**Characteristics of plsql**

High performance: plsql send block of sql statements between application and and database server. This increases the performance

Plsql allows

**What is %type %rowtype and type record?**

%type can be used to declare the type of a variable. This type is as same as the type for previously declared variables or table columns. We do not need to know what the type is. When the data type for referenced item get changed, the data type for referring item get changed as well.

%rowtype: %rowtype is the table-based record, it returns the entire columns of a table or a view..

Type record: type record is the programmer defined record. It returns some columns from different tables or views.

**What are the data types available in plsql**

1. Scalar data type: is the data type is made up with a single value

For example: the sql data types : varchar number Boolean date char

Rowid: rowid represents the unique physical address of the row in a table

1. Composite data type:is made up of other data types (has internal component). Example: table, varray, record
2. Reference data type: REF cursor
3. Large object data type bfile BLOB CLOB

**What are the difference between char and varchar**

If the datatype receiver is char, plsql blank-pads the value to the maxium size. If the receiver is varchar2, plsql won’t do the blank-pad.

Pls\_integer vs number: 1 pls\_integer values require less storage. 2.the calculation is faster than number

SIMPLE\_INTEGER is a subtype of PLS\_INTEGER which has the same range as pi but has a NOT NULL constraint

**What are the difference between EXIT and CONTINUE?**

EXIT STATEMENT: it exits the current iteration of the loop and transfer the control to the end of the loop, say it breaks the loop as Java

CONTINUE STATEMENT: it exits the current iteration of the loop and transfer the control to the next iteration of the loop. The loop is not interrupted.

**Sequential control statements GOTO**

GOTO STATEMENT: GOTO is always used with label, it transfer the control to a label unconditionally.

GOTO constrains:1. Donot use GOTO in a complex nested structure, it makes it harder to maintain the program. 2. Cannot use GOTO transfer the control into a IF statement.

**User-defined subtype**

There are two types of user-defined subtype, the subtypes has scalar datatype as the base type

1. Unconstrained
2. Constrained including NOT NULL ,RANGE,

Note: the only base type can be used to declare a range subtype is PLS\_INTEGER

**Composite datetype: Collections and Record**

Collection: in a collection the internal components always have a same data type, we call them the element of a collection. The element can be accessed through its unique index

Collection types:

1. Associated array: is a set of key-value pairs, each key has an unique index. An associated array can hold unspecified number of elements, the elements can be accessed without knowing its position

When to use:

1. a relatively small lookup table

2. Passing the collection to and from database server(FORALL & BULK COLLECT)

Conclusion: the associated array is used for a temporary data storage

1. VARRAY: is an array which contains zero to maximum number of elements

Index: the lower bound is the 1 and the upper bound is the number of elements, the upper bound cannot exceed maximum size of array

Constructor: the uninitialized varray is null, we have to initialized it in order to use it

When to use:

1. You know the max number of elements varray contains

2. usually access the elements sequentially.

Conclusion: because you must retrieve and store the all elements at the same time, so varray is normally not used for large numbers of elements

1. Nested table: it is a column type stores unspecified number of rows in no particular order.

Index: plsql gives rows consecutive index, start with1, we can access a single row by using index. but the index and row order of nested table might be changed.

Constructor: the uninitialized nested table is null, we have to initialize it in order to use it

When to use:

1. the number of elements is not to set

2. Index value are not consecutive

3. You must delete or update some elements but not all the elements simultaneously

4. You could create a separate look up table, with multiple entries for each row of the main table and access it through join quires.

**Differences between Nested table and array**

1. An array has a declared number of elements, but the size for nested table can increase dynamically.
2. An array is always dense, the nested table is dense initially, but it can become sparse when deleting elements

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Collection type | Number of elements | Index type | Dense or sparse | Unintialized status | Where defined | Can be ADT |
| Associated array | Not specified | String or pls\_integer | Either | empty | In plsql block or package | No |
| Varray | Specified | Number | dense | null | Plsql block, package or schema level | Only when defined at schema level |
| Nested table | Not specified | Number | Started with dense but canbe sparce | null | Plsql block  Package  Schema level | Only when defined at schema level |

Collection constructor: is the system defined function with the same name as collection type, which returns a collection of that type. It is used to initialize a varray or nested table.

SET & MULTISET (nested table):

1. SET: convers the nested table into a set by eliminate the duplicates
2. MULTISET : combine two nested tables into one nested table. Can be used with union, intersect, except
3. CARDINALITY: returns number of elements for a nested table. If the nested table is null, it returns null
4. SUBMULTISET OF: example: nt1 SUBMULTISET of nt2 means if nt2 includes nt1 (retrun Boolean)
5. MEMBER OF : see if an element is in a nested table (return Boolean)
6. IS A SET: nt1 SI A SET means if the elements in nt1 are all distinct (return Boolean)

Multidimensional collections:

It can be achieved by making an array as an element in an array

Collection comparison:

1. You cannot compare a associated array with null
2. You cannot compare two collection variables with relational operators , also the collection variables cannot appear in group by, distinct or order by clause
3. Tow nested table variables are equal when they have same set of elements(in any order)
4. If two nested table variables have same nested table type, and that nested data type does not have elements of record type, then we can compare two nested table variable with relational operators

Collection methods:

1. DELETE(index): delete an element form collection (cannot use delete on a varray)
2. TRIM(number): delete an element from end of a varray or nested table
3. EXTEND: add elements to the end of varrray or nested table (extend(n) append n null elements; extend(n,i) append n I elements copy in the end of collection)
4. EXISTS(element): return true if and only if the specific element index is existing in a varray or nested table
5. FIRST: return first index of a collection (lower bound)
6. LAST: return last index of a collection (upper bound)
7. COUNT:returns number of elements in a collection
8. LIMIT: returns the max number of elements a collection can have(returns null if there is no max size)
9. PRIOR: return index that precedes specified index
10. NEXT: returns index that successed the specified index

Collection type defined in package specification:

A collection type defined in package is not compatible with the identical collection type defined in local plsql block or db schema

**Record variables**

Record: in record the internal components can have different data types and are called fields. We can access each fields by its name **Note : the record viable can only contain the information for one single row. This is different from collections**

The record variable can be created in three ways:

1. Define a record type then define a variable of that type. (define a type record)
2. Use %rowtype to declare a record variable that represents either a full or partial row of a table or a view
3. Use %type to declare a variable has the same type as a previously declared record variable or table column

For a variable of a record type, the initial value for each field is null.

The record type can be defined in plsql block and package, cannot define record type on schema level .means record type cannot be ADT

**%rowtype** :

1. Declare a record type represents the full row of a table, the fields have same name and data type as columns in a row : e.g.: emp employees%rowtype
2. Declare a record type represents partial row of a table : use cursor associated with a query select partial columns of a table.

e.g. define a cursor with selected columns on a table. Then emp cursor\_name%rowtype then fetch cursor into variable

3. %rowtype on a cursor can also define a type represents join row

4. if there is a column on table is invisible, the %rowtype can only represents those visible columns, we have to make the column visible before select full row into the record variable.

**Assign values to record type:**

1. Assign one record variable to another
2. Select into
3. Fetch cursor into e.g. cursor [] return [record type] is select … Note: we can fetch a cursor into a record variable without return
4. RETURNING INTO : DML has RETURNING INTO clause to return all affected rows in a plsql record variable
5. Null can be assigned to a record variable by [record variable] := null;

Compare record :My\_record is null ; rc1 = rc2 ; rc1>rc2

**Record inserting and update**

A record variable can be inserted into a table : INSERT INTO [TABLE] VALUES [RECORD VARIABLE]

A record variable can be used to update the table row:

UPDATE [TABLE] SET ROW= [RECORD VARIABLE] WHERE …

**Static SQL**

Statements:

1. Select – query
2. DML (see merge later)

**Merge :** can be used to update and insert rows into a table at the same time. If the records exists the update otherwise insert

1. TCL

Pseudocolums : behaves like table columns but not stored in a table

CURRVAL & NEXTVAL : use with sequence

LEVEL :

OBJECT\_VALUE

ROWID : this is the physical address of a row in disk

ROWNUM: A build on SQL function that ordered rows from select query

**What is cursor?**

Cursor is a pointer to a private sql area (context area) that stores information about processing the specific select and DML

(returning from the execution of DML and select statement)

**Implicate cursor:**

1. The implicate cursor is automatically created by Oracle db when a select or DML statement executed
2. The session cursor is opened every time running a select or DML statement, and it automatically close after the execution of the statement
3. The programmer cannot control it
4. When a DML statement get executed, the cursor is associated with those statements. When insert into a table, it holds the data to be inserted. When update or delete the record. It identified affected rows.

There are some attributes for implicated cursor also known as sql cursor:

%notfound return true if there is no row affected by sql statement

%rowcount return the number of rows get affected by sql statement

%found : return true if there are rows affected by sql statements

%isopen: always return false as the cursor get closed automatically after the execution of sql statement

**Explicit cursor**

Explicit cursor is the programmer defined cursor. It helps programmer gain the control of context area.

There are four steps of creating explicit cursor:

1. Declare cursor: associated with a select statement

a. Can be declared on plsql block, packages, subprograms

2. Open cursor: allocate memory for the cursor, makes the cursor get ready for fetching the row returned by select statement

3. Fetch cursor: access rows hold by cursor once a time (associated with a query)

a. The fetch statement retrieves the current row of a result set returned by query

b. Fetch statement is usually used in a loop

c. Fetch statement can be used with SELECT BULK COLLECT INTO statement for

4. Close cursor: free the allocated memory

Explicit cursor that accepts parameters:

Formal parameter is the parameter when declare the cursor, Actual parameter is the passed in parameter, pass actually parameter when open the cursor

Explicit cursor attribute

%ISOPEN : is the cursor open?

%FOUND: if there is any row fetched

%NOTFOUND if there is no row fetched

%ROWCOUNT how many row have been fetched

**Processing query result sets**

Cursor can be used to process result set, we can use both explicit and implicit cursor to achieve this

1. SELECT INTO to be used when return one row

SELECT BULK COLLECT INTO when return multiple rows

1. Processing data result set through FOR LOOP

FOR LOOP can be used either implicit cursor or explicit cursor

Note: when an exception raised in the loop, the cursor is closed before program get into exception handling section. It means we cannot use the cursor attribute in exception handling section

Note2: cause there is no fetch clause, we do not need to open a cursor

Even though it is easy to use implicit cursor FOR LOOP, it is more flexible to use explicit cursor FOR LOOP

1. We can process multiple result sets in parallel by using multiple cursors
2. Process multiple rows in one iteration of a loop
3. We can pass parameters into an explicit cursor
4. Processing result sets with subqueries

**Cursor Variables**

The cursor variable is like an explicit cursor except that:

1. It is not limit to one query : you can open a cursor variable for a query process the result set then use the cursor variable for another query
2. You can assign value to cursor variable
3. Cursor variable can be used as a parameter to pass result sets through subprograms
4. Cursor variable can be used to pass result set between plsql and client application
5. Cursor variable can be used in an expression
6. It cannot accept parameters

A cursor variable is a pointer, before reference a cursor variable, you must make it point to a sql work area. This can be done by assigning a value or opening it

Cursor variable and explicit cursor is not interchangeable. (Cursor variable cannot replace the explicit cursor). you cannot use cursor variable where the explicit variable expected.

Note: cursor type cannot be in collection, record and object

The cursor variable is also called **REF CURSOR**

1. Declare cursor variable

Weak cursor type

1. Without return clause when define a cursor variable
2. Can be associated with any query
3. More flexible but easier to make error
4. The value of a weak cursor variable can be assigned to any weak cursor variable
5. SYS\_REFCURSOR is a weak cursor type

Strong cursor type

1. With return clause then define cursor variable
2. Can be only associated with queries return specified type
3. You can assign the value of a strong cursor variable to another only if they have same data type
4. Open cursor variable
5. open cursor viable via OPEN FOR clause
6. it associates the cursor variable with a query
7. allocate memory to process the query
8. process query: identifies the result set , if query has FOR UPDATE clause, lock the row of the table
9. position the cursor before the first row of a result set
10. close the cursor variable
11. close it when the cursor variable is no longer to be used
12. free the memory of processing query
13. you do not need to close the cursor variable before reopen it
14. fetching data with cursor variables
15. after open the cursor, you can fetch the rows of result set
16. assigning values to cursor variable
17. target\_cursor\_variable := source cursor variable;
18. if the source cursor variable is open, the target variable will be open as well
19. if the source cursor v is not open, then the target cursor v cannot be opened.
20. Variables in cursor variable query,

If a cursor query used a variable, if the variables number changed, the result set cannot be changed until reopen the cursor variable with the same query.

1. Querying a collection

The data type of the collection element must be declared at schema level or in a package.

1. Cursor variable attributes : as same as explicit cursor
2. Cursor variables as subprogram parameter
3. Open a cursor variable in one subprogram and process it in another subprogram
4. Plsql subprogram can use a cursor variable to return a result set to a subprogram written in other language(e.g. java)
5. If the subprogram opens or assign values to cursor variable the parameter mode must be IN OUT
6. If the subprogram only fetch the row from result set or close the cursor variable, then the parameter mode can be IN or IN OUT
7. Define a REF CURSOR type in a package the rest parameters can reuse it.
8. Cursor variable as a host variable
9. When a cursor variable is a host variable, plsql and client share a pointer points to the context area that stores result set
10. Use plsql anonymous block to open and close several cursor variables in a single round trip to reduce network traffic

This technique is useful when populating a multi-block form

1. Using ref cursor on java application

Cursor expression

1. A cursor expression returns nested cursor
2. The cursor expression can only be used on explicit cursor and cursor variables
3. Passing cursor expression to pipelined table function
4. When SQL select pass a CURSOR expression to a function, the cursor is open when the function begins and close when the function completes

Transaction processing and control

A transaction is a sequence of one or more sql statements that oracle database treat as a unit

Transaction processing is a oracle database feature to let multiple user use the database at the same time. In order to ensure each user sees consistant data and all changes applied in a right order.

Oracle db will lock the data automatically

COMMIT makes the change permanent and visible to other users

ROLLBACK : ends the current transaction and undo any changes during the transaction, ROLLBACK usually used on EXCEPTION section

SAVEPOINT: makes database rollback the partial transactions instead of the full transaction

SET TRANSACTION : begin a read-only or read-write transaction

Read-only transaction refer a snapshot of the database the data won’t be changed when other user updates the data.

Read only cannot be used with update

Override default locking

1. LOCK table

Locks the whole table. Only one user can lock the table, other users cannot DML the table

1. Select for update clause

Select rows from result set and locks them

For update cursor with where current of : for update cursor can only be used with update clause with where current of

The fetch can be locked if there is no update statement on current iteration

**Autonomous transaction**

Autonomous transaction is an independent transaction started by other transaction, the main transaction.

It is usually in a subprogram, it does the commit and rollback without commit or rollback the main transaction

Advantage: autonomous transaction is fully independent it shares no locks resources or commit-dependeciers with the main transaction

Help you build reusable components, you can encapsulate the autonomous transaction in stroed subprograms a invoking application does not need to know if the transaction in subprogram is successful or failed.

PRAMGA AUTONOMOUS\_TRANSACTION

Widely used on log error will cause deadlock when AT is trying to access the data MT controls, cause when execute AT the MT is locked, after the execution the MT resumed

Autonomous trigger

The triggers has to be autonomous when run TCL and DDL statements

Dynamic SQL

It is a program methodology for generating and running sql statements at run time. It is useful when writing flexible system.

Relates to DDL statements

1. Native dynamic sql : NDS runs faster than DS , to write a NDS you must know the compile time, number and data type of the IN OUT variables of a sql statements
2. DBMS\_SQL (no longer to be used as the NDS is way easier)

WHEN:

1. SQL whose text is unknown at the compile time : e.g. SELECT statements missing table name, WHERE clause when the subcluases is unknown during compile
2. SQL that is not supported as static SQL (DDL)
3. Use static sql when you can use both: because static sql will have a successful compilation

Native dynamic SQL(NDS):

1. EXECUTE IMMIDAIATE (do not use table name as a bind variable, use || instead )
2. EXECUTE IMM (SELECT/UPDATE/INSERT SQL) USING (bind variables)
3. Multiple rows:
4. EXECUTE IMMIDAIATE [SAL ] BULK COLLECT INTO [collection]
5. EXECUTE IMM (SQL) in bind USING out bind RETURN INTO
6. OPEN FOR FETCH CLOSE statements

SQL injection

Use concatenation to build where clause then execute with dynamic sql. It is vulnerable because the string input is not fully validated.

query := 'SELECT value FROM secret\_records WHERE user\_name='''

|| user\_name

|| ''' AND service\_type='''

|| service\_type

|| '''';

DBMS\_OUTPUT.PUT\_LINE('Query: ' || query);

EXECUTE IMMEDIATE query INTO rec ;

DBMS\_OUTPUT.PUT\_LINE('Rec: ' || rec );

Make the SQL injection invulnerable

1. Use bind variable
2. Validation checks
3. Convert date type or numeric types to text if you cannot use bind varaibles, this can make sure the format consistency

**Subprograms**

What :

1. Subprogram is block of sql statements that can be invoked repeatedly.
2. Two type of subprogram : function and procedure
3. Function is used to compute and return a value
4. Procedure is used to perform an action

Why use subprogram:

1. Makes you break the program into manageable, well-defined modules (like method in java)
2. Easier application design
3. Easy to maintain : you can change the implementation detail of a subprogram without change the invoker
4. Packagebility: subprograms can be grouped in a package
5. Reusability: the subprogram in a package or schema level can be used on different applications and environments
6. Better performance: reduced the network traffic and improves invoking time

How:

Define:

Nested subprogram : defined in plsql BLOCK

Package subprogram : define in package (stored subprogram)

Standalone subprogram : defined in schema level (stored subprogram)

.

**-- Declare and define procedure**

PROCEDURE create\_email ( **-- Subprogram heading begins**

name1 VARCHAR2,

name2 VARCHAR2,

company VARCHAR2

) **-- Subprogram heading ends**

IS

**-- Declarative part begins**

error\_message VARCHAR2(30) := 'Email address is too long.';

BEGIN **-- Executable part begins**

email := name1 || '.' || name2 || '@' || company;

EXCEPTION **-- Exception-handling part begins**

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE(error\_message);

END create\_email;

**Difference between function and procedure:**

1. Function has to have a RETURN clause in heading
2. The execution part has to have a RETURN
3. Only function heading can include following options
4. PIPELINED :makes a table pipelined
5. DETERMINISTIC: helps optimizer avoid redundant invocations
6. PARALLEL\_ENABLE : enable parallel execution
7. RESULT CACHE: stores function result in plsql result cache

**RETURN Statement**

*Return statement in function:* every execution path must lead to a RETURN statement and every RETURN statement must specify an expression

Note : pipelined option does not need a RETURN

Note2: Return exits the function and return control to invoking statement

Note3: take care of return in if statement, it will cause warning if there is a case does not have return value

Return statement in procedure:

In a procedure, the RETURN statement returns the control to the invoker, the control returns right after the invocation

Return statement in anonymous block:

The return statement exits it’s own block and enclose blocks (exit inner and outter block at the same time). the return statement should not have expression

**Forward declaration**

Can be used when nested subprograms invoke each other, the one functions needs to be declared as forward declaration

**Parameters**

Formal parameter: the parameter on header

Actual parameter the parameter passed into the subprogram

If we use a subtype for defining formal parameter, then the not null and range constraint can be inherited to formal parameter, but the size of the varchar or the precise of number cannot be inherited

DECLARE

**SUBTYPE License IS VARCHAR2(7) NOT NULL**;

n License := 'DLLLDDD';

PROCEDURE p (**x License**) IS

BEGIN

DBMS\_OUTPUT.PUT\_LINE(x);

END;

BEGIN

p('1ABC123456789'); **-- Succeeds; size is not inherited**

p(NULL); **-- Raises error; NOT NULL is inherited**

END;

/

**Subprogram parameter passing methods**

2 ways to passing actual parameter:

1. By reference : passing the pointer of the parameter same memory location
2. By value: different memory location

Note the compiler automatically convert actual parameter to formal parameter with implicit data convert, it is NOT recommended.

**Parameter modes**

Determines parameter behaviors

|  |  |  |
| --- | --- | --- |
| **IN** | **OUT** | **IN OUT** |
| Default mode | Must be specified | Must be specified |
| Pass value to subprogram | Returns values to invoker | Pass the initial value to subprogram and return the updated value to invoker |
| Formal parameter like a constant the subprogram cannot change the value | Initialized with the default value of the formal parameter which null except those with not null constraint.  If the actual parameter assign a value to the formal parameter, the fp will ignore the value and use the intial value | The initial value is passed by actual parameter and this value is recommended to be updated in subprogram |
| Actual parameter can be constant ,initialized values , literal or expression | If the default value of formal parameter is null the actual parameter should not have not null constraint | Actuall parameter must be a vraible |
| Actual parameter is passed by reference | By default, the actual parameter is passed by value , if you specify NOCOPY it might be passed by a reference | By default, the actual parameter is passed by value , if you specify NOCOPY it can be passed by a reference |

Ideally, do not use IN OUT AND OUT parameters in a function, it will cause side effects

Parameter aliasing:

Having two different names for the same memory location.

1. Parameter passed by reference

e.g. if pass a actual parameter with a value to a IN OUT formal parameter passed in by reference. The program makes decision this returned parameter contains actual parameter value or the subprogram assigned value. Say it can be either actual p value or return value

1. Parameter passed by cursor variable

cursor variable cv1 and cv2 , if cv1 = cv2, when we close cv2, cv1 is also closed

default values for IN parameter (Note it is only for IN parameter, the value for this parameter is default value once the actual parameter is omitted)

**Positional, Named, Mixed notation for actual parameter:**

|  |  |  |
| --- | --- | --- |
| Positional | Named Notation | mixed |
| Use order | Use formal => actual | Start with positional, then use named notaion for rest |
| Can omit trailing optional parameters | Can omit any optional prameter | Combine for positional and named |
| The actual parameter in an wrong order will cause problem | No wrong order | combine |
| When subprogram formal parameter changes the invoker actual parameters have to be changed with the same order | Invoker’s actual parameter only changes when the new formal parameter added in subprogram |  |
|  | Recommended when invoke a subprogram defined or maintained by others | Convenient when you have a optional formal parameter |

**Subprogram invocation solutions**

Start with current scope (check the name- >check parameter match -> check if multiple match ) then go to enclosing scope, any failed check for 2 and 3 will cause compilation error

**Overloading subprograms**

You can put multiple procedures with same name in the same block package, plsql determines the invoked procedure by it’s formal parameter

For numeric data type: the varchar2 value can match number, binary\_float, binary\_double

Plsql looks for matching numeric parameters in this type :

1. PLS\_NUMBER
2. NUMBER
3. BINARY\_FLOAT
4. BINATY\_DOUBLE

Cannot overload cases:

1. you cannot overload standalone subprogram
2. the only difference is formal parameter mode
3. formal parameter only differ in subtype (in same type family)
4. functions that only differ in return type

**Recursive subprogram**

A recursive subprogram needs to have at least two execution paths: one leading to invoke the subprogram and leading to terminate subprogram

Subprogram side effects

**Result cached function for minimizing side effects and increase the performance:**

What:

When a result cache function invoked, the system checks cache, if the cache contains the result of previous invocation of the function with same parameter values the system returns cached result to invoker and does not need to execute the function body. If the cache does not contain the result, the system execute the function body and add the result to cache before running control to the invoker.

Two types of result cache:

1. function result cache
2. SQL query result cache

How:

Include RESULT\_CACHE in function definition

Note the RESULT\_CACHE\_MODE = FORCE can make database stores the result of all queries in SQL query result cache.

Developing applications with result-cached function: ???

1. The first time a session on the database instance invokes the function with these parameter values
2. When the cached result for thest parameter value is invalid
3. When the cached result for these parameter values are aged out.
4. When the function bypass the cache

**Restrictions on result cached function**

1. RESULT\_CACHE function cannot be invoked in plsql blocks or subprograms
2. It is not a pipelined function
3. It does not reference a dictionary tables (?),temporary tables, sequences or nondeterministic SQL functions
4. It has not OUT or IN OUT parameters
5. Not IN parameter with datatype, large object datatype (BLOB, CLOB..,) and ref cursor, collection, object , record
6. The return type is none of those large object datatype, REF CURSOR, object, record or collection contains unsupported data type.

Why: the best case for using result-caching are functions that are invoked frequently but depend on the information that changed infrequently. Result-caching avoids redundant computations on recursive functions

DBMS\_RESULT\_CACHE handled by DBA ?? need to have execute priv

**Invoker’s rights and definer’s rights**

AUTHID : determines if the unit IR or DR

for schema HR and OE , both of them have the table employees but have different data,

a function defined in HR have DR-> AUTHID DEFINER when OE invokes this function, the HR.table will get changed

if the function is IR: AUTHID CURRENT\_USER when OE invokes the function the OE.table will get changed.

**External subprogram**

You can call java or c method from plsql

AS LANAGUAGE JAVA NAME ‘java method’

-- Publish Adjuster.raiseSalary as standalone PL/SQL procedure:

CREATE OR REPLACE PROCEDURE raise\_salary (

empid NUMBER,

pct NUMBER

) AS

LANGUAGE JAVA NAME 'Adjuster.raiseSalary (int, float)'; -- call specification

/

BEGIN

raise\_salary(120, 10); -- invoke Adjuster.raiseSalary by PL/SQL name

END;

/

**What is trigger**

Trigger is a stored program that automatically get execute when some events occur

When you enable a trigger the database invoke it automatically when specified events occur, you cannot invoke trigger by yourself. This is the differ to the stored subprogram

Trigger event

Trigger statement

Create on / define on item (table, view, schema, or a database)

Timing point

**DML trigger:** the trigger event is compose of DML statements

**System trigger**: if the trigger is created on a schema or database, the trigger event is composed of either DDL or database operation statements (GRANT etc.)

Conditional trigger: has a when clause that specifies a SQL condition that the database evaluate each row that trigger statements affects.

Instead of trigger:

1. A DML trigger created on no editable views or a nested table column of a no editable view
2. A system trigger defined on CREATE statement

When a trigger fires, table that the trigger references might be undergoing changes made by SQL statements in other userr’s transaction

Reasons to use trigger:

1. Automatically generate virtual column value
2. Log events
3. Gather statistic on table access
4. Modify table data when DML a no editable view
5. Prevent invalid transactions
6. Prevent DML operations on a table after regular business hours

How trigger and constraint differ

Both trigger and constraint can constrain data input.

Differences：

1. A trigger always applies on new data only. The prevention start working after trigger enabled
2. A constraint applies on both new and existing data.

Use trigger data input constraint when:

Complex business that hard to define constraint

**DML trigger:**

Simple DML trigger

Timing points: BEFORE :statement level and row level

AFTER :statement level and row level

**Conditional predicates for detecting DML trigger statement**

INSERTING, UPDATING, UPDATING (‘COL’), DELETING

Conditional predicates are BOOLEAN

Correlation names

OLD NEW PARENT using REFERENCING TO change the correlation names of CREATE TRIGGER

Restrictions of pseudorecord

1. Cannot appear in a record-level operation (:new = null is not valid)
2. Cannot be a subprogram parameter
3. Cannot change OLD field values
4. If the trigger statement is delete then cannot change the NEW field values
5. An AFTER trigger cannot change NEW field values

Note: a BEFORE trigger can change the NEW field value and put the value into table

If a statement trigger both BEFORE and AFTER trigger the BEFORE trigger changed NEW values and AFTER

Trigger sees that value

**OBJECT\_VALUE Pseudocolumn**

A trigger on an object table can reference OBJECT\_VALUE which returns system generated named for the columns of the object table. The trigger can also invoke a PLSQL subprogram that has a formal IN parameter whose data type is OBJECT\_VALUE

**Instead of DML trigger**

Instead trigger is used for DML statements on no editable views ( the view composed with multiple tables).

1. Instead of trigger cannot be conditional
2. Instead of trigger can read NEW and OLD values but cannot change them

Instead of triggers on nested table columns of views