PL/SQL – PART 1

CIS-673, LECTURE#22

BY RAJ PATIL

PL SQL BASICS

- **PLSQL Block**

  - Declare

  --declare variables, procedures, etc.

  - begin

  --execution block begins

  - end;

  /

3

PL SQL BASICS

- Local vs. global variables
- Labels and GOTO statement
- If elif, else statement
- Case statement
- Loop Types: while, for, loop
- Loop : exit, exit when, continue, continue when

4 PROCEDURES, FUNCTIONS, PARAMETERS

- **Procedures:** do not return value
- **Functions:** return values
- **Parameters:**
  - IN: used to pass values; are constants (read only) and cannot be updated
  - OUT: used to return values; can be updated.
  - IN OUT: used to pass and return values;
    - An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller.

- **Passing value by**
  - Position based technique (default)
  - named notations (keyword arguments)

5 COLLECTIONS

- **Major Collection Types:**
  - Varrays
  - Nested Tables
  - Associative Arrays

- Unlike a database table, these collections are stored in main-memory(RAM), and therefore does not need disk space or I/O access.

VARRAYS

- Varrays (Variable-Size Arrays): fixed length (cannot extend/add, delete, or trim elements)
- A varray (variable-size array) is an array whose number of elements can vary from zero (empty) to the declared maximum size.
- To access an element of a varray variable, use the syntax variable_name(index).
- The lower bound of index is one (1); the upper bound is the current number of elements.
- The upper bound changes as you add or delete elements, but it cannot exceed the maximum size.
- A varray is appropriate when:
  - the maximum number of elements are known.
  - access the elements sequentially.

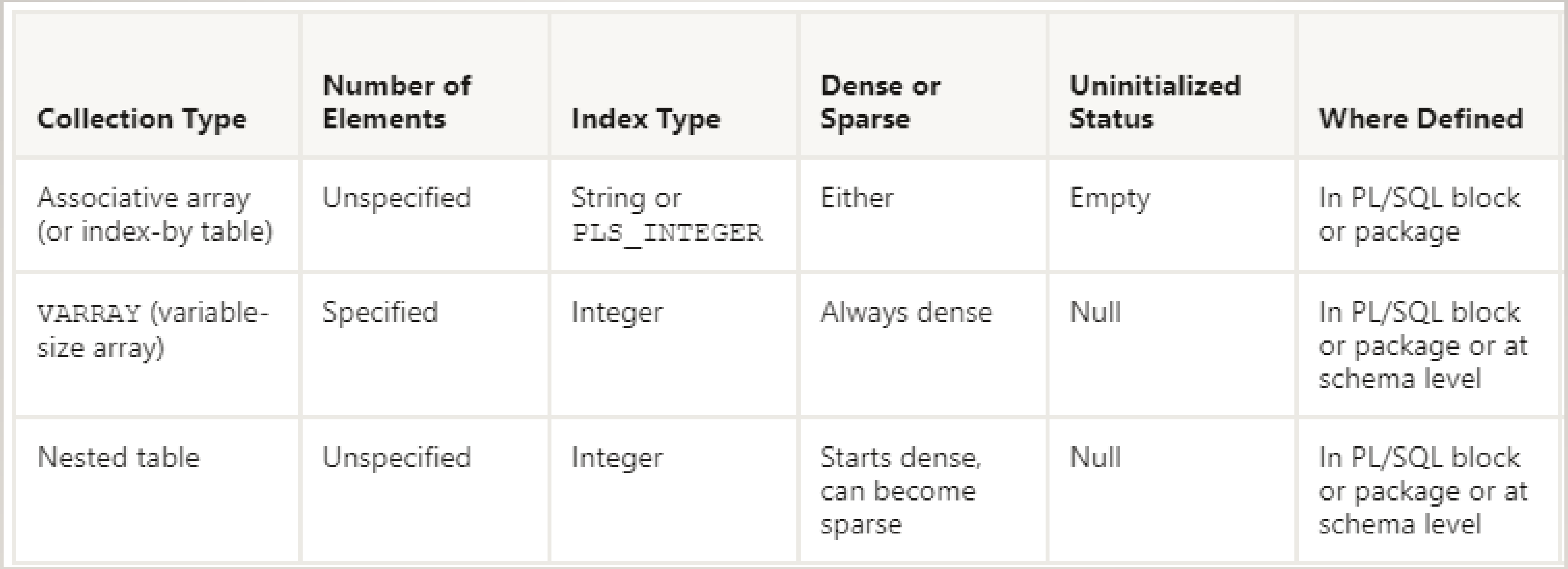
NESTED TABLES

- Nested Tables: dynamic size (can extend/add, delete, trim elements)
- In the database, a nested table is a column type that stores an unspecified number of rows in no particular order.
- When you retrieve a value from a PL/SQL nested table variable, PL/SQL gives the rows consecutive indexes, starting at 1.
- Using these indexes, you can access the individual rows of the nested table variable.
- The amount of memory that a nested table variable occupies can increase or decrease dynamically, as you add or delete elements.

ASSOCIATIVE ARRAYS

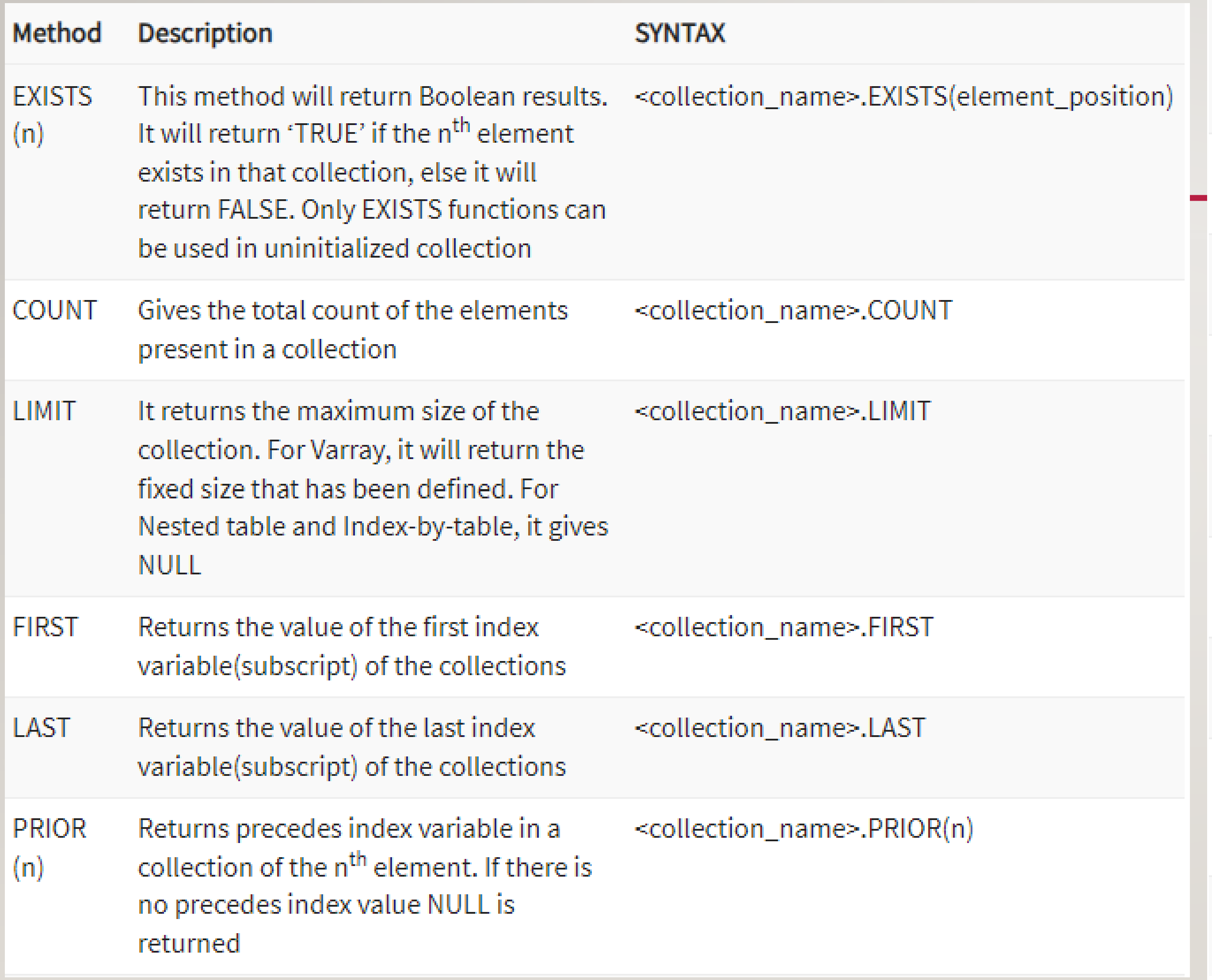
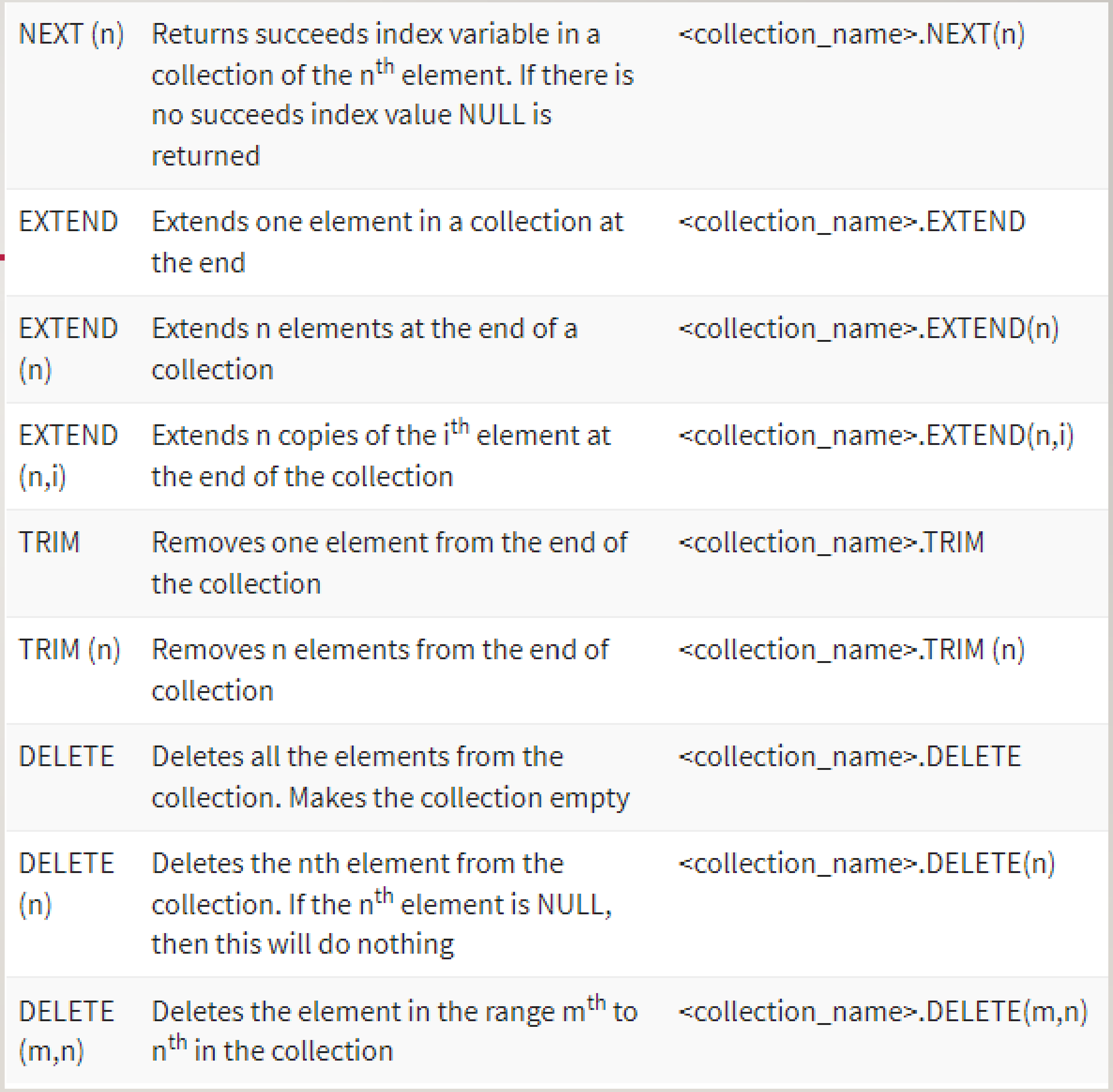
- Associative arrays: It is a set of (key-value) pairs,   index-by table
- Each key is a unique index, used to locate the associated value with the syntax variable_name(key)
- - The data type of index can be either a string type or PLS_INTEGER.
- Indexes are stored in sort order, not creation order.
- Can hold an unspecified number of elements, which you can access using keys, without knowing their positions.
- An associative array is appropriate for: A relatively small lookup table

COLLECTIONS COMPARISON



Collection Type	Number of Elements	Index Type	Dense or Sparse	Uninitialized Status	Where Defined
Associative array (or index-by table)	Unspecified	String or `PLS_INTEGER`	Either	Empty	In PL/SQL block or package
`VARRAY` (variable-size array)	Specified	Integer	Always dense	Null	In PL/SQL block or package or at schema level
Nested table	Unspecified	Integer	Starts dense, can become sparse	Null	In PL/SQL block or package or at schema level

COLLECTION METHODS

Method	Description	SYNTAX
EXISTS (n)	This method will return Boolean results. It will return 'TRUE' if the nth element exists in that collection, else it will return FALSE. Only EXISTS functions can be used in uninitialized collection	<collection_name>.EXISTS(element_position)
COUNT	Gives the total count of the elements present in a collection	<collection_name>.COUNT
LIMIT	It returns the maximum size of the collection. For Varray, it will return the fixed size that has been defined. For Nested table and Index-by-table, it gives NULL	<collection_name>.LIMIT
FIRST	Returns the value of the first index variable(subscript) of the collections	<collection_name>.FIRST
LAST	Returns the value of the last index variable(subscript) of the collections	<collection_name>.LAST
PRIOR (n)	Returns precedes index variable in a collection of the nth element. If there is no precedes index value NULL is returned	<collection_name>.PRIOR(n)
NEXT (n)	Returns succeeds index variable in a collection of the nth element. If there is no succeeds index value NULL is returned	<collection_name>.NEXT(n)
EXTEND	Extends one element in a collection at the end	<collection_name>.EXTEND
EXTEND (n)	Extends n elements at the end of a collection	<collection_name>.EXTEND(n)
EXTEND (n,i)	Extends n copies of the ith element at the end of the collection	<collection_name>.EXTEND(n,i)
TRIM	Removes one element from the end of the collection	<collection_name>.TRIM
TRIM (n)	Removes n elements from the end of collection	<collection_name>.TRIM (n)
DELETE	Deletes all the elements from the collection. Makes the collection empty	<collection_name>.DELETE
DELETE (n)	Deletes the nth element from the collection. If the nth element is NULL, then this will do nothing	<collection_name>.DELETE(n)
DELETE (m,n)	Deletes the element in the range mth to nth in the collection	<collection_name>.DELETE(m,n)

CURSOR

- A cursor holds the rows (one or more) returned by a SQL statement.
- You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time.
- Two types: implicit and explicit cursor.
- Steps involved in using an **explicit cursor**: declare, open, fetch, and close
    - %FOUND: Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or A SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
    - %ROWCOUNT: Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.
- Parameters can be passed explicitly to a Cursor

IMPLICIT CURSORS

- Implicit cursors are automatically created whenever an SQL statement is executed and when there is no explicit cursor created for the statement.
- Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement
- Cursor attributes:
  - %found,
  - %notfound,
  - %rowcount,
  - %isopen

%TYPE, %ROWTYPE, RECORD TYPES

- %Rowtype variable
  - enables to capture result/data returned by select * statement.
  - holds all the columns in a given table.
- %Record variable
  - enables to capture data of specified/selective columns returned by select A1, A2, …, An statement
- %Type
  - variable enables to capture singular data associated with a specific column of a table.

- All the above types are helpful when the data-types of table-columns are unknown and/or need to be pulled dynamically.

DYNAMIC SQL

- Dynamic SQL is a programming methodology for generating and running SQL statements at run time. It is useful when
  - writing programs that must run database definition language (DDL) statements, or
  - when you do not know at compilation time the full text of a SQL statement or the number or data types of its input and output variables.
- Use the EXECUTE IMMEDIATE statement to execute a Dynamic-SQL statement.
- Oracle EXECUTE IMMEDIATE can also build up statements to execute operations in which you do not know the table names, column names, or other properties.
- Place holders are defined using colon (:), and there associated variables are listed with the keyword 'USING'.
- Example: EXECUTE IMMEDIATE 'INSERT INTO ' || person_table || ' VALUES (:1, :2)' USING person_name, person_age;

REF CURSORS

- A "normal" plsql cursor is static in definition. However, a "Ref-cursor" may be dynamically opened or opened based on logic.
- A ref cursor is defined at runtime and can be opened dynamically but a regular cursor is static and defined at compile time.
- A regular plsql "cursor" cannot be returned to a client, whereas ref cursor can be returned to a client.
- a REF CURSOR is a pointer or a handle to a result set on the database, and refers to a memory address on the database.
- A REF CURSOR is not updatable. The result set represented by the REF CURSOR is read-only.

PACKAGES

- **package specification:** defines what is contained in the package; it is analogous to a header file in a language such as C++.

- **package body:** contains the code for the procedures and functions defined in the specification, and the code for private procedures and functions that are not declared in the specification. This private code is only visible within the package body.

- The specification defines all public items. The specification is the published interface to a package.