PROCEDURE PROCESS;

- Example 1: A Stored procedure that accepts two numbers and return addition, subtraction, multiplication and division of two numbers or in other words a stored procedure to return multiple values through arguments.

Code of Stored procedure process that accepts two numbers and return addition, subtraction, multiplication and division of two numbers or in other words a stored procedure to return multiple values through arguments.

```
CREATE OR REPLACE PROCEDURE PROCESS( A IN NUMBER, B IN
NUMBER, C OUT NUMBER, D OUT NUMBER, E OUT NUMBER, F OUT
NUMBER) IS
BEGIN
        C:=A+B;
        D:=A-B;
        E:=A*B;
        F:=A/B;
END;
```

Code of PL/SQL block to call stored procedure process.

```
DECLARE
        FIRST_NUMBER NUMBER;
        SECOND_NUMBER NUMBER;
        ADD NUMBER;
        SUB NUMBER;
        MUL NUMBER;
        DIV NUMBER;
BEGIN
FIRST_NUMBER:=4;
SECOND_NUMBER:=2;
PROCESS(FIRST_NUMBER,SECOND_NUMBER,ADD,SUB,MUL,DIV);
DBMS_OUTPUT.PUT_LINE('ADDITION IS '||ADD || ' SUBTRACTION IS '||SUB ||' MUTIPLICATION IS '||MUL||' DIVISION IS '||DIV);
END;
```

Example 2: Code to create a stored procedure fire_employee to delete employee on the basis of passed employee number.

```
CREATE OR REPLACE PROCEDURE fire_employee (emp_id
NUMBER) AS
BEGIN
  DELETE FROM emp WHERE empno = emp_id;
END;
```

Code of PL/SQL block to call fire_employee procedure.

```
DECLARE
        e number;
BEGIN
        e:=100;     --e:=&empno;
        fire_employee(e);
END;
```

e Live SQL - SQL V

https://livesql.oracle.com/apex/f?p=590:1:9057298392591::NO:::

Live SQL

Feedback Help parteek.bhatia@gmail.com

sheet

n

s

als

ary

SQL Worksheet

Clear Find Actions Save Run

```
CREATE TABLE EMP(ENO NUMBER(3), ENAME CHAR(20), JOB CHAR(10), SALARY NUMBER(9), DEPTNO NUMBER(3));
INSERT INTO EMP VALUES(100,'RAM', 'PROF',90000,10);
INSERT INTO EMP VALUES(101,'RAJ', 'ASSOC PROF',80000,20);
INSERT INTO EMP VALUES(102,'SHAM', 'PROF',95000,20);
SELECT * FROM EMP;
CREATE OR REPLACE PROCEDURE fire_employee (emp_id NUMBER) AS
BEGIN
   DELETE FROM emp WHERE empno = emp_id;
END;
SELECT * FROM USER_ERRORS;
```

NAME	TYPE	SEQUENCE	LINE	POSITION	TEXT	ATTRIBUTE	MESSAGE_NUMBER
FIRE_EMPLOYEE	PROCEDURE	1	3	26	PL/SQL: ORA-00904: "EMPNO": invalid identifier	ERROR	0
FIRE_EMPLOYEE	PROCEDURE	2	3	4	PL/SQL: SQL Statement ignored	ERROR	0

© 2020 Oracle Corporation · Privacy · Terms of Use

Integrated Cloud

Oracle Learning Library · Ask Tom · Dev Gym · Database Doc 19c , 18c , 12c · Follow on Twitter

Sub Program Assignment: Question 1

- Create a stored sub program to insert a record in emp table. We have to just pass the values of columns to this stored sub program and it should insert a record with passed parameter values.

Stored sub procedure to insert a record in emp table.

```
CREATE OR REPLACE PROCEDURE INSERT_EMP
( ENO IN NUMBER, ENAME CHAR, JOB CHAR, SALARY NUMBER, DEPTNO NUMBER) IS
BEGIN
INSERT INTO EMP VALUES (ENO, ENAME, JOB, SALARY, DEPTNO);
END;
```

CODE OF PL/SQL BLOCK TO CALL THIS PROCEDURE

```
BEGIN
INSERT_EMP(1,'RAM','PROF',200000,10);
DBMS_OUTPUT.PUT_LINE('1 ROW HAS BEEN INSERTED');
END;
```

Sub Program Assignment: Question 2

Create a stored sub program to delete employees on the basis of passed department number. It should return the value to provide the information of number records deleted to the calling environment.

Sub Program Assignment: Question 2

```
CREATE OR REPLACE FUNCTION fire_employee (dep_id IN
NUMBER) RETURN NUMBER AS
BEGIN
  DELETE FROM emp WHERE DEPTNO = dep_id;
  RETURN(SQL%ROWCOUNT);
END;
```

Sub Program Assignment: Question 2

```
DECLARE
        N NUMBER;
BEGIN
        N:= fire_employee(10);-- N:= fire_employee(&d);
        DBMS_OUTPUT.PUT_LINE('number of records deleted '||N);
END;
```

Sub Program Assignment

- Let us consider CUST table with columns
- Cno PK, MeterNo Unique, Prev_Reading, Current_Reading, Units, Bill_Amount
- Units=Current_Reading-Prev_Reading
- Bill will be charged according to following rate:
- First 100 Units 0.5 Rs per unit and after 0.75 Rs for every unit consumed.

Cno	MeterNo	Prev_Reading	Current_Reading	Units	Bill_Amount
89	100	300	800		
67	101	790	1000		
90	200	800	1200		
62	789	200	800		
70	899	3200	8700		
66	500	3000	9800		

Sub Program Assignment

Q 1: Write a stored sub program to update units and bill amount of a customer whose customer number is passed as parameter to this stored sub program.

Q 2: Write a stored sub program to update units and bill amount of all the customers in CUST table.

Cno	MeterNo	Prev_Reading	Current_Reading	Units	Bill_Amount
89	100	300	800		
67	101	790	1000		
90	200	800	1200		
62	789	200	800		
70	899	3200	8700		
66	500	3000	9800		

Stored Procedure to update the units and bill of customer in CUST table whose Customer Number is passed as parameter to this procedure.

```
CREATE OR REPLACE PROCEDURE BILL(CUST_NO IN NUMBER) IS
        REC CUST%ROWTYPE;
        B NUMBER;
        U NUMBER;
        EXTRA NUMBER;
BEGIN
        SELECT * INTO REC FROM CUST WHERE CNO=CUST_NO;
        U:=REC.CURRENT_READING-REC.PREV_READING;
                IF U<100 THEN
                        B:=U*0.5;
                ELSE
                        EXTRA:=U-100;
                        B:=100*0.5+EXTRA*0.75;
                END IF;
        UPDATE CUST SET UNITS=U, BILL_AMOUNT=B WHERE CNO=CUST_NO;
END;
```

Stored Procedure to update units and bill amount of all the customers in CUST table

```
CREATE OR REPLACE PROCEDURE BILL_ALL IS
        CURSOR C1 IS SELECT * FROM CUST;
        B NUMBER;
        U NUMBER;
        EXTRA NUMBER;
BEGIN
        FOR REC IN C1 LOOP
                U:=REC.CURRENT_READING-REC.PREV_READING;
                IF U<100 THEN
                        B:=U*0.5;
                ELSE
                        EXTRA:=U-100;
                        B:=100*0.5+EXTRA*0.75;
                END IF;
        UPDATE CUST SET UNITS=U, BILL_AMOUNT=B WHERE CNO=REC.CNO;
        END LOOP;
END;
```

```
--calling of this Stored Procedure
BEGIN
        BILL_ALL;
END;
```

Live SQL

Feedback Help parteek.bhat

SQL Worksheet

Clear Find Actions Sa

```
            EXTRA:=U-100;
            B:=100*0.5+EXTRA*0.75;
        END IF;
    UPDATE CUST SET UNITS=U, BILL_AMOUNT=B WHERE CNO=REC.CNO;
    END LOOP;
END;

--calling of this Stored Procedure
BEGIN
    BILL_ALL;
END;

select * from cust;
```

CNO	METERNO	PREV_READING	CURRENT_READING	UNITS	BILL_AMOUNT
89	100	300	800	500	350

Concept of Trigger

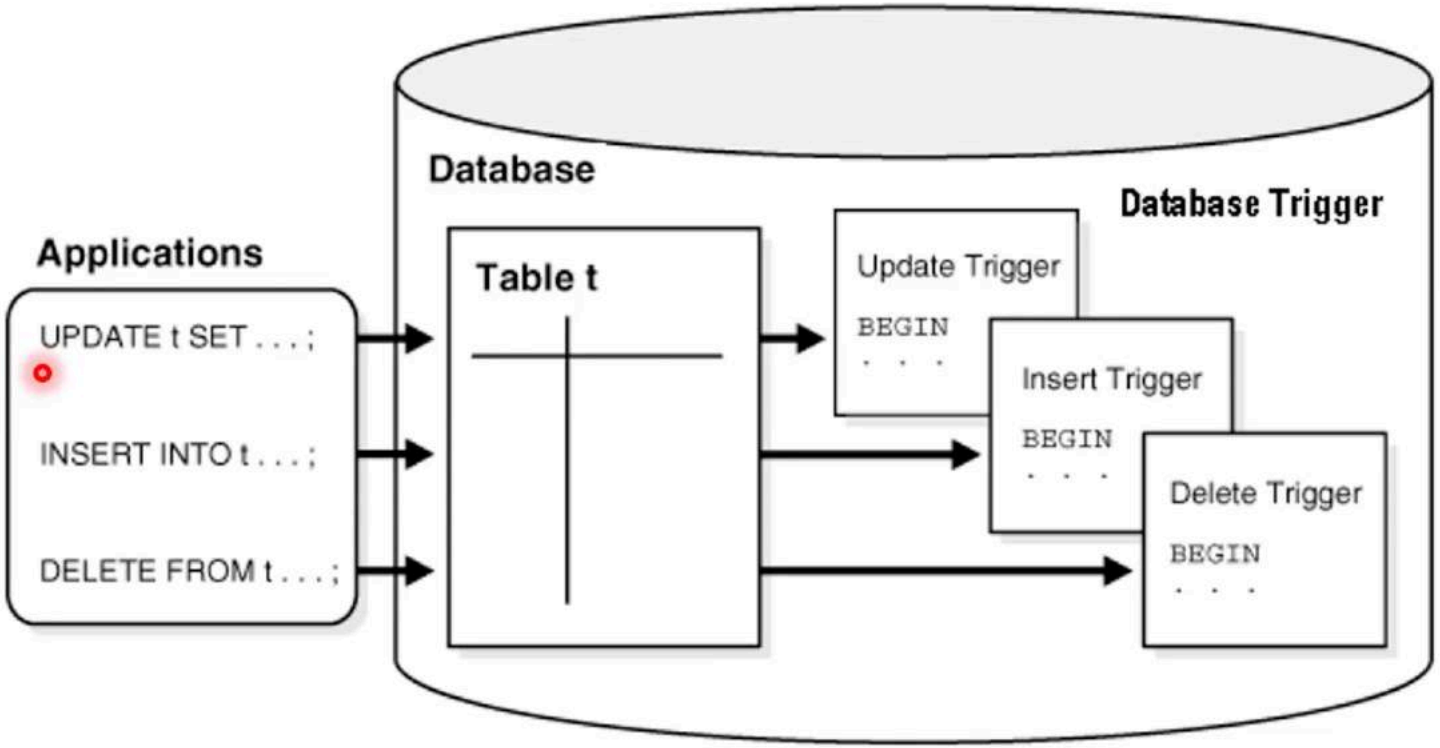
- A trigger is a code that is automatically executed or fired when its corresponding database event (Insert, Update and Delete) occurs.

- A trigger defines an action the database should take when some database related event occurs.

Introduction to Triggers

- A database trigger is similar to a stored procedure that is fired when an INSERT, UPDATE, or DELETE statements is issued against the associate table.
- A trigger is automatically executed without any action required by the user. A stored procedure on other hand needs to be explicitly invoked.

Working of Trigger



Database Trigger V/s Procedure

Database Triggers	Procedures
Triggers do not accept parameters	Procedures can accept parameters
A trigger is automatically executed without any action required by the user.	A stored procedure on other hand needs to be explicitly invoked.

Uses of Database Triggers

- To derive column values automatically.
- To enforce complex integrity constraints.
- To enforce complex business rules.
- To customize complex security authorizations.
- To maintain replicated tables.
- To audit data modifications.

Examples of Triggers: Primary Key Constraint

- Primary key constraint acts like a trigger which is automatically created by Oracle and it internally fires a code to check that input value of primary key should not already present in the database and it should be not null.
- INSERT INTO EMP VALUES(1,'RAM','PROF',100000,10);

Examples of Triggers: Primary Key Constraint

- INSERT INTO EMP VALUES(1,'RAM','PROF',100000,10);

```
PL/SQL CODE
---
SELECT * INTO REC FROM EMP
WHERE ENO=1;
```

If Yes
Stop Insert Execution
Else
Allow Insert Execution

It can be considered as Trigger

ENO	ENAME	JOB	SAL	DEPTNO
2	--	--	--	--
3	--	--	--	--
1	--	--	--	--
5	--	--	--	--
7	--	--	--	--

Examples of Triggers: Foreign Key Constraint

Employee table

Eno	Ename	Job	Deptno
1	Ram	Prof	10
2	Rajesh	Asst Prof	20
3	Suresh	Asst Prof	10
4	Ramesh	Prof	20
5	Surinder	Asst Prof	30

FK

P K

Department table

Deptno	Dname
10	Computer
20	Chemical
30	Mechanical

If Yes
Allow Insert Execution
Else
Stop Insert Execution

Query the database for presence for Deptno 10 in Master

SELECT * INTO REC FROM DEPT WHERE DEPTNO=10;

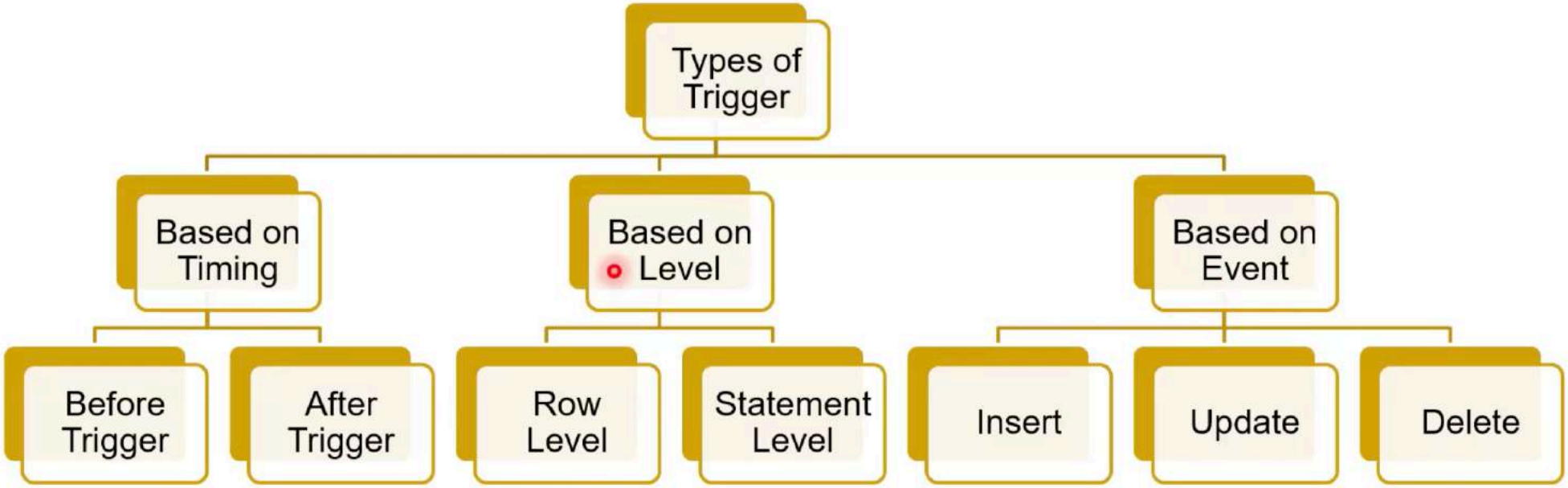
INSERT INTO EMP VALUES(6,'RAM','PROF',100000,10);

Some Applications of Triggers

- To enforce that no update should perform on Sunday.
- To enforce that record of job title CEO should not be removed from the database.
- To enforce that when you update EMP table its corresponding backup table should also be updated automatically.
- To enforce that user only insert values of Cno, Meter_no, Prev_Reading and Current_Reading while Units and Bill_Amount be will be automatically calculated and inserted along with these values for CUST table.

  CUST(Cno, Meter_no, Prev_Reading, Current_Reading, Units, Bill_Amount)

Types of Trigger

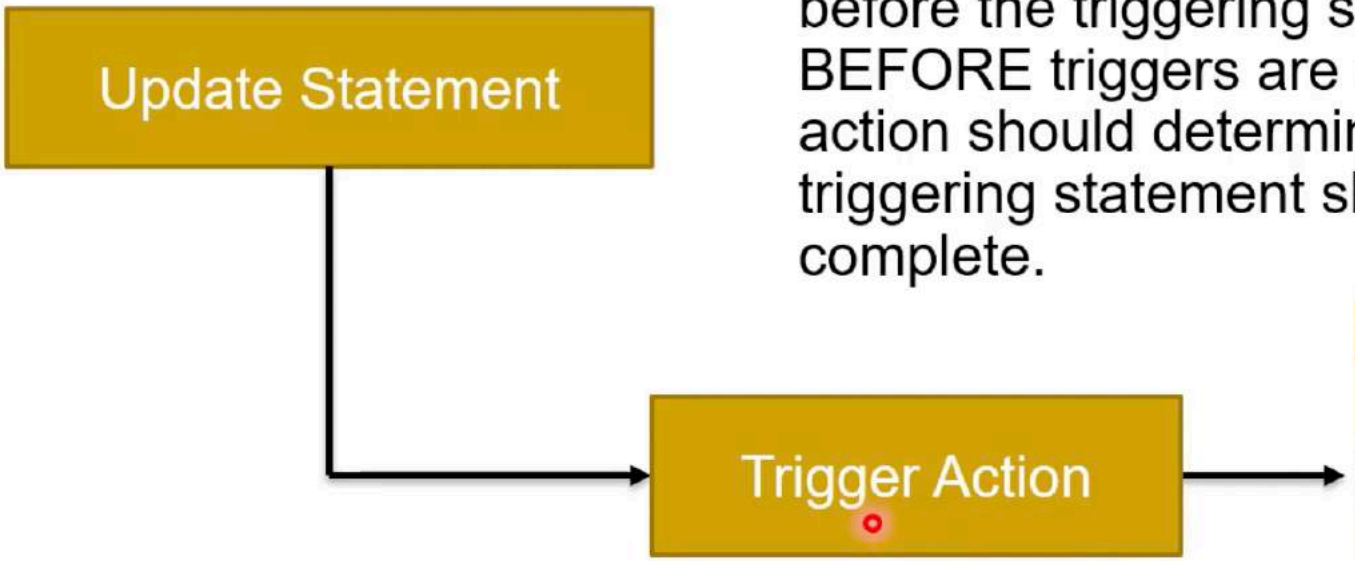


Trigger types

There are 12 types.

- BEFORE INSERT row level
- BEFORE INSERT statement level
- AFTER INSERT row level
- AFTER INSERT statement level
- BEFORE UPDATE row level
- BEFORE UPDATE statement level
- AFTER UPDATE Row level
- AFTER UPDATE statement level
- BEFORE DELETE row level
- BEFORE DELETE statement level
- AFTER DELETE row level
- AFTER DELETE statement level

Before Trigger

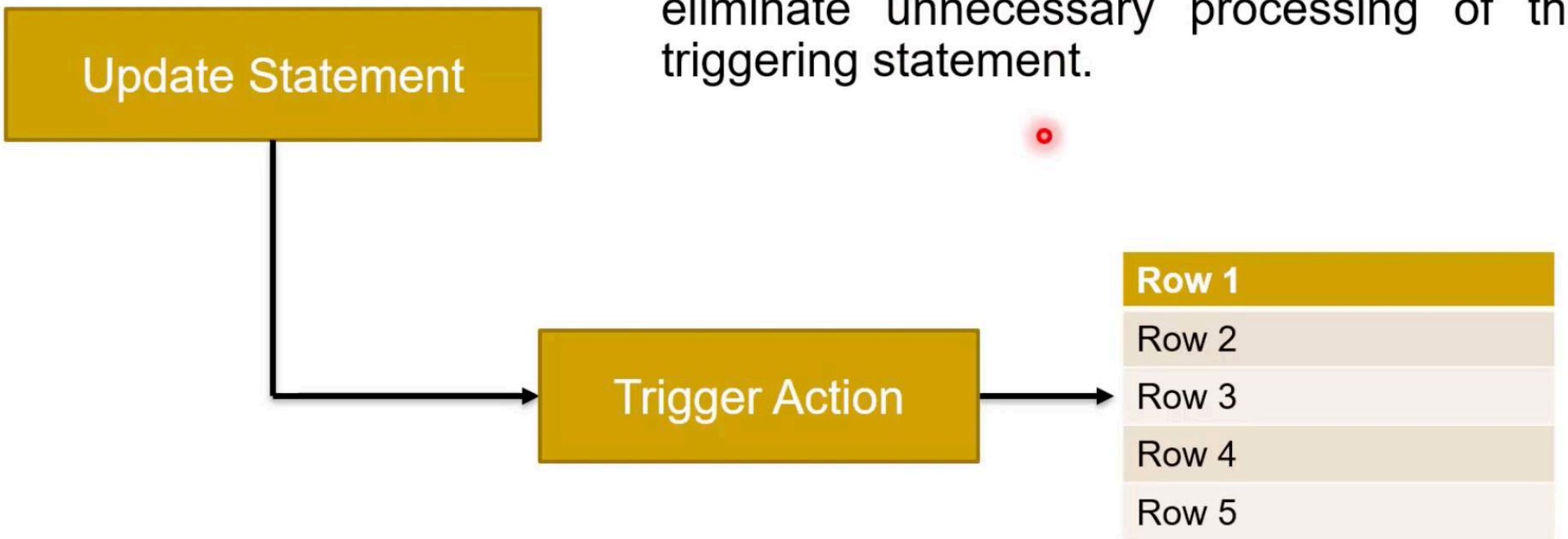


BEFORE trigger execute the trigger action before the triggering statement.
BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete.

Row 1
Row 2
Row 3
Row 4
Row 5

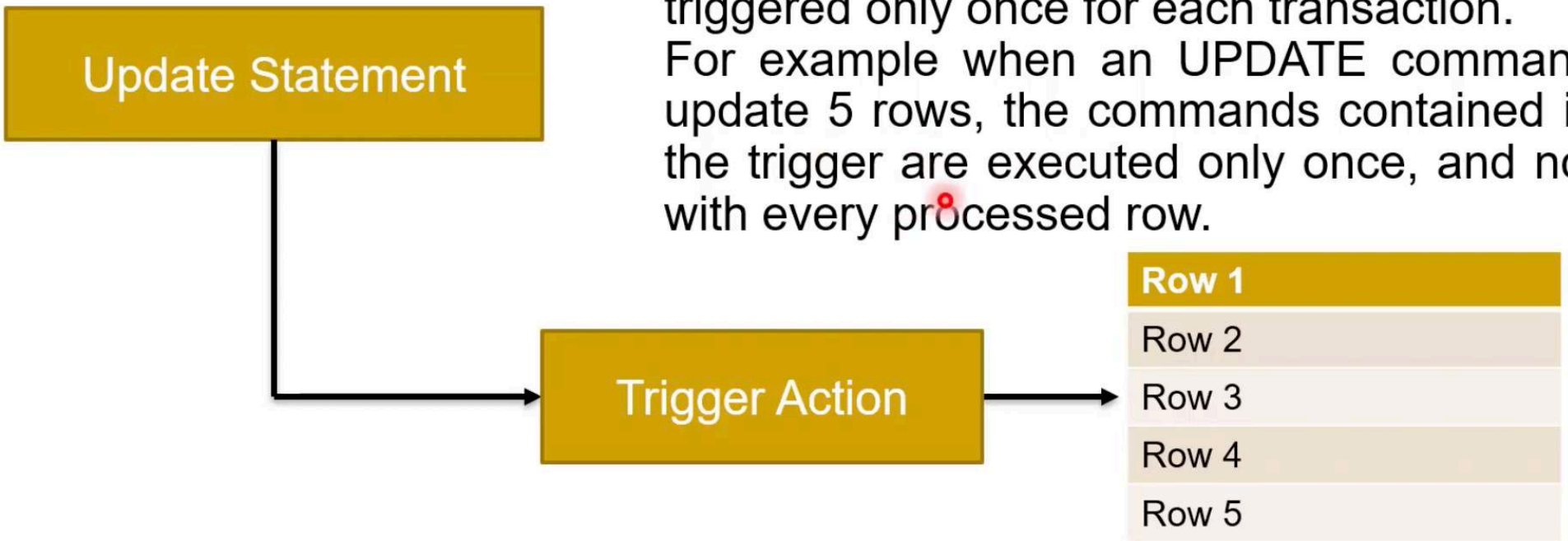
Before Trigger

By using a BEFORE trigger, you can eliminate unnecessary processing of the triggering statement.



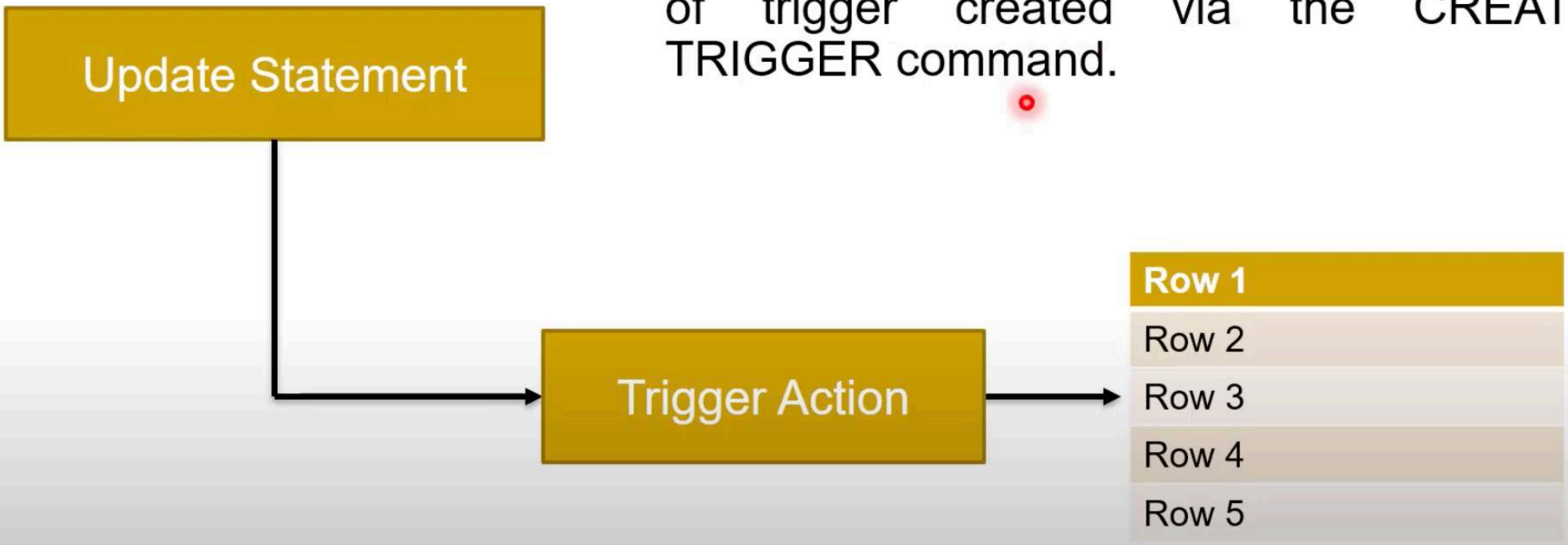
Before Statement Level Trigger

Before Statement level triggers are triggered only once for each transaction. For example when an UPDATE command update 5 rows, the commands contained in the trigger are executed only once, and not with every processed row.

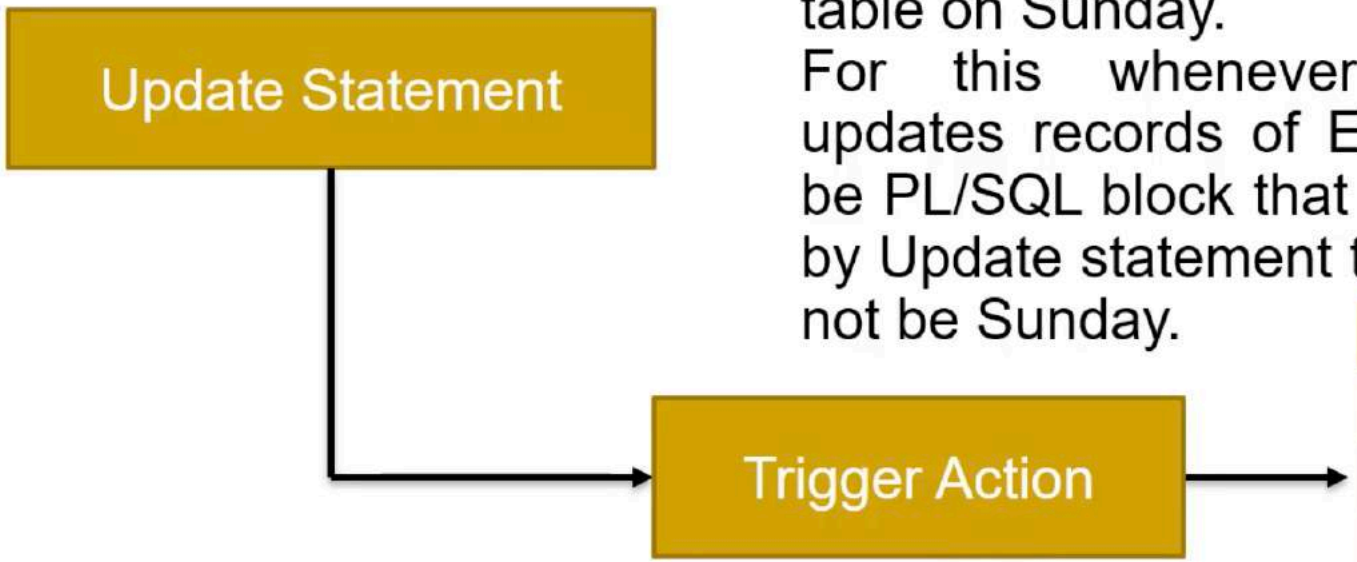


Before Statement Level Trigger

Statement level trigger are the default types of trigger created via the CREATE TRIGGER command.



Application: Before Statement Level Trigger

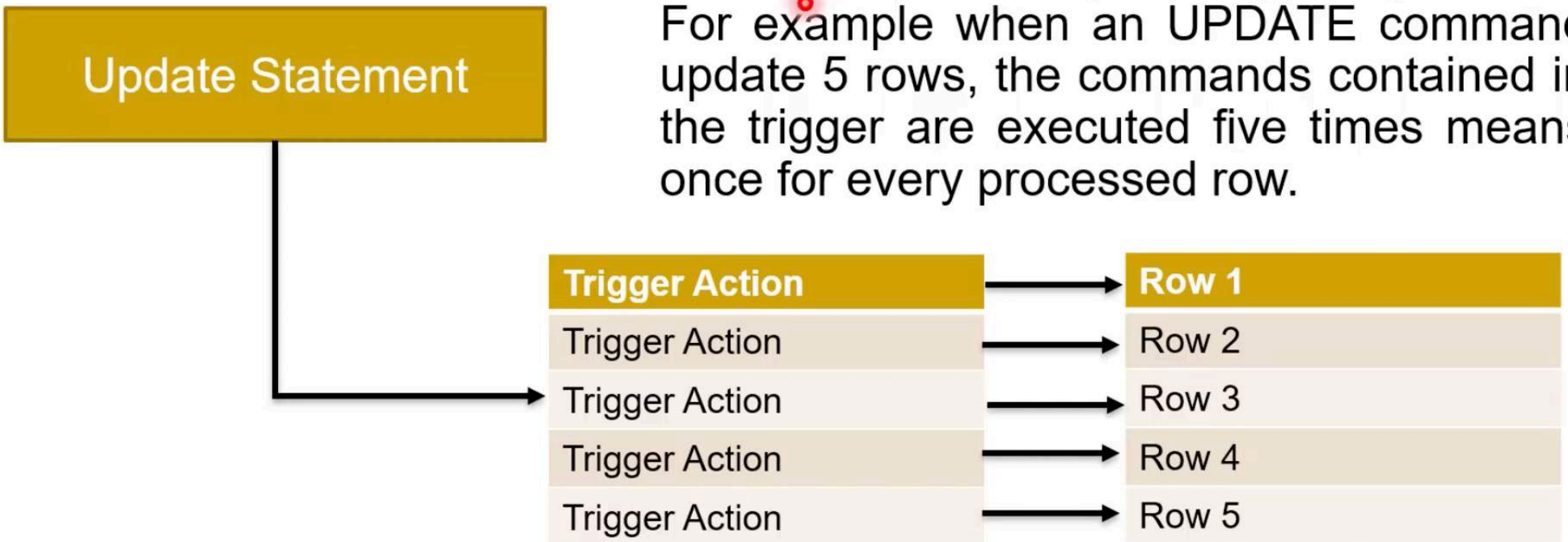


To prevent the Update operation on EMP table on Sunday.
For this whenever Update statement updates records of EMP table, there must be PL/SQL block that will be fired only once by Update statement to check that day must not be Sunday.

Row 1
Row 2
Row 3
Row 4
Row 5

Before Row Level Trigger

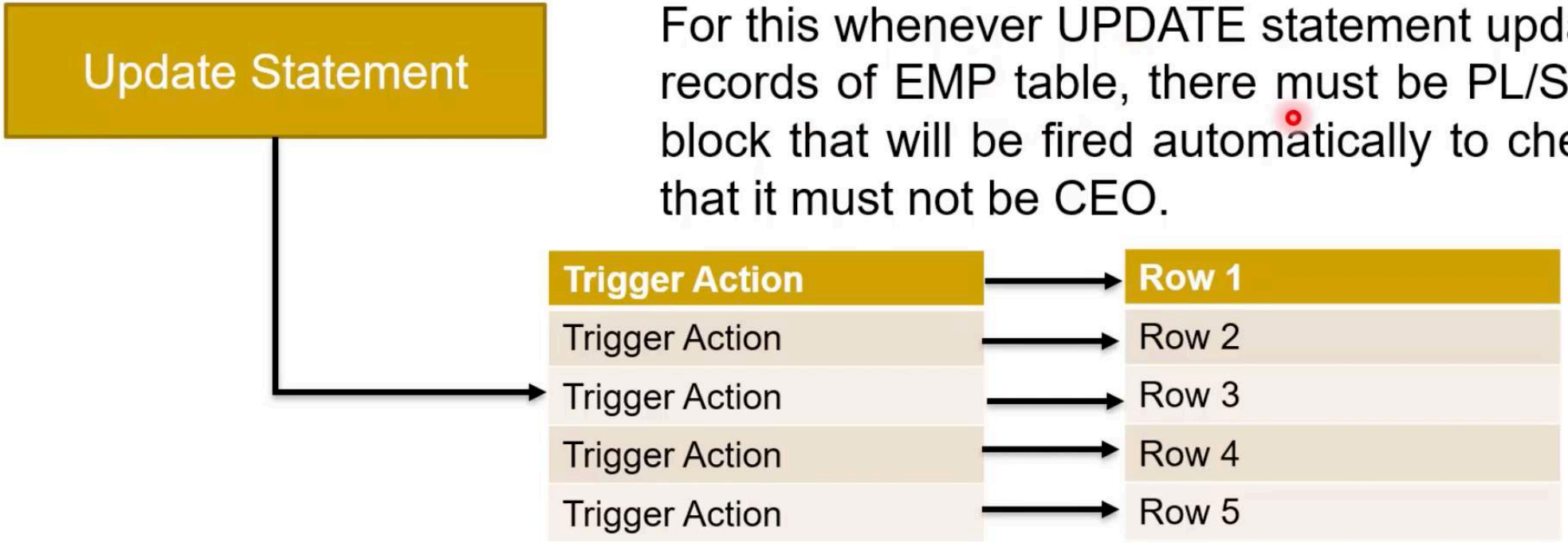
Before row level triggers are triggered for each row modified by the triggering event. For example when an UPDATE command update 5 rows, the commands contained in the trigger are executed five times means once for every processed row.



Application: Before Row Level Trigger

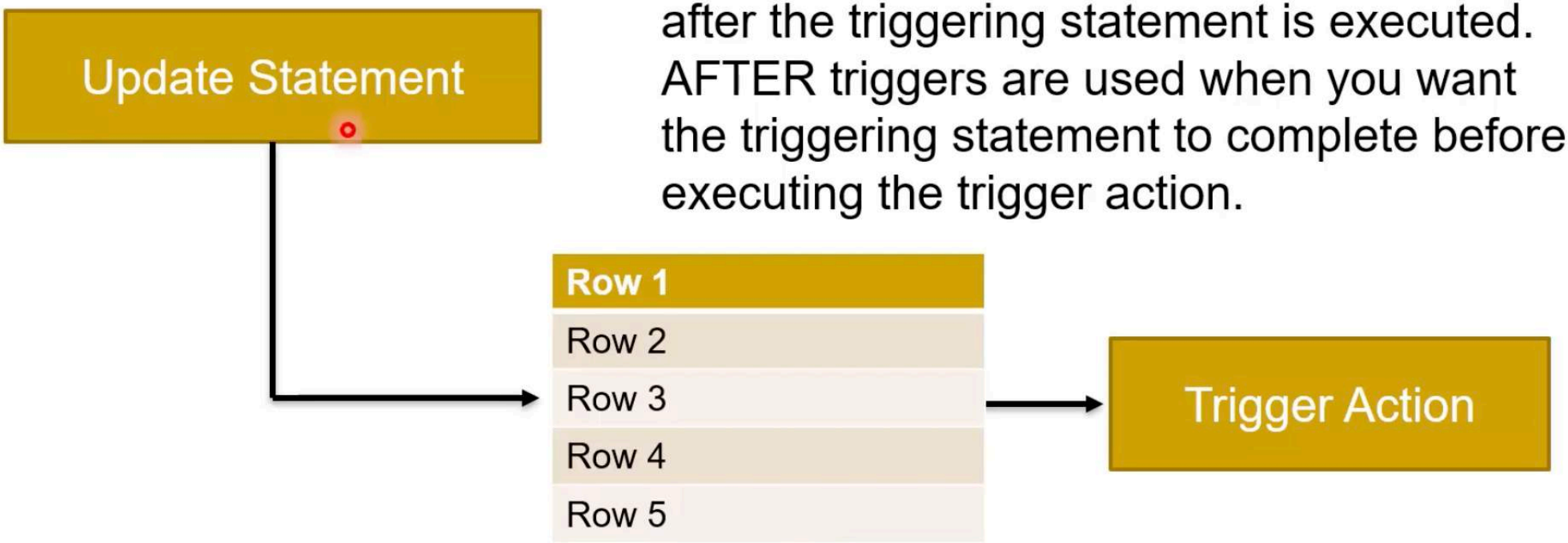
To prevent updation of job title CEO records of EMP table.

For this whenever UPDATE statement update records of EMP table, there must be PL/SQL block that will be fired automatically to check that it must not be CEO.



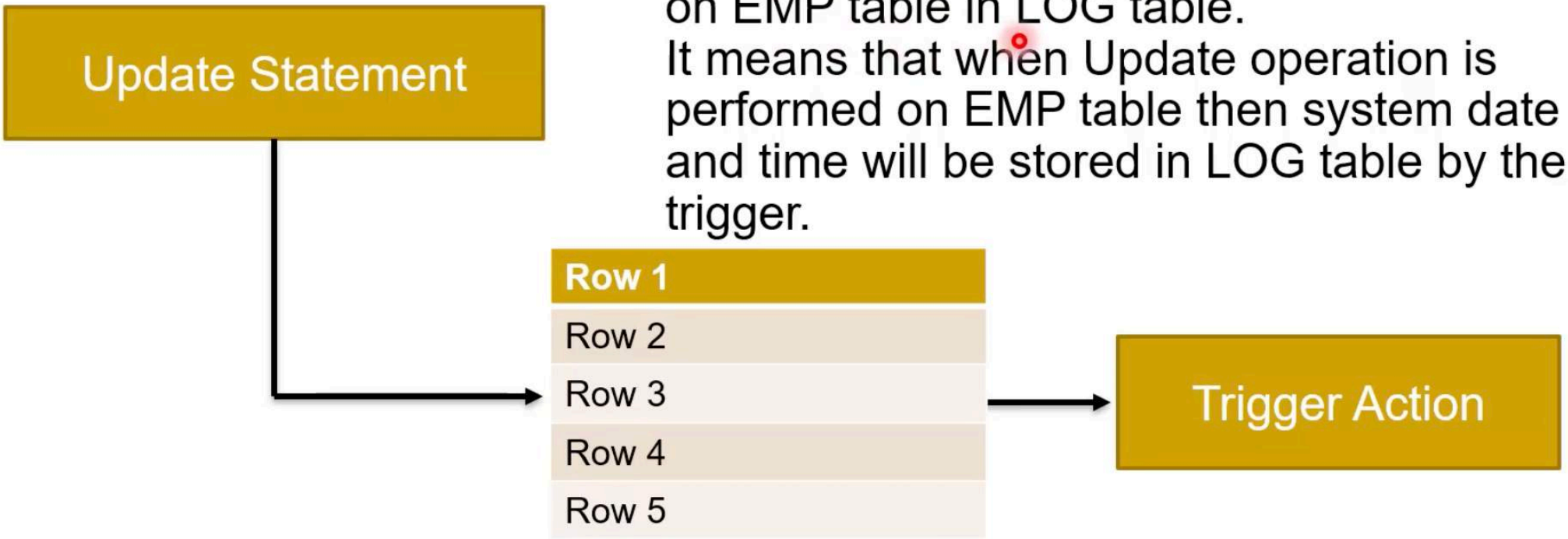
After Statement Level Trigger

AFTER trigger executes the trigger action after the triggering statement is executed. AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.



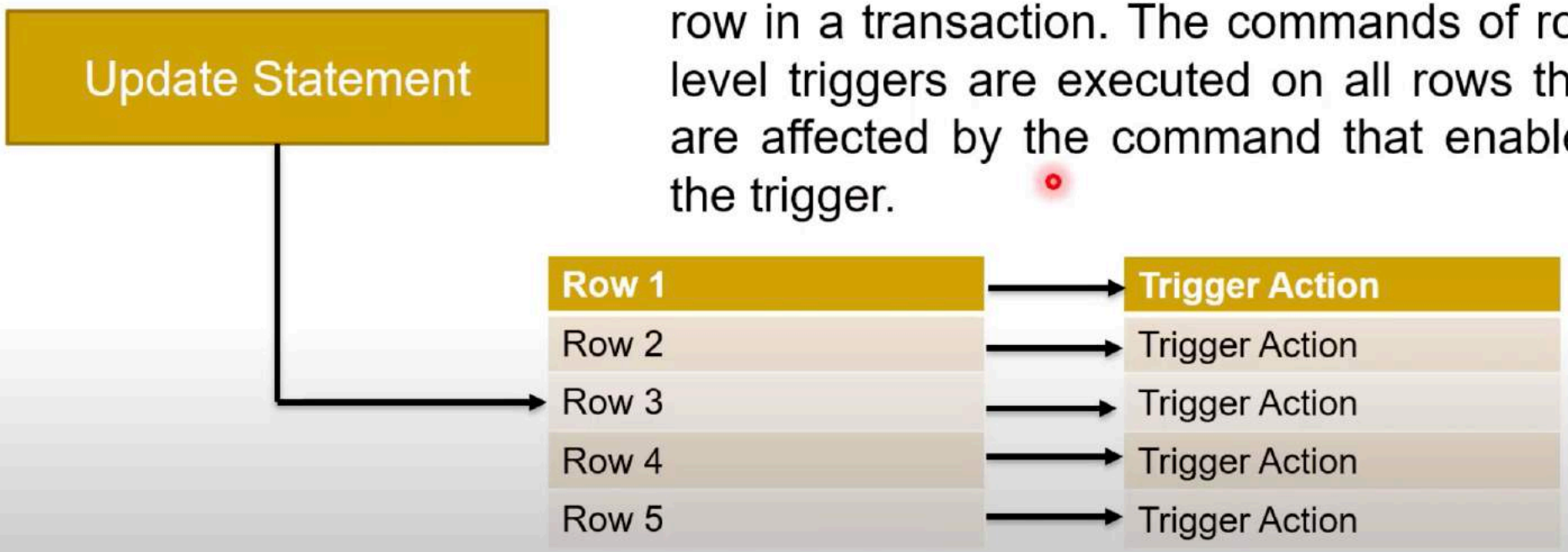
Application: After Statement Level Trigger

To store date and time of Update operation on EMP table in LOG table.
It means that when Update operation is performed on EMP table then system date and time will be stored in LOG table by the trigger.



After Row Level Trigger

Row level triggers execute once for each row in a transaction. The commands of row level triggers are executed on all rows that are affected by the command that enables the trigger.



Before and After Trigger

Name	Statement Level	Row Level
BEFORE option	Oracle fires the trigger only once, before executing the triggering statement	Oracle fires the trigger before modifying each row affected by the triggering statement
AFTER option	Oracle fires the trigger only once, after executing the triggering statement	Oracle fires the trigger after modifying each row affected by the triggering statement

Parts of a Trigger

- Triggering event or Statement.
- Trigger constraint (Optinal)
- Trigger action

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER}
{| DELETE | [OR] INSERT | [OR] UPDATE [OF column_name, …]
ON table_name
[REFERENCING {OLD AS old, NEW AS new}]
[FOR EACH ROW [WHEN condition ]
DECLARE
Variable declaration;
Constant declaration;
BEGIN
        PL/SQL subprogram body;
[EXCEPTION
        exception PL/SQL block;]
END;
```

Triggering event

Triggering restriction

Triggering action

Compilation of Trigger

- If an error occurs during the compilation of trigger, an invalid trigger is created.
- These errors can be viewed using the following commands:

SELECT * FROM user_errors;

Or

Show errors;

Dictionary Information about Triggers

- Oracle has provided us many dictionary views to show information about the triggers.
- We can query USER_TRIGGERS to view information of the trigger.

Dictionary Information about Triggers

Columns of USER_TRIGGERS

- TRIGGER_NAME
- TRIGGER_TYPE
- TRIGGERING_EVENT
- TABLE_NAME
- REFERENCING_NAMES
- WHEN_CLAUSE
- STATUS
- TRIGGER_BODY

Dictionary Information about Triggers

- SQL > select trigger_name,trigger_type, triggering_event
  from USER_TRIGGERS
  Where TABLE_NAME ='EMP';
- SQL > select * from USER_TRIGGERS
  Where TABLE_NAME ='EMP';

Enabling and Disabling Triggers

- When a trigger is created it is automatically enabled and is triggered whenever the triggering command and the execution command is true.
- To disable the execution use the ALTER TRIGGER command with the DISABLE clause. A disable trigger does not execute the trigger body even if the triggering statement is issued.
- ALTER TRIGGER <trigger name> DISABLE / ENABLE
- For example:
- ALTER TRIGGER EMP_UPDATE DISABLE;

Dropping Triggers

- Triggers may be dropped via the drop trigger command.
- In order to drop a trigger, you must either own the trigger or have the DROP ANY TRIGGER system privilege.

Syntax:

- DROP TRIGGER trigger_name;
- For Example:
- DROP TRIGGER EMP_UPDATE;

Concept of New and Old Keywords

- Trigger has a capability to access both old and new values of a column.

ENO	ENAME	SALARY
1	RAM	50000
2	RAJ	60000
3	RAJESH	80000

ENO	ENAME	SALARY
1	RAM	90000
2	RAJ	60000
3	RAJESH	80000

UPDATE EMP SET SALARY=SALARY+40000 WHERE ENO=1;

Concept of New and Old Keywords

- Trigger for Update Statement can access both old and new values of column by using :old.column_name and :new.column_name

ENO	ENAME	SALARY
1	RAM	50000
2	RAJ	60000
3	RAJESH	80000

ENO	ENAME	SALARY
1	RAM	90000
2	RAJ	60000
3	RAJESH	80000

TRIGGER BODY

:OLD.SALARY **:NEW.SALARY**

UPDATE EMP SET SALARY=SALARY+40000 WHERE ENO=1;

Concept of New and Old Keywords

- Trigger for Insert Statement can access new values of column by using :new.column_name, while :old.column_name will be NULL.

ENO	ENAME	SALARY
1	RAM	50000
2	RAJ	60000
3	RAJESH	80000

ENO	ENAME	SALARY
1	RAM	90000
2	RAJ	60000
3	RAJESH	80000
4	SHAM	70000

TRIGGER BODY
:OLD.SALARY IS NULL WHILE **:NEW.SALARY IS 70000.**

INSERT INTO EMP VALUES(4, 'SHAM',70000)

Concept of New and Old Keywords

- Trigger for Delete Statement can access old values of column by using :old.column_name, while :new.column_name will be NULL.

ENO	ENAME	SALARY
1	RAM	50000
2	RAJ	60000
3	RAJESH	80000

ENO	ENAME	SALARY
1	RAM	90000
2	RAJ	60000

TRIGGER BODY
:OLD.SALARY IS 80000 WHILE :NEW.SALARY IS NULL

DELETE FROM EMP WHERE ENO=3;

Having own names for keywords for Old and New

REFERENCING:

- Specified correlation names. The user could use the Correlation names in the PL/SQL block.
- The default correlation names are OLD and NEW.
- If the row is associated with a table named OLD or NEW, this clause can be used to specify different names to avoid confusion between the table name and the correlation name.

Having own keywords for Old and New

- Referencing old as previous New as latest

- Use :previous.column_name to access old values
- Use :latest.column_name to access new values

Create a trigger on EMP table, so that it converts supplied values of ENAME column to uppercase for INSERT and UPDATE statement.

- INSERT INTO EMP VALUES(1,'ram','prof',90000,10);
- UPDATE EMP SET ENAME='raj';

TRIGGER ACTION
It will access Ename value given in Insert statement and convert it to upper case

→

INSERT INTO EMP
VALUES(1,'ram','prof',90000,10);
UPDATE EMP SET ENAME='raj';

Create a trigger on EMP table, so that it converts supplied values of ENAME column to uppercase for INSERT and UPDATE statement.

- INSERT INTO EMP VALUES(1,'ram','prof',90000,10);
- UPDATE EMP SET ENAME='raj';

TRIGGER ACTION
:new.ename : = UPPER (:new.ename);

→

INSERT INTO EMP
VALUES(1,'ram','prof',90000,10);
UPDATE EMP SET ENAME='raj';

- Trigger will be of Before Type and it will perform its action for every row inserted before INSERT or UPDATE statement, so it will trigger FOR EACH ROW.

Create a trigger on EMP table, so that it converts supplied values of ENAME column to uppercase for INSERT and UPDATE statement.

```
CREATE OR REPLACE TRIGGER emp_upper
BEFORE INSERT OR UPDATE OF ename
ON emp
FOR EACH ROW
BEGIN
   :new.ename :=UPPER(:new.ename);
END;

INSERT INTO EMP VALUES(1,'ram','prof',90000,10);
UPDATE EMP SET ENAME='raj’;
```

//livesql.oracle.com/apex/f?p=590:1:12570879552785::NO:::

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

```
create table emp(eno number(3), ename char(20), job char(20), salary number(9), deptno number(2));
CREATE OR REPLACE TRIGGER emp_upper
BEFORE INSERT OR UPDATE OF ename
ON emp
FOR EACH ROW
BEGIN
    :new.ename :=UPPER(:new.ename);
END;

INSERT INTO EMP VALUES(1,'ram','prof',90000,10);
UPDATE EMP SET ENAME='raj';
select * from emp;
```

ENO	ENAME	JOB	SALARY	DEPTNO
1	RAJ	prof	90000	10

Having own keywords for Old and New

- Referencing old as previous New as latest

- Use :previous.column_name to access old values
- Use :latest.column_name to access new values

Feedback Help parteek.bhatia@gma

SQL Worksheet

Clear Find Actions Save

```
create table emp(eno number(3), ename char(20), job char(20), salary number(9), deptno number(2));
CREATE OR REPLACE TRIGGER emp_upper
BEFORE INSERT OR UPDATE OF ename
ON emp
Referencing New as latest
FOR EACH ROW
BEGIN
    :latest.ename :=UPPER(:latest.ename);
END;

INSERT INTO EMP VALUES(1,'ram','prof',90000,10);
UPDATE EMP SET ENAME='raj';
select * from emp;
```

ENO	ENAME	JOB	SALARY	DEPTNO
1	RAJ	prof	90000	10

Dictionary Information about Triggers

- SQL > select trigger_name,trigger_type, triggering_event
  from USER_TRIGGERS
  Where TABLE_NAME ='EMP';
- SQL > select * from USER_TRIGGERS
  Where TABLE_NAME ='EMP';

livesql.oracle.com/apex/f?p=590:1:12570879552785::NO:::

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_OWNER	BASE_OBJECT_TYPE	TABLE_NAME	COLUMN_NAME	RE
EMP_UPPER	BEFORE EACH ROW	INSERT OR UPDATE	SQL_LDLSPDWKGTQAGJNTQIRLHHYAZ	TABLE	EMP	-	RE AS OL

Download CSV

vesql.oracle.com/apex/f?p=590:1:1257087955278 5::NO:::

Feedback Help parteek.bhatia@gmail.com

SQL Worksheet

Clear Find Actions Save Run

WHEN_CLAUSE	STATUS	DESCRIPTION	ACTION_TYPE	TRIGGER_BODY	CROSSEDITION	BEFORE_STATEMENT	BEFORE_ROW	AFTER_
-	ENABLED	emp_upper BEFORE INSERT OR UPDATE OF ename ON emp Referencing New as latest FOR EACH ROW	PL/SQL	BEGIN :latest.ename :=UPPER(:latest.ename); END;	NO	NO	NO	NO

Create a trigger on the emp table, which shows the old values and new value of ename after every updation on ename of emp table.

- Trigger Type
- Timing:

  AFTER
- Level:

  FOR EACH ROW
- Action:

  To display values of :old.ename and :new.ename

Create a trigger on the emp table, which shows the old values and new value of ename after every updation on ename of emp table.

```
CREATE OR REPLACE TRIGGER EMP_UPDATE
AFTER UPDATE OF ENAME
ON EMP
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('OLD NAME: ' ||:OLD.ENAME);
  DBMS_OUTPUT.PUT_LINE('NEW NAME:' ||:NEW.ENAME);
END;
```

Feedback Help parteek.bhatia@

SQL Worksheet

Clear Find Actions Save

```
CREATE OR REPLACE TRIGGER EMP_UPDATE
AFTER UPDATE OF ENAME
ON EMP
FOR EACH ROW
BEGIN
    DBMS_OUTPUT.PUT_LINE('OLD NAME: ' ||:OLD.ENAME);
    DBMS_OUTPUT.PUT_LINE('NEW NAME:' ||:NEW.ENAME);
END;
select * from emp;
UPDATE EMP SET ENAME='RAM';
```

```
1 row(s) updated.
OLD NAME: RAJ
NEW NAME:RAM
```

Create a trigger so that no operation can be performed on EMP table on Sunday.

```
TYPE OF TRIGGER:
TIMING:
   BEFORE
EVENTS:
   INSERT OR UPDATE OR DELETE
LEVEL:
   STATEMENT LEVEL
ACTION:
   DAY:=TRIM((TO_CHAR(SYSDATE,'DAY'));
   --MAX LENGTH IN CASE OF ‘WEDNESDAY’ i.e 9 characters. So, SUNDAY WILL BE
   --STORED LIKE ‘SUNDAY   ’
   IF DAY=‘SUNDAY’ THEN
```

Create a trigger so that no operation can be performed on EMP table on Sunday.

```
CREATE OR REPLACE TRIGGER EMP_SUNDAY
BEFORE INSERT OR UPDATE OR DELETE
ON EMP
DECLARE
DAY VARCHAR(20);
BEGIN
    DAY:=TRIM((TO_CHAR(SYSDATE,'DAY'));
    IF DAY='SUNDAY' THEN
    DBMS_OUTPUT.PUT_LINE('NO OPERARTION CAN BE PERFORMED ON
    SUNDAY');
   END IF;
END;
```

Just raises a warning message but does not do anything to stop it.

Error Handling in Triggers

- The Oracle engine provides a procedure named *raise_appliation_error* that allows programmers to issue user-defined error messages.
- This procedure terminates the corresponding statement, rolls back its effects, and returns a user-specified error number and message.

Syntax:

RAISE_APPLICATION_ERROR (*error_number, message);*

Here:

error_number is a negative integer in the range-20000 to –20999

Message is a character string up to 2048 bytes in length

Create a trigger so that no operation should be performed on EMP table on Sunday.

```
CREATE OR REPLACE TRIGGER EMP_SUNDAY
BEFORE INSERT OR UPDATE OR DELETE
ON EMP
DECLARE
DAY VARCHAR(20);
BEGIN
   DAY:=TRIM((TO_CHAR(SYSDATE,'DAY'));
   IF DAY='SUNDAY' THEN
   RAISE_APPLICATION_ERROR(-20022,'NO OPERATION CAN BE PERFORMED ON
   EMP TABLE ON SUNDAY');
   END IF;
END;
```

Create a trigger on the emp table, which should store the ENO and name of operation performed on this ENO, i.e., Insert, Update or Delete and date on which this operation is performed on the EMP table in another EMP_AUDIT table.
EMP_AUDIT(ENO, NAME_OF_OPEARTION, DATE_OF_OPERATION)

- TYPE OF TRIGGER
- TIMING:

  AFTER
- LEVEL:

  FOR EACH ROW
- ACTION:

  We have keywords INSERTING, UPDATING and DELETING

  It will return TRUE when corresponding operation is performed on the table.

Create a trigger on the emp table, which should store the ENO and name of operation performed on this ENO, i.e., Insert, Update or Delete and date on which this operation is performed on the EMP table in another EMP_AUDIT table.

EMP_AUDIT(ENO, NAME_OF_OPEARTION, DATE_OF_OPERATION)

```
CREATE OR REPLACE TRIGGER EMP_AUDIT_TRIGGER
AFTER INSERT OR UPDATE OR DELETE ON EMP
FOR EACH ROW
 BEGIN
    IF INSERTING THEN
          INSERT INTO EMP_AUDIT VALUES(:NEW.ENO,'INSERT', SYSDATE);
          ELSIF UPDATING THEN
          INSERT INTO EMP_AUDIT VALUES(:NEW.ENO,'UPDATE', SYSDATE);
          ELSIF DELETING THEN
          INSERT INTO EMP_AUDIT VALUES(:OLD.ENO,'DELETE', SYSDATE);
    END IF;
 END;
```

s://livesql.oracle.com/apex/f?p=590:1:10941748010673::NO:::

Feedback Help parteek.bhatia

SQL Worksheet

Clear Find Actions Save

```
BEGIN
    IF INSERTING THEN
    INSERT INTO EMP_AUDIT VALUES(:NEW.ENO,'INSERT', SYSDATE);
    ELSIF UPDATING THEN
    INSERT INTO EMP_AUDIT VALUES(:NEW.ENO,'UPDATE', SYSDATE);
    ELSIF DELETING THEN
    INSERT INTO EMP_AUDIT VALUES(:OLD.ENO,'DELETE', SYSDATE);
    END IF;
END;
INSERT INTO EMP VALUES(2,'RAJ','PROF',100000,10);
SELECT * FROM EMP_AUDIT;
UPDATE EMP SET SAL=90000;
DELETE EMP;
```

ENO	NAME_OF_OPEARTION	DATE_OF_OPERATION

Problem Statement

- To enforce that user only insert values of Cno, Meter_no, Prev_Reading and Current_Reading while Units and Bill_Amount will be automatically calculated and inserted along with these values for CUST table. CUST(Cno, Meter_no, Prev_Reading, Current_Reading, Units, Bill_Amount)
- INSERT INTO CUST VALUES(1, 100, 300,350,NULL, NULL);
- Units will be calculated by
- Units=Current_Reading-Prev_Reading
- Bill will be charged according to following rate:
- First 100 Units 0.5 Rs per unit and after 0.75 Rs for every unit consumed.

```
CUST(Cno, Meter_no, Prev_Reading, Current_Reading, Units, Bill_Amount)
INSERT INTO CUST VALUES(1, 100, 300,350,NULL, NULL);
CREATE OR REPLACE TRIGGER INSERT_CUST BEFORE INSERT ON CUST
FOR EACH ROW
DECLARE
        B NUMBER;
        U NUMBER;
        EXTRA NUMBER;
BEGIN
        U:=:NEW.CURRENT_READING-:NEW.PREV_READING;
        IF U<100 THEN
                B:=U*0.5;
        ELSE
                EXTRA:=U-100;
                B:=100*0.5+EXTRA*0.75;
        END IF;
        :NEW.UNITS:=U;
        :NEW.BILL_AMOUNT:=B;
END;
```

Create a trigger to implement the primary key constraint on column ENO of table EMP. EMP(eno, ename, job, sal, deptno).

```
TYPE OF TRIGGER
TIMING:
    BEFORE
LEVEL:
    FOR EACH ROW
ACTION:
INSERT INTO EMP VALUES(1,'RAM','PROF',100000,10)
PRIMARY KEY ENO SHOULD NOT BE NULL
IF (:NEW.ENO IS NULL) THEN
    RAISE_APPLICATION_ERROR( );
END IF;
ENO SHOULD BE UNIQUE
QUERY THE EMP TABLE TO CHECK PRESENCE OF RECORD OF :NEW.ENO;
IF EXISTS RAISE_APPLICATION_ERROR( );
ELSE NULL;
END IF;
```

Create a trigger to implement the primary key constraint on column ENO of table EMP. EMP(eno, ename, job, sal, deptno).

```
CREATE OR REPLACE TRIGGER PRIMARY_KEY
BEFORE INSERT OR UPDATE OF ENO ON EMP
FOR EACH ROW
DECLARE
    REC EMP%ROWTYPE;
BEGIN
    IF (:NEW.ENO IS NULL) THEN
    RAISE_APPLICATION_ERROR(-20002, 'PRIMARY KEY VIOLATION BECAUSE IT CANNOT BE
    NULL');
    END IF;
    SELECT * INTO REC FROM EMP WHERE ENO=:NEW.ENO;
    RAISE_APPLICATION_ERROR(-20003, 'PRIMARY KEY VIOLATION BECAUSE IT SHOULD BE
    UNIQUE');
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    NULL;
END;
```