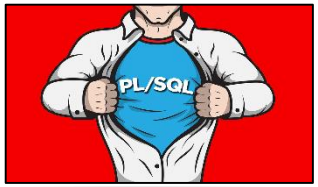
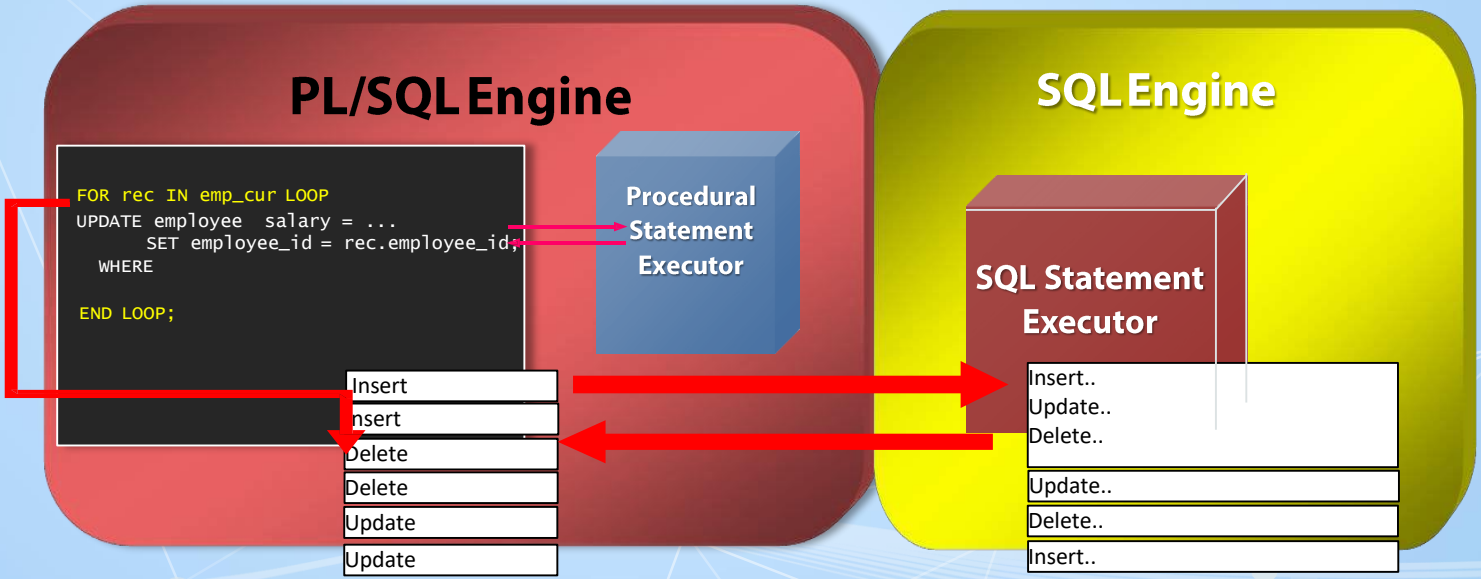
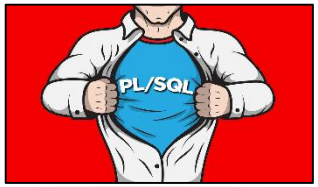


What is PL/SQL ?

1. What is PL/SQL?

- PL/SQL Stands for «**(P)**rocedural **(L)**anguage Extension to **SQL**»



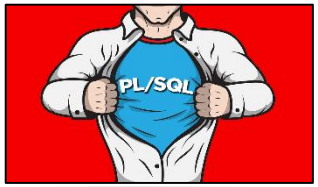


What is PL/SQL ?

- How will you have PL/SQL?

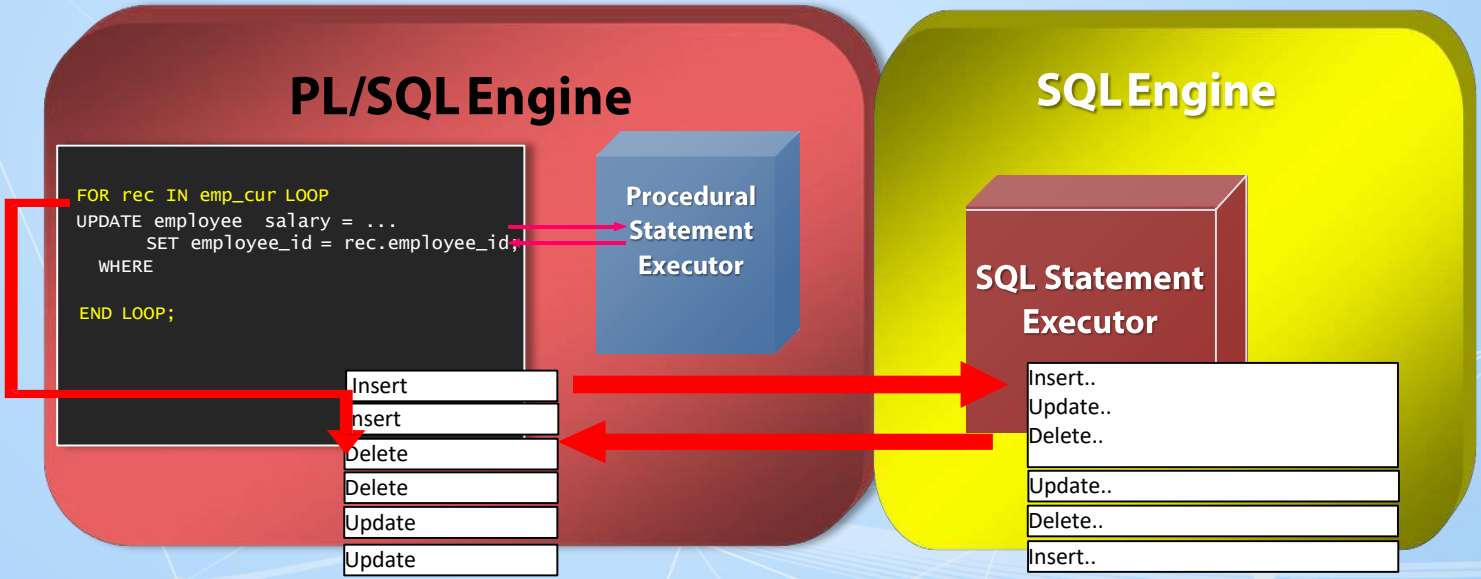
- Platform Independence

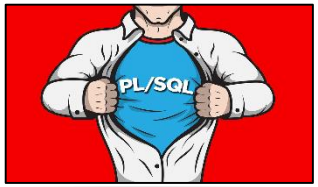
- So, Why PL/SQL?



PL/SQL Architecture

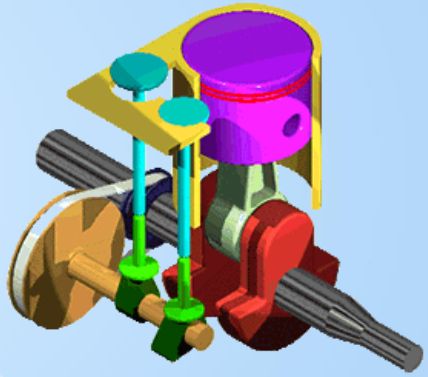
- Pyhsical Architecture

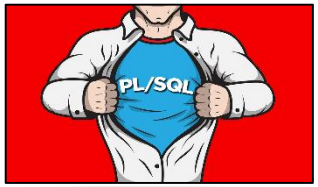




What is PL/SQL?

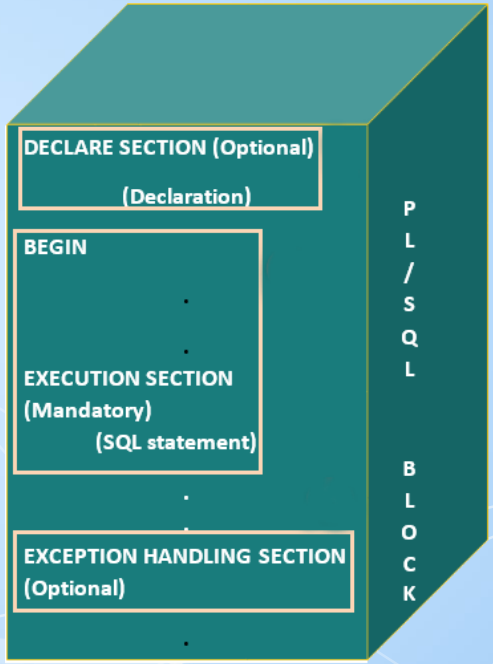
- **PL/SQL Logical Architecture**
  - Cooperates with SQL Engine
  - Enables Subprograms
  - Dynamic Queries
  - Case Insensitivity
  - Optimizer
  - Enables Object-Orinted Programming
  - Web Development
  - To sum up!..

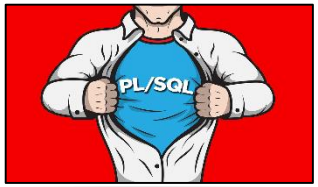




Let's Start Coding!

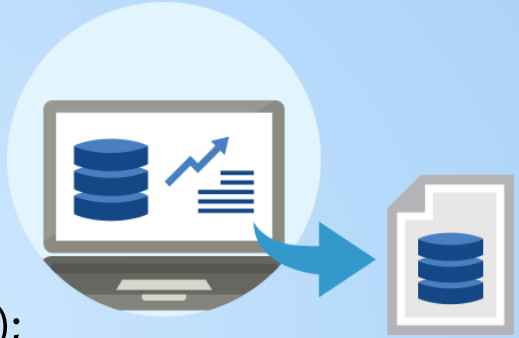
- **Blocks**
  - What are blocks?
    - DECLARE (Optional)
    - BEGIN (Mandatory)
    - EXCEPTION (Optional)
    - END; (Mandatory)
  - Three types of blocks
    - Anonymous Blocks
    - Procedures
    - Functions



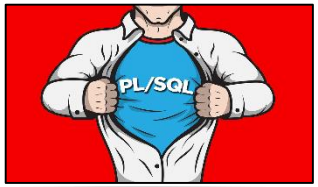


Let's Start Coding!

- **PL/SQL outputs**
  - Not an output language.
  - SET SERVEROUTPUT ON
  - DBMS_OUTPUT
  - DBMS_OUTPUT.PUT_LINE(*output_text*);



- Nested Blocks



Variables

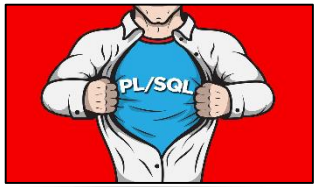
- What are Variables & Why do we Need Them?
- PL/SQL Variable Types

SCALAR

REFERENCE

LARGE OBJECTS

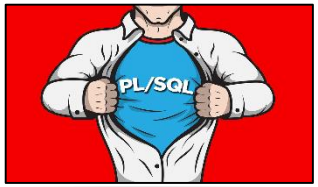
COMPOSITE



Variables

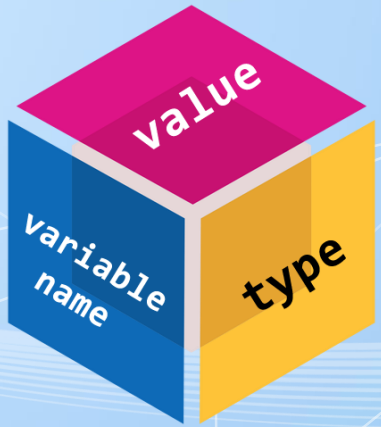
- Scalar Data Types

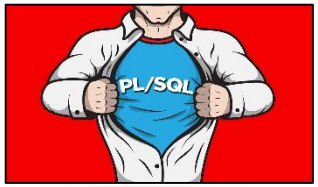
1. CHAR (max_length)
2. VARCHAR2(max_length)
3. NUMBER[precision,sale]
4. BINARY_INTEGER = PLSINTEGER
5. BINARY_FLOAT
6. BINARY_DOUBLE
7. BOOLEAN
8. DATE
9. TIMESTAMP (precision)
10. TIMESTAMP(p) WITH TIME ZONE
11. TIMESTAMP(p) WITH LOCAL TIME ZONE
12. INTERVAL(p) YEAR TO MONTH
13. INVERVAL(p) DAY TO SECOND(p)



Variables

- **Variable Naming Rules**
  - Must start with a letter.
  - Can contain some special characters (like _ # $)
  - Can be maximum 30 characters.
  - Can not have Oracle's reserved words (select , varchar2 , etc)
- **Naming Conventions**
  - Why Naming Conventions?
    - VARIABLE – v_variable_name -- v_max_salary
    - CURSOR – cur_cursor_name -- cur_employees
    - EXCEPTION – e_exception_name – e_invalid_salary
    - PROCEDURE – p_procedure_name – p_calculate_salary
    - BIND VARIABLE – b_bind_name – b_emp_id

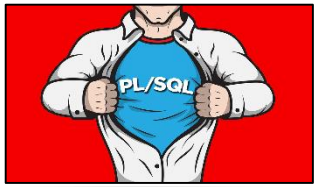




Variables

- **Declaring & Initializing & Using Variables**
  - General Usage :

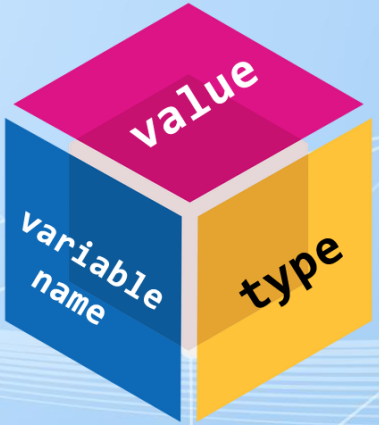
```
Name [CONSTANT] datatype [NOT NULL] [:= DEFAULT value|expression];
```

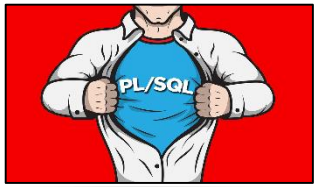


Variables

- **USING %TYPE ATTRIBUTE**
  - What is %TYPE Attribute?
  - Why & Where to use it?
  - How do we Use it?

```
name table.column_name%TYPE | another_variable%TYPE ;
```



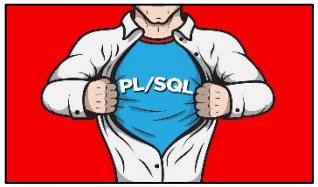


Variables

- **DELIMITERS & COMMENTING YOUR CODE.**
  - Delimiters are symbols with meaning.

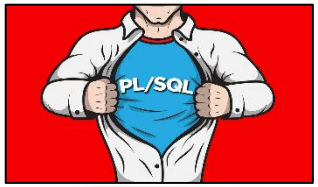
Symbol	Meaning
+	Addition
-	Substraction | Negation
*	Multiplication
/	Division
=	Equality
@	Remote Access
;	Statement

Symbol	Meaning
<>	Inequality
!=	Inequality
||	Concatenation
:=	Assignment
--	Single Line Comment
/* */	Multi-Line Comment



Variables

- **USING BIND VARIABLES.**
  - Why Bind Variables?
  - Increases Performance
  - Created in the host environment
  - Its scope is whole worksheet
  - Let's see by coding!..



Control Structures

IF STATEMENT

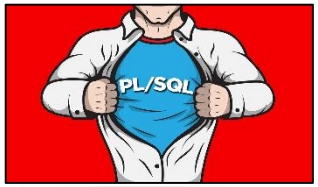
- What does Control Structures Mean?
- What is IF Statement?

```
IF condition THEN statements;
[ELSIF condition THEN statements;]
[ELSIF condition THEN statements;]
[ELSE  statements;]
END IF;
```

AND LOGIC TABLE			
AND	TRUE	FALSE	NULL
TRUE	true	false	null
FALSE	false	false	false
NULL	null	false	null

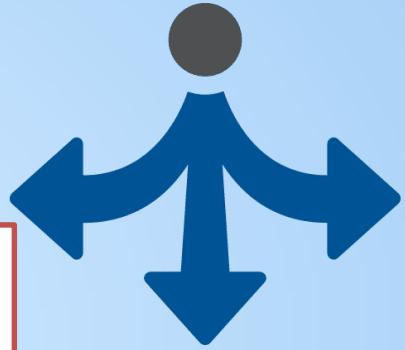
OR LOGIC TABLE			
OR	TRUE	FALSE	NULL
TRUE	true	true	true
FALSE	true	false	null
NULL	true	null	null

NOT LOGIC TABLE	
NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

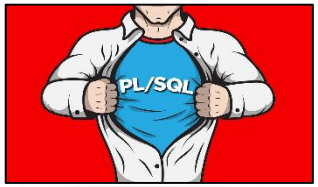


Control Structures

- **CASE EXPRESSIONS**
  - What is a Case Expression & Why to Use?
  - Case Expressions



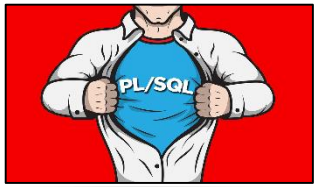
```
CASE [expression||condition]
       WHEN condition1 THEN  result1
       WHEN condition2 THEN  result2
                 ::::::::::
     [WHEN conditionN THEN  resultN]
     [ELSE                  result]
END;
```



Control Structures

- **LOOPS**
  - BASIC LOOPS

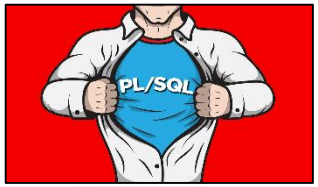
```
LOOP
        your_code;
                ::::::::::
        EXIT [WHEN condition];
END LOOP;
```



Control Structures

- **LOOPS**
  - WHILE LOOPS

```
WHILE condition LOOP
        your_code;
                  ::::::::::
END LOOP;
```

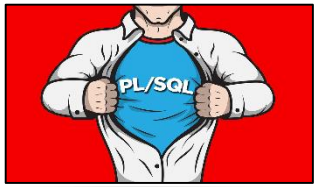


Control Structures

- **LOOPS**
  - FOR LOOPS

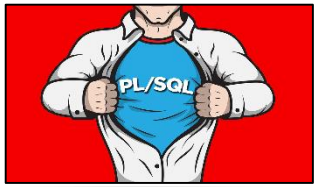
```
FOR counter IN [REVERSE]
        lower_bound..upper_bound LOOP
        your_code;
                ::::::::::
END LOOP;
```

- We cannot reach the counter outside of the loop.
- We cannot assign any value to the counter.
- Our bounds cannot be null.



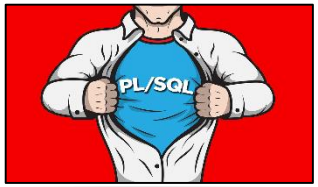
Control Structures

- **LOOPS – Recap**
  - **Which one to use?**
    - Use Basic Loops when..
    - Use While Loops when..
    - Use For Loops when..
  - To Sum Up !..



Control Structures

- **Nesting and Labeling the Loops**
  - We can nest loops..
  - We can use labels for loops.
  - Exting the inner loop will not exit the outer.
  - We can use labels to exit the outer loop.

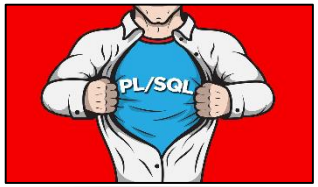


Control Structures

- **CONTINUE STATEMENT**
  - Why to use Continue Statement?
  - What are the benefits?
  - How is it used?

```
CONTINUE [label_name] [WHEN condition];
```

  - Let's make some examples..

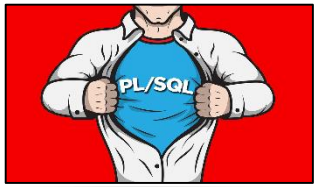


Control Structures

- **GOTO STATEMENT**
  - What is GOTO Statement?
  - Why to use GOTO Statement?
  - How is it used?

```
GOTO label_name;
```

  - There are some restrictions.
    - Into a control structure
    - Into an inner block from the outer
    - Out of a subprogram
    - In or Out of an Exception handler
  - Let's make some examples..



Using SQL in PL\SQL

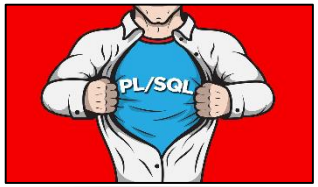
- **OPERATING WITH SELECTED QUERIES**
  - Why to use SQL in PL/SQL?
  - There are some issues???
    - ➢ You cannot use DDL commands directly..
    - ➢ A block does not mean a transaction..
  - How to use SQL in PL/SQL?

```
SELECT columns|expressions
INTO variables|records
FROM table|tables
[WHERE condition];
```

  - Naming Conventions & Ambiguities

Using SQL in PL\SQL

- **DML OPERATIONS IN PL/SQL**
  - We can use DML commands in PL/SQL code.
  - We can use PL/SQL Variables in a DML command.
    - *INSERT*
    - *UPDATE*
    - *DELETE*
    - *MERGE*

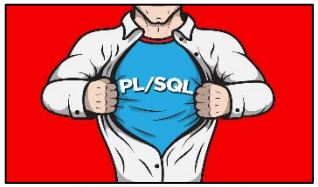


Using SQL in PL\SQL

- **USING SEQUENCES IN PL/SQL**
  - Why to use Sequences?
  - Where to use Sequences?
  - How to use Sequences?

```
SELECT sequence_name.nextval|currval INTO variable|column
FROM table_name|dual;
```

```
Variable|column := sequence_name.nextval|currval;
```



Composite Data Types

- **SIMPLE DATA TYPES vs COMPOSITE DATA TYPES**

```
declare
 v_name varchar2(50);
 v_salary employees.salary%type;
 v_employee_id employees.employee_id%type := 130;
begin
  select first_name ||' '|| last_name, salary into v_name, v_salary from employees
where employee_id = v_employee_id;
  dbms_output.put_line('The salary of '|| v_name || ' is : '|| v_salary );
end;
```

Composite Data Types

PGA

Records

rec_emp

id	first_name	last_name	salary	hire_date
101	Neena	Kocchar	17000	21-SEP-05

Nested Tables

c_emp_names

index	name
1	Luis
2	TJ
3	Nandita
4	William

Varrays

c_emp_names(4)

index	name
1	Luis
2	TJ
3	Nandita
4	William

Ass. Arrays

c_emp_names

index	name
113	Luis
132	TJ
184	Nandita
206	William

In Memory Tables

t_emps

index	first_name	last_name	salary	hire_date
101	Neena	Kocchar	17000	21-SEP-05
102	Lex	De Haan	17000	13-JAN-01
145	John	Russell	14000	01-OCT-04
104	Bruce	Ernst	6000	21-MAY-07

t_emps

index	last_name	salary	hire_date	department_id
Neena	Kocchar	17000	21-SEP-05	90
Lex	De Haan	17000	13-JAN-01	90
John	Russell	14000	01-OCT-04	80
Bruce	Ernst	6000	21-MAY-07	60

Composite Data Types

- **PL/SQL RECORDS**
  - What is a record?

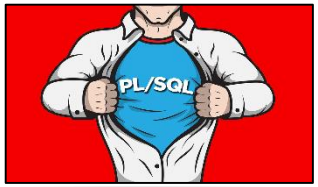
A Record

	Employee_id	first_name	last_name	salary	hire_date
rec_emp	101	Neena	Kocchar	17000	21-SEP-05

```
r_name TABLE_NAME%rowtype;
```

```
type type_name is record ( variable_name variable_type,
                           [variable_name variable_type,]
                           [..........]);
```

  - In Variable Types area, we can use:
    - ✓ All Valid PL/SQL Types, %TYPE, %ROWTYPE, NOT NULL and DEFAULT keywords.



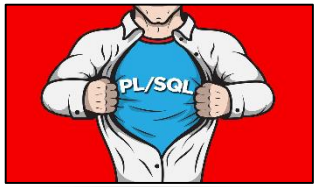
Composite Data Types

- **EASY DML WITH RECORDS**
  - How is the known way?

```
insert into departments values (280,'Temp Dept',null,1700);
```

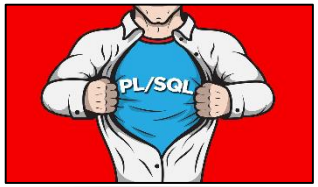
```
insert into departments (department_id, department_name) values (290,'Temp Dept2');
```

  - Why is it useful?
  - Let's make examples..



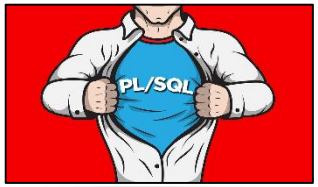
Composite Data Types

- **WHAT ARE COLLECTIONS?**
  - The second type of the composite data types.
  - Why do we need a collection?
  - Collections have a list of the same type.
  - There are 3 types of collections..
    - Nested Tables
    - VARRAYs
    - Associative Arrays
  - TABLE Operator



Composite Data Types

- **VARRAYS**
  - Identical of Arrays in other languages.
  - Upper limit is certain.
  - About Varrays :
    - Varrays are bounded
    - It's index start from 1
    - Varrys are one dimentional arrays
    - Varrays are null by default
  - Let's make examples..

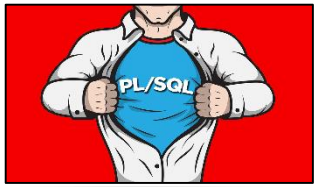


Composite Data Types

- **NESTED TABLES**
  - Very similar with the varrays.
  - Compare to varrays..
    - ✅ Key-Value Pairs
    - ✅ 2 GB at most
    - ❌ We can delete any values
    - ❌ Not stored consecutively
    - ❌ Nested tables are unbounded

```
type type_name as table of
     value_data_type [NOT NULL]);
```

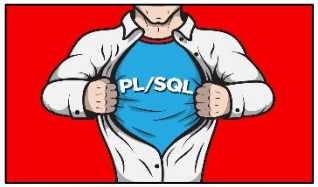
```
create or replace type type_name as table of
     value_data_type [NOT NULL]);
```



Composite Data Types

- **ASSOCIATIVE ARRAYS**
  - Known as Index By Tables
  - Compare to other collections..
    - ✗ The key can have a string
    - ✗ Keys does not need to be sequential
    - ✓ Can have scalar & record types
    - ✗ Do not initialize associative arrays
    - ✗ Associative arrays are indexed

```
type type_name as table of value_data_type [NOT NULL]
INDEX BY {PLS_INTEGER | BINARY_INTEGER |VARCHAR2(size)};
```

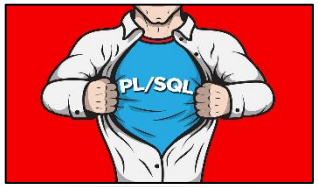



Composite Data Types

- **STORING COLLECTIONS IN TABLES**
  - We can store varrays and nested tables in database tables.
  - They are stored with different methods
  - Store and use easier-faster
  - Our types must be schema level

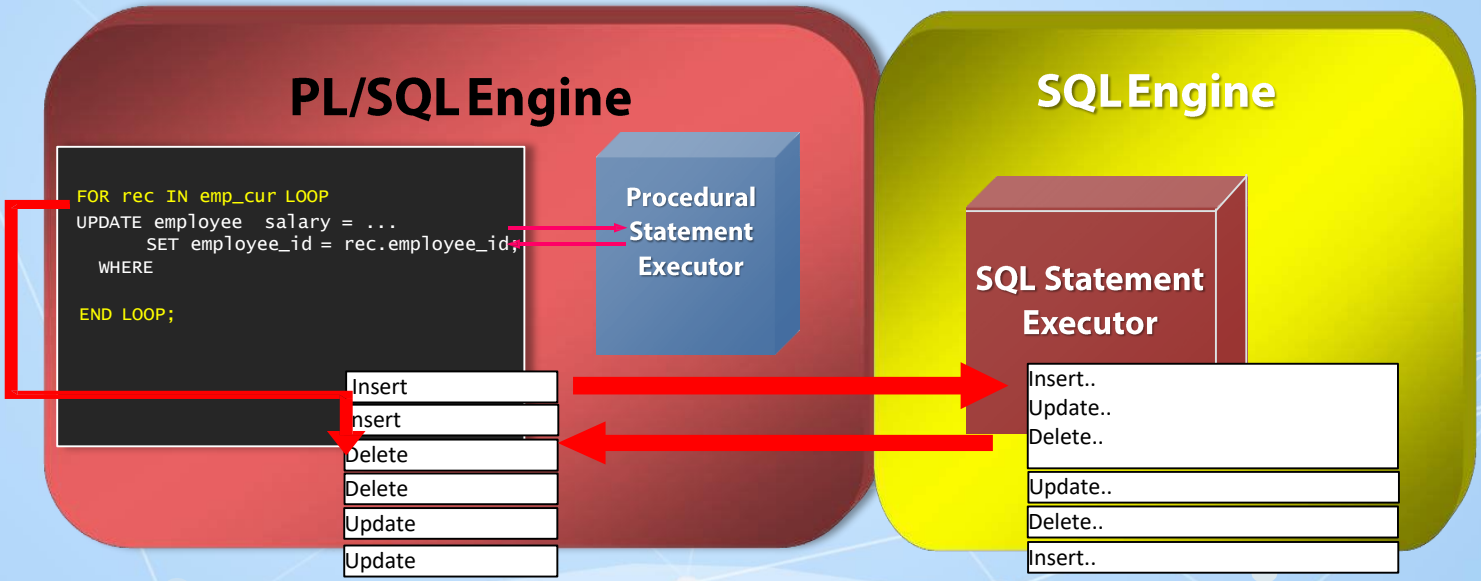
Employees table

EMP_ID	FIRST_NAME	LAST_NAME	PHONE_NUMBER	
101	Bob	Green	Home	'111.111.1111'
			Work	+1-541-754-3010
102	David	Brown	Home	'111.111.1111'
			Work	+1-541-754-3010
			Mobile	+1-541-754-3510
			Fax	+1 (408) 999 8888



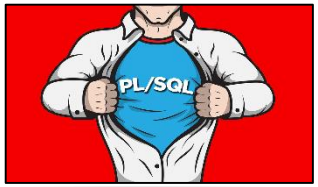
Cursors

- **WHAT ARE CURSORS & CURSOR TYPES?**



Cursors

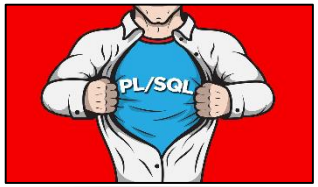
- **WHAT ARE CURSORS & CURSOR TYPES?**
  - Cursors are pointers to the data
  - There are two types of cursors.
    - Implicit Cursor
    - Explicit Cursor
  - We can control the cursors
  - Explicit cursors are created by the programmers
  - Collections vs Cursors
  - You cannot go back in cursors



Cursors

- **USING EXPLICIT CURSORS**
  - The Cursor Usage :
    - Declare
    - Open
    - Fetch
    - Check
    - Close

```
 declare
   cursor cursor_name is select_statement;
begin
   open cursor_name;
   fetch cursor_name into variables,records etc;
   close cursor_name;
end;
```



Cursors

- **CURSORS WITH PARAMETERS**
  - Cursors can be used for many times
  - We can use cursors with parameters

```
 declare
   cursor cursor_name(parameter_name datatype,…)
   is select_statement;
begin
   open cursor_name(parameter_values);
   fetch cursor_name into variables,records etc;
   close cursor_name;
end;
```

Cursors

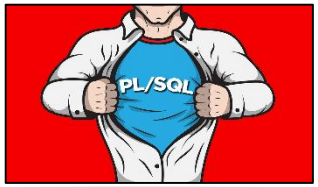
CURSOR ATTRIBUTES

- There are four cursor attributes
  - %FOUND – *Returns true if the fetch returned a row.*
  - %NOTFOUND – *Opposite of %found*
  - %ISOPEN – *Returns true if the cursor is open*
  - %ROWCOUNT – *Returns the number of fetched rows*

Cursors

- **FOR UPDATE CLAUSE**
  - When you update a row, it is locked to the others
  - **for update** clause locks the selected rows
  - **nowait** option will terminate execution if there is a lock
  - Default option is **wait**
  - **for update of** clause locks only the selected tables

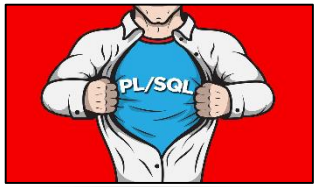
```
cursor cursor_name(parameter_name datatype,…)
   is select_statement
   for update [of column(s)] [nowait | wait n]
```



Cursors

- **REFERENCE CURSORS (REF CURSORS)**
  - Why we need REF CURSORS?
  - Cursors are pointers.
  - You can use a cursor for multiple queries
  - We cannot :
    - ⊗ Assing null values
    - ⊗ Use in table-view create codes
    - ⊗ Store in collections
    - ⊗ Compare
  - How do we use ref cursors?
  - There two types of reference cursors
    - ➢ Strong (restrictive) cursors
    - ➢ Weak (nonrestrictive) cursors.

```
type cursor_type_name is ref cursor [return return_type]
open cursor_variable_name for query;
```



Exceptions

- **What are the Exceptions?**
  - What does compilers do in basic?
  - What are exceptions?
  - How to handle the exceptions?
  - A block has three sections.
    - Declaration section
    - Begin-End Section
    - Exception Section
  - We can handle exceptions in three ways.
    1. Trap
    2. Propagate
    3. Trap & Propagate
  - How to know the exceptions?

Exceptions

- **Handling Exceptions**

```
declare
    ..............................
begin
    {An exception occurs here}
exception
    when exception_name then
        ..............................
    when others then
        ..............................
end;
```

- There are three types of exceptions :
  - Predefined Oracle Server Errors
  - Nonpredefined Oracle Server Errors
  - User-defined Errors

Exceptions

- **Non-predefined Exceptions**
  - Unnamed Exceptions
  - We cannot trap with the error codes
  - We declare exceptions with the error codes

```
exception_name EXCEPTION;
```

```
PRAGMA EXCEPTION_INIT(exception_name, error_code)
```

  - What is PRAGMA ?
  - What does exception_init do?

Exceptions

- **User-predefined Exceptions & Raising Exceptions**
  - You need to handle some exceptions about your business.
  - These exceptions are not an error for the database.
  - Define a user-defined exception and raise it.

```
exception_name EXCEPTION;
```

```
RAISE exception_name;
```

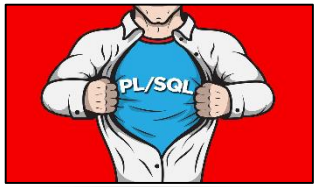
  - You can raise any type of exceptions with RAISE statement.

Exceptions

- **RAISE_APPLICATION_ERROR Procedure**
  - Sometimes you want to raise an exception out of the block
  - raise_application_error raises the error to the caller

```
raise_application_error(error_number,error_message[,TRUE|FALSE]);
```

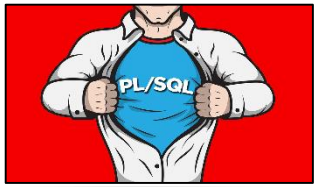
  - What is the error stack?
  - Will stop execution of the application
  - Error number must be between -20000 and -20999
  - Message will be up to 2048 bytes long



Functions & Procedures

What are Functions & Stored Procedures?

- Reusability is important in programming
- What are the differences than anonymous blocks?
  - Stored in the database with names.
  - Compiled only once
  - Can be called by another block or application
  - Can return values
  - Can take parameters

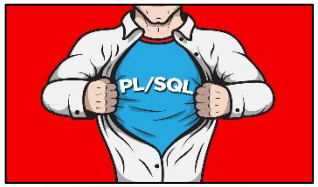


Functions & Procedures

Creating & Using Stored Procedures

- If you need to do the same thing again and again, you use procedures.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter_name [IN | OUT | IN OUT] type [, ...])] {IS | AS}
    ..........................  --declarative section
begin
    ..........................
exception
    ..........................
end;
```



Functions & Procedures

Using IN & OUT Parameters

- Why we pass parameters?
- We can create our procedures and functions with parameters

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter_name [IN | OUT | IN OUT] type [, ...])] {IS | AS}
    ..... --declarative section
begin
    .....
exception
    .....
end;
```

Functions & Procedures

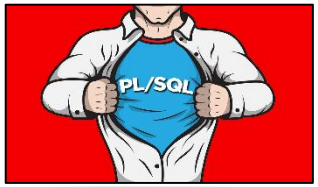
NAMED - MIXED NOTATIONS & DEFAULT OPTION

- We can run the procedures with or without functions
- We do that with the DEFAULT option

```
CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter_name [IN | OUT | IN OUT] type DEFAULT value|expression
  [, ...])] {IS | AS}
```

- Named notation allows us to pass parameter independent from the position.

```
EXECUTE procedure_name(parameter_name => value|expression);
```

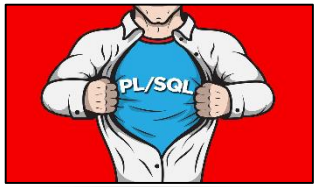


Functions & Procedures

CREATING AND USING FUNCTIONS

- Functions are pretty similar with the procedures
- Functions can get IN and OUT parameters
- Functions must RETURN a value
- Functions are very similar with procedures on creation, except its usage
- Functions can be used within a select statement
- You can assign a function to a variable.

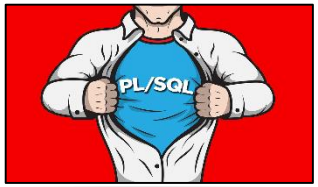
```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter_name [IN | OUT | IN OUT] type DEFAULT value|expression
  [, ...])] RETURN return_data_type {IS | AS}
```



Functions & Procedures

CREATING AND USING FUNCTIONS

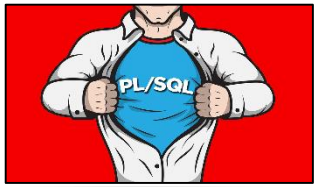
- **Differences and Similarities of Functions & Procedures**
  - Procedures are executed within a begin-end block or with execute command.
  - Functions are used within an SQL Query or assigned to some variable.
  - We can pass IN & OUT parameters to both
  - Procedures does not return a value, but functions return



Functions & Procedures

CREATING AND USING FUNCTIONS

- **The restrictions of using functions in SQL expressions**
  - Must be compiled and stored in the database
  - Your functions should not have an OUT parameter.
  - Must return a valid type of the SQL Data Types
  - Cannot be used in table creation codes
  - Cannot call a function that contains a DML Statement
  - Cannot include COMMIT, ROLLBACK or DDL Statements
  - If the function has a DML operation of the specified table
  - You need to have the related privilege



Functions & Procedures

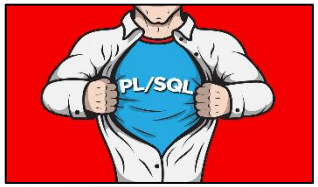
LOCAL SUBPROGRAMS

- We create procedures and functions to reduce the code crowd
- They are called as stored procedures & stored functions
- Sometimes it is unnecessary to store the subprograms
- We can create subprograms inside of an anonymous blocks or in another subprogram.
- The benefits of local subprograms :
  - ✓ Reduce code repetition
  - ✓ Improve code readability
  - ✓ Need no more privilege
- The cons of the local variables :
  - ✓ They are accessible only in the blocks they are defined

Functions & Procedures

OVERLOADING THE SUBPROGRAMS

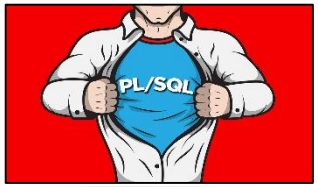
- Overloading means creating more than one subprogram with the same name.
- Overload the subprograms with same name but with different parameters.
- Overloading is pretty useful when using subprograms with packages
- Overloading the Subprograms :
  - ✓ Enables creating two or more subprograms with the same name.
  - ✓ So we can build more flexible subprograms.
  - ✓ We can overload local subprograms and package subprograms. But not standalone subprograms!..
  - ✓ Parameters must be different in data types or orders or numbers.
  - ✓ If parameters are in the same family or subtype, it won't work.
  - ✓ Differentiating only the return type will not be accepted



Functions & Procedures

USING HOST VARIABLES

- Overloading means creating more than one subprogram with the same name.
- Overload the subprograms with same name but with different parameters.
- Overloading is pretty useful when using subprograms with packages
- Overloading the Subprograms :
  - ✓ Enables creating two or more subprograms with the same name.
  - ✓ So we can build more flexible subprograms.
  - ✓ We can overload local subprograms and package subprograms. But not standalone subprograms!..
  - ✓ Parameters must be different in data types or orders or numbers.
  - ✓ If parameters are in the same family or subtype, it won't work.
  - ✓ Differentiating only the return type will not be accepted



Functions & Procedures

HANDLING THE EXCEPTIONS IN SUBPROGRAMS

- Why to handle the exceptions in subprograms?
- You can have an exception section in all begin-end blocks.
- When an exception is raised, the control is passed to the exception section.
- What happens if we don't handle the exceptions?
- We may have exceptions in functions & procedures, too.
- In subprograms, the unhandled exception is raised to the caller.
- The control must be on you.
- You should handle all the possible exceptions.

Functions & Procedures

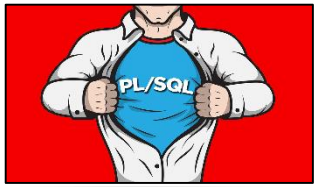
REGULAR & PIPELINED TABLE FUNCTIONS

- What are table functions?
- Table functions return a table of varray or nested tables
- Regular table functions returns after completing the whole data
- Pipelined functions return each row one by one
- Which one to choose?

Packages

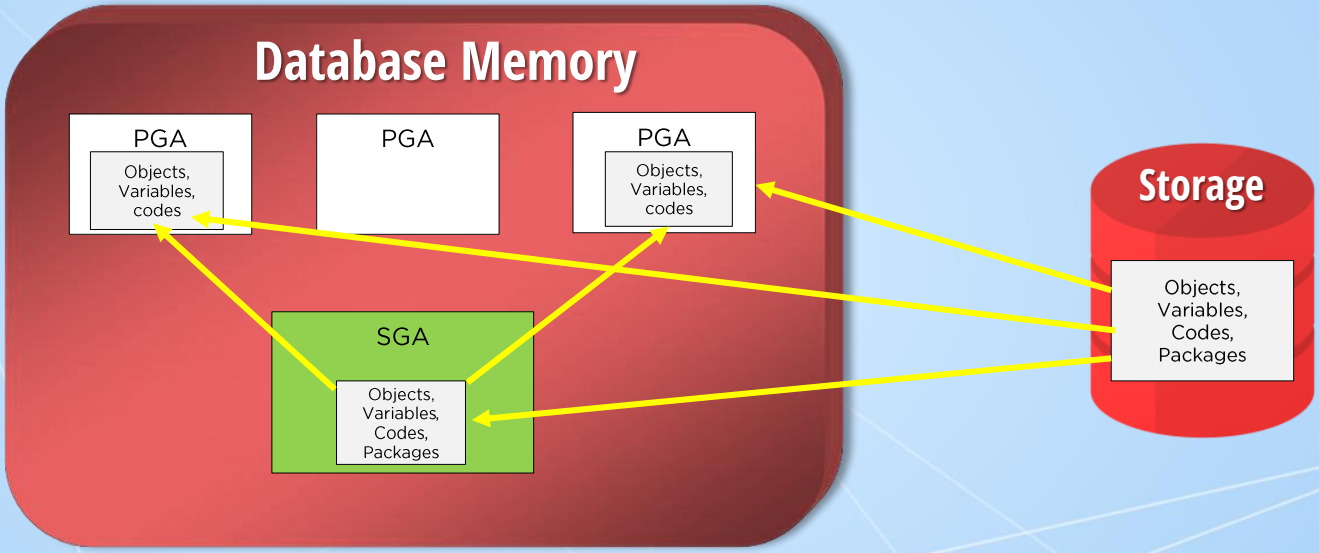
WHAT IS A PACKAGE?

- Most of the times, our objects work together.
- There will be an object crowd in a real work in time
- Packages groups subprograms, types, variables etc in one container.
- There is one more reason for using packages



Packages

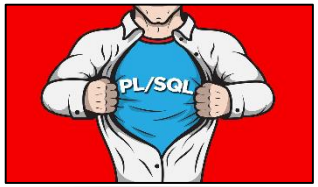
DIFFERENCE OF A PACKAGE USAGE IN MEMORY



Packages

ADVANTAGES OF USING PACKAGES

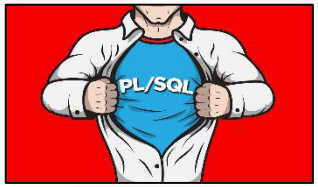
- Modularity
- Easy Maintenance
- Encapsulation & Security
- Performance
- Functionality
- Overloading



Packages

CREATING & USING PACKAGES

- To use a package, we have two main reasons :
  - ✓ Logically grouping the objects
  - ✓ The performance gain
- With logically grouping, we can decrease code complexity and crowd.
- A package consists of two parts.
  - ➢ Package Specification (spec)
  - ➢ Package Body (body)
- Body and Spec must have the same name



Packages

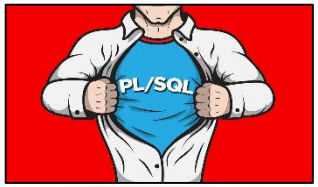
VISIBILITY OF PACKAGE OBJECTS

- An object is visible to the others only if it is declared in the package spec.
- Where can we declare the variables in packages?
  - ✓ Inside of the package spec
  - ✓ Inside of the package body

Packages

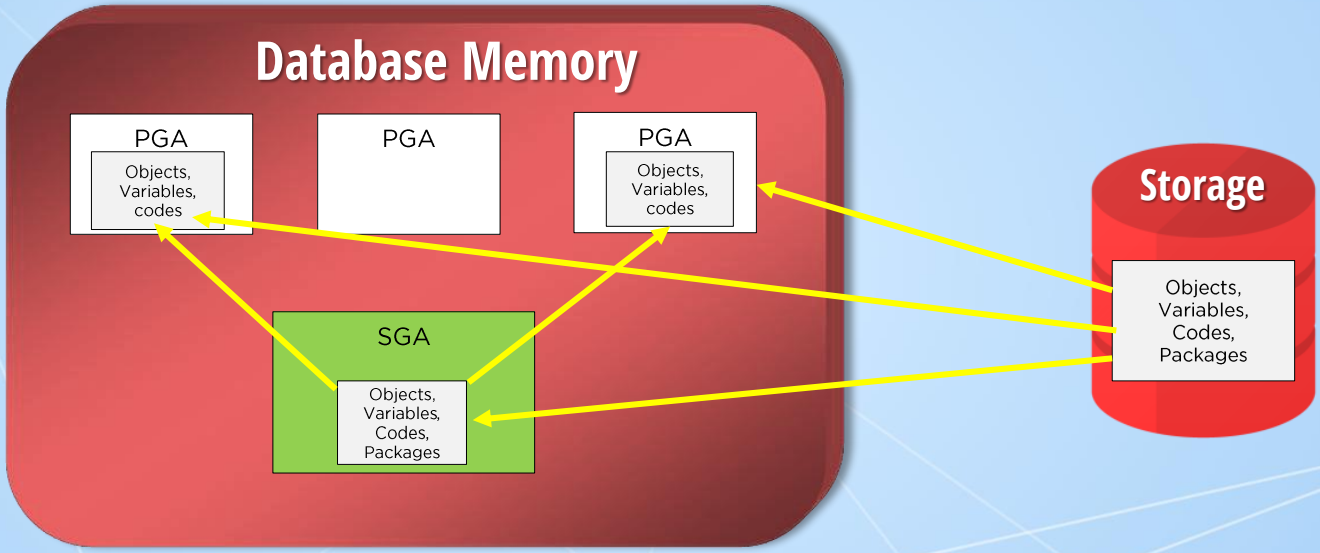
PACKAGE INITIALIZATION

- A package is loaded into the memory on the first call.
- Uninitialized variables are null by default.
- Oracle let's us to initialize the variables with one more way.
- The initialized variables will be overridden with that way.
- We can also do some business here.
- We do that as the last begin-end block.



Packages

PERSISTENT STATE OF PACKAGES



Packages

PERSISTENT STATE OF PACKAGES

- A package is loaded into the memory at the first reference.
- Variables and objects are stored in your PGA.
- These variables are persistent for your session.
- These variables are unique for your session.
- Subprograms of the packages are stored in the SGA.
- We can change the persistent state of variables.
- PRAGMA SERIALLY_REUSABLE
- Serially reusable packages cannot be accessed from :
  - Triggers
  - Subprograms called from SQL Statements.