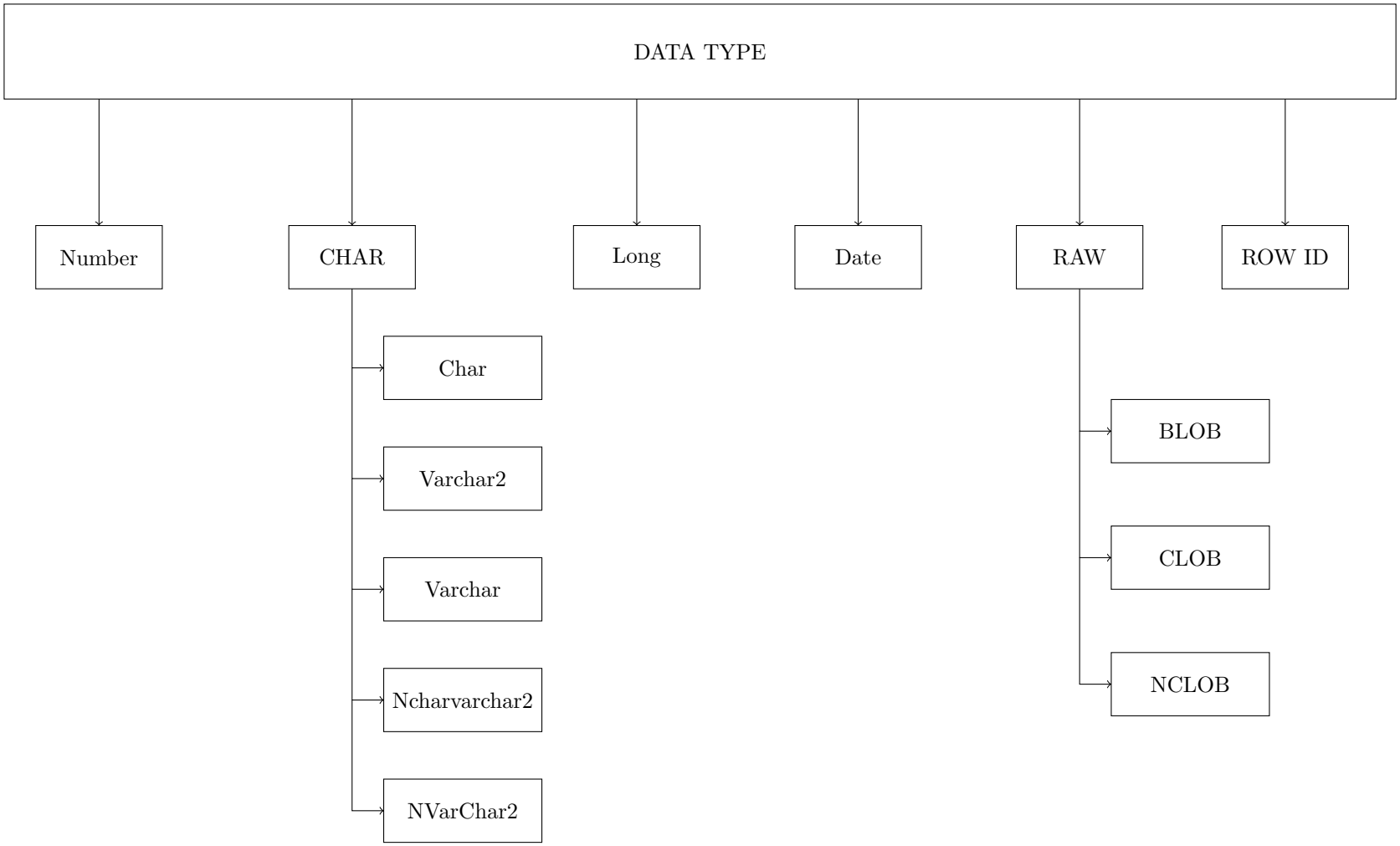
prettytitle

This is the content inside the pretty box.

1 Data Types

- Number :
- Char :
  - Char
  - Varchar
  - Varchar2
  -
  -
- Date :
- Long :
- Raw :
  - Blob :
  - Clob :
  - Nclob :
- Row ID :



2 Classification Of SQL Statement

- DDL :
  - Create :
  - Alter :
  - Drop :
  - Truncate :
  - Rename :
- DRL/SQL :
- DML :
  - Insert :
  - Update :
  - Delete :

- DCL :
  - Grant :
  - Revoke :
- TCL :
  - Commit :
  - Rollback :
  - SavePoint :

3 DDL Commands

3.1 Create Table

Definition

To create a table in oracle sql we just have to give the table a name and define each column known as attribut by giving each of them a name , a dataType and an optional constraint that can be added in same line of attribut definition or on its own line , we will see contraint in details in the next section

Syntax :

Table Creation

create table <table_name>(
<attribute$_1$> <DataType$_1$> [Constraint$_1$],
<attribute$_2$> <DataType$_2$> [Constraint$_2$],
.....
<attribute$_n$> <DataType$_n$> [Constraint$_n$],
[table contraints]
);

Example :

let's create student table

```
create table student (
id number,
firstname varchar2(50),
lastname varchar2(50),
grade number(2,2)
);
```

3.2 Table Constraints

Definition

Constraints are conditions set on the columns (attributes) of a table to ensure data integrity and consistency. Constraints can be defined:

- During table creation, either on the same line as the attribute definition or on a separate line
- After table creation using the ALTER TABLE command

There are two types of constraints: static and dynamic.

Static Constraints

Static constraints are fixed conditions that do not change based on data input.

- **NOT NULL**: Ensures that the attribute must have a value when inserting into the table.
- **UNIQUE**: Ensures that each value in the attribute is distinct. Unlike PRIMARY KEY, it allows null values.
- **PRIMARY KEY**: Combines UNIQUE and NOT NULL properties to ensure each value is unique and not null. Used to identify rows uniquely.
- **FOREIGN KEY**: References a primary key from another table to establish a relationship between tables.
- **DELETE ON CASCADE**: When deleting a row from the referenced (parent) table, all rows in the child table that contain the matching foreign key are also deleted.

Dynamic Constraints

Dynamic constraints apply conditions that can change based on specified criteria.

- **CHECK**: Validates a specified condition before allowing data to be inserted or updated.
- **DEFAULT**: Sets a default value for the attribute if no value is provided during insertion.

Syntax

Inline Constraint

<attribute$_i$> <DataType$_i$> not null
<attribute$_i$> <DataType$_i$> unique
<attribute$_i$> <DataType$_i$> primary key
<attribute$_i$> <DataType$_i$> references referenced_table(references_attribute)
<attribute$_i$> <DataType$_i$> default (value)

Outline Constraint

constraint <contraint_name> <attribute$_i$> not null
<attribute$_i$> not null

constraint <contraint_name> <attribute$_i$> unique
<attribute$_i$> unique

constraint <contraint_name> <attribute$_i$> primary key
primary key (attribute$_1$,..., attribute$_n$)

constraint <contraint_name> foreign key <attribute$_i$> references referenced_table(references_attribute)
foreign key <attribute$_i$> references referenced_table(references_attribute)

constraint <contraint_name> <attribute$_i$> default (value)
<attribute$_i$> default (value)

Example :

let's create a new table section and recreate the student table with constraints

Creating Section Table

Inline Method

```
    create table section (
    id_section number primary key,
    name varchar2(5) not null
    );
```

Outline Method

```
    create table section (
    id_section number,
    name varchar2(5),
    name not null,
    constraint pk_sec primary key (id_section)
    );
```

Create Student Table
Inline Method

```
    create table student (
    id number primary key,
    lastname varchar2(50) not null,
    firstname varchar2(50) not null,
    id_section number references section(id_section) on delete cascade,
    grade number(4,2) default 00.00 check (grade between 0 and 20),
    dob date not null check (dob<= add_months(sysdate,-18*12))
    );
```

Outline Mehtod

```
    create table student (
```

```
    id number,
    constraint pk_student primary key(id),
    lastname varchar2(50),
    firstname varchar2(50),
    constraint nn_student_lastname lastname not null,
    constraint nn_student_firstname firstname not null,
    id_section number,
    constraint fr_student foreign key (id_section) references section(id_section) on delete cascade,
    grade number(4,2),
    grade default 00.00,
    constraint chk_student_grade check (grade between 0 and 20),
    dob date not null,
    constraint chk_student_dob check (dob<= add_months(sysdate,-18*12))
    );
```

Note

Name Convention Of Constraint

- Primary Key : PK_<tableName>
- Foreign Key : FK_<tableName>
- Unique : UQ_<tableName>_<columnName>
- Check : CHK_<tableName>_<columnName>
- Default : DF_<tableName>_<columnName>
- Not Null : NN_<tableName>_<columnName>

Constraint Name Must Be Unique
Tables inside the same PDB (pluggable data base) can't share the same constraints name
Multiple Constraints
It is possible to define multiple constraints on a single attribute using the inline method. However, with the outline method, each constraint needs to be specified individually.

3.3 Delete Table

Definition

We can delete table using the drop command

3.3.1 Syntax

Table Deletion

drop table <tableName>;

3.3.2 Example

lets delete the section table we created

```
drop table section;
```

3.4 Rename Table

Definition

We can rename tables by using the rename command

Syntax

Renaming Table

rename <old_tableName>to <new_tableName>;

Example

```
rename section to mama;
```

3.5 Alter Table

Definition

The ‘ALTER‘ command is a versatile command that allows us to change various aspects of a table:

- Columns
  - **Renaming Column**: Rename the column.
  - **Modify Column**: Change the constraint and data type.
  - **Add Column**: Add a new column.
  - **Remove Column**: Remove a column.
- Constraints
  - **Add Constraint**: Add a new constraint.
  - **Remove Constraint**: Remove a constraint.
  - **Enable Constraint**: Enable an already existing constraint.
  - **Disable Constraint**: Disable an already existing constraint without deleting it.

Syntax

Columns Modification

Renaming Column
alter table <tableName>rename column <old_columnName>to <new_columnName>;
Modify Column
alter table <tableName>modify (columnName [new column definition & constraints]);
Add Column
alter table <tableName>add (columnName [column definition & constraints]);
Remove Column
alter table <tableName>drop column <columnName>;

Constraints

Rename Constraint
alter table <tableName>rename constraint <old_constraintName>to <new_constraintName>;
Add Constraint
alter table <tableName>add constraint <constraintName>[Constraint];
Remove Constraint
alter table <tableName>drop constraint <constraintName>;
Enable Constraint
alter table <tableName>enable constraint <constraintName>;
Disable Constraint
alter table <tableName>disable constraint <constraintName>;

Example

3.6 Truncate Table

Definition

To remove all rows from a table efficiently we use the truncate command

Syntax

Truncing Table

truncate table <tableName>;

Example

lets delete all records from student table

```
1    truncate table student;
```

4 DRL Commands

4.1 Select

Definition

To display the contents of one or more tables at once, we use the SELECT command. We can choose specific columns and tables to display, and if a table name is too long, we can assign a shorter alias to columns and tables name using the AS keyword. When selecting from multiple tables without a join condition, a Cartesian product occurs, meaning each row from one table is paired with each row from the other.

Syntax

Table Selection

SELECT * FROM $table_1$;
SELECT $column_1$,..., $column_n$ FROM $table_1$;
SELECT $t_1.col_1$,...,$t_1.col_n$, $t_2.col_1$,...,$t_2.col_n$ FROM $table_1$ AS t_1, $table_2$ AS t_2;

4.2 Where

Definition

The WHERE clause is used to filter rows in a table when displaying data with the SELECT command. Only rows that meet the specified condition(s) are shown in the result.

Syntax

Where Clause

SELECT $column_1$, $column_2$, ... FROM table WHERE [Conditions];

4.3 Order By

Definition

We can sort the results of a query in either ascending or descending order using ORDER BY. This can be applied to one or multiple columns. The order of the columns specified is important; the database first sorts by the first column, and if there are rows with identical values in that column, it then sorts those rows by the next column, and so on. This allows for a prioritized sorting strategy.

Syntax

Order By Clause

SELECT $column_1$, $column_2$, ... FROM table WHERE [Conditions] ORDER BY $column_1$ DESC ,..., $column_n$ ASC;

4.4 Group By

Definition

To group rows that have the same value in a specified column, we use the GROUP BY command. We can group by multiple columns; the order is important because it will first group by the first column. If there are rows that have the same value in the first column but differ in the second column, those rows will appear in separate groups in the output. This allows us to apply aggregate functions to summarize data for each group.

Syntax

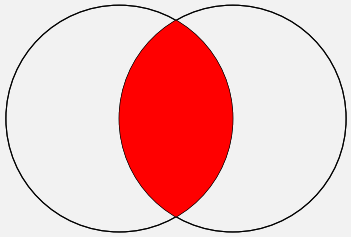
Group By Clause

SELECT column$_1$, column$_2$, ... FROM table WHERE [Conditions] GROUP BY column$_1$, ... ,column$_n$
ORDER BY column$_1$ DESC ,..., column$_n$ ASC;

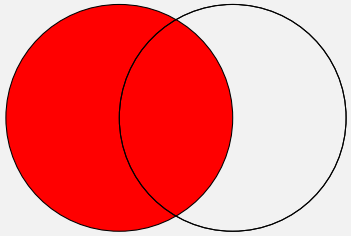
4.5 Joins

Definition

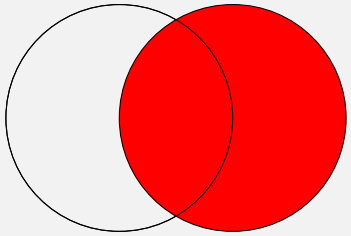
joins allow you to combine rows from two or more tables based on related columns (referenced key)
Inner Join An Inner Join returns only the common rows between tables



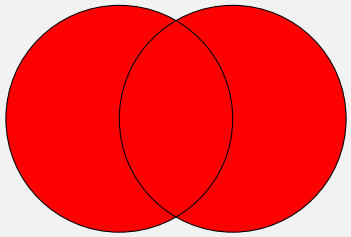
Left Join A Left Join returns all rows from the left table and the matched rows from the right table. If there's no match, NULL values are returned for columns from the right table.



Right Join A Right Join returns all rows from the right table and the matched rows from the left table. If there's no match, NULL values are returned for columns from the left table.



Full Join A Full Join returns all rows when there is a match in either left or right table. If there is no match, NULL values are returned for unmatched columns.



Syntax

Joins

Inner Join
SELECT column_1,...,column_n FROM table1 INNER JOIN table2 ON table1.common_column = table2.common_column;
Left Join
SELECT column_1,...,column_n FROM table1 LEFT JOIN table2 ON table1.common_column = table2.common_column;
Right Join
SELECT column_1,...,column_n FROM table1 RIGHT JOIN table2 ON table1.common_column = table2.common_column;
Full Join
SELECT column_1,...,column_n FROM table1 FULL OUTER JOIN table2 ON table1.common_column = table2.common_column;

4.6 Aggregation Functions

Definition

Aggregation functions perform calculations on a set of values and return a single result. They are commonly used in conjunction with the GROUP BY clause to summarize data.

- **Avg()** : Calculates the average (mean) of numeric values in a specified column.
- **Min()** : Returns the smallest (minimum) value in a specified column.
- **Max()** : Returns the largest (maximum) value in a specified column.
- **Count()** : Counts the number of non-null entries in a specified column (or all rows if * is used).
  - **count(*)** : Counts All rows
  - **count(column$_i$)** : counts number of rows where column_i is not null
  - **count(distinct column$_i$)** : counts number of rows where column_i is not null without repetition
- **Sum()** : Adds up all values in a specified numeric column.

4.7 Operators

Definition

Operators are symbols that specify operations to be performed on operands. They can be categorized as follows:

- **Logical Operators**: Used to combine conditions.
  - Logical And : AND
  - Logical Or : OR
  - Logical Not : NOT
- **Comparison Operators**: Used to compare values.
  - Equal : =
  - Not Equal : !=
  - Greater : >
  - Greater Or Equal : >=
  - Less : <
  - Less Or Equal : <=
  - Between : BETWEEN value_1 AND value_2
  - In : IN (set of values)
- **Arithmetic Operators**: Used for mathematical calculations.
  - Multiplication : *
  - Division : /
  - Sum : +
  - Subtraction : -

5 DML Commands

5.1 Insert

To insert rows into a table, we use the INSERT command. We can insert one row at a time or multiple rows at once from the same or different tables using the ALL keyword.

Syntax

Insert

Insert One Row At A Time
INSERT INTO tableName (column$_1$,...,column$_n$) VALUES (value$_1$,...,value$_n$);
Insert Multiple Rows
INSERT ALL
INTO tableName$_1$ (column$_1$,...,column$_n$) VALUES (value$_1$,...,value$_n$);
INTO tableName$_2$ (column$_1$,...,column$_n$) VALUES (value$_1$,...,value$_n$);
.....
INTO tableName$_n$ (column$_1$,...,column$_n$) VALUES (value$_1$,...,value$_n$);
SELECT * FROM dual;

5.2 Update

Definition

To change the values of some rows in a table, we use the UPDATE command, accompanied by the WHERE clause to update only specific rows.

Syntax

Update

UPDATE tableName SET column$_1$ = value$_1$,...,column$_n$ = value$_n$
WHERE [condition];

5.3 Delete

Definition

To delete rows from a table, we use the DELETE command, accompanied by the WHERE clause to delete specific rows. Although it is possible to delete all rows using DELETE, it is better to use TRUNCATE for that purpose due to performance considerations.

Syntax

Delete

DELETE SPECIFIC ROWS
DELETE FROM tableName WHERE [condition];
DELETE ALL ROWS
DELETE FROM tableName;

6 PL/SQL

6.1 Introduction

Definition

PL/SQL, or Procedural Language/Structured Query Language, is an extension of SQL. While SQL (Structured Query Language) is primarily used for CRUD operations (querying, inserting, updating, and deleting data in relational databases), PL/SQL allows for full programmatic control with features such as control structures (loops and conditionals), variables, and error handling with exceptions. This enables the creation of scripts that can automate tasks with functions, procedures, and triggers, implement complex business logic, and manipulate data at a higher level than SQL alone.

Differences Between PL/SQL and SQL

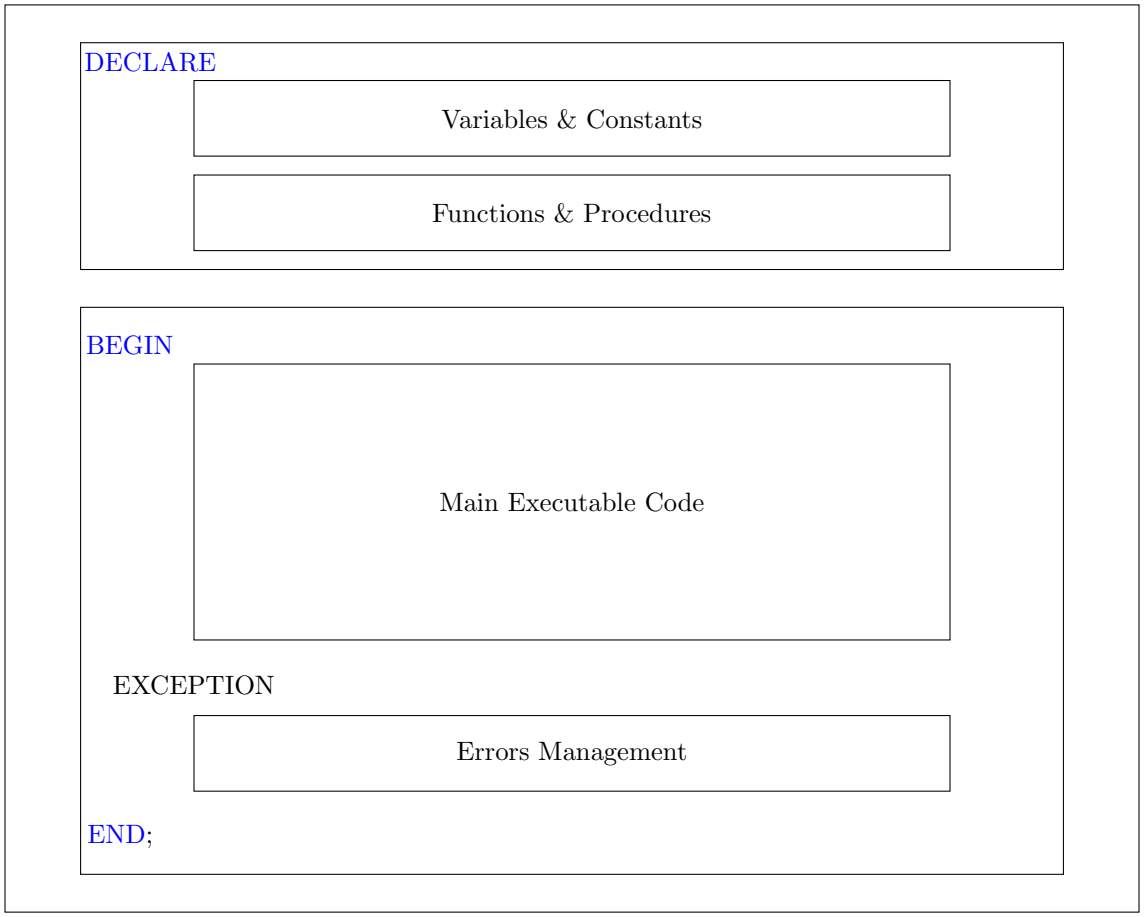
- SQL is limited to CRUD operations; PL/SQL adds procedural programming capabilities.
- PL/SQL provides advanced error handling through exceptions.
- PL/SQL supports modular programming with functions, procedures, and triggers.
- PL/SQL is specific to Oracle databases, whereas SQL is standardized across various databases.

6.2 Overview Of Plsql's Structure

Programme Structure

A PL/SQL has 3 blocks :

- DECLARE(Optional Block) : contains all the declared variables , constants & modules(functions,procedure)
- MAIN : contains the main executable code
- EXCEPTION(Optional Block) : handls erros with exceptions



6.3 Comments

Comments Syntax

One Line Comment:

```
- - Comment here
```

Multiple Lines Comment:

```
/*
This is a Mulitple
Lines Comment
*/
```

6.4 Printing

DBMS_OUTPUT.PUT_LINE

To print messages in the console, we use the `DBMS_OUTPUT.PUT_LINE` command. The message should be enclosed in single quotes ' ' and we use double pipes `||` to concatenate with variables:

DBMS_OUTPUT.PUT_LINE('Hello ' || name || '!');

Note

To be able to see the printed messages in the console of SQL*Plus, SQL Developer, etc., we need to activate the buffer responsible for printing the messages by using the command:

SET SERVEROUTPUT ON;

Note that this is only needed once , and it will remain active unless you explicitly turn it off.

6.5 Variables Declaration & Types

Variables & Constants

All variables and constants must be declared in the DECLARE scope the syntax is as follows

```
varName dataType := value                    - - Variable Declaration
constName CONSTANT dataType := value         - - Constant Declaration
```

Types

PL/SQL supports many standard data types, as seen previously. Here, we introduce two additional types:

- **Type:** Used to define a variable with the same data type as a column in a table:

  varName tableName.columnName%TYPE;

- **RowType:** Used to define a variable as a record with the structure of a row in a table:

  varName tableName%ROWTYPE;

Store Select Output In Variables

We can store the output of the SELECT command in variables using the INTO clause as follows:

SELECT col_1, col_2, ..., col_n INTO var_1, var_2, ..., var_n FROM tableName WHERE [conditions];

Note

Order Of Variables Is Important
The order of the variables in the INTO clause must match the order of the selected columns
Select Should Ouput One Line Only
When Storing the output of SELECT in variables , the ouput should be one line and not a table if not we will have to use cursor to navigate through the table we will cover that later on

6.6 Control Structures

Definition

In PL/SQL, control structures are constructs that help control the flow of execution in a block of code. They determine the order and conditions under which statements are executed and help make the code dynamic and responsive to varying conditions. The main types of control structures in PL/SQL are:

6.6.1 Conditional Control

If

```
IF condition1 THEN
- - statements to execute if condition1 is true
ELSIF condition2 THEN
- - statements to execute if condition2 is true
ELSE
- - statements to execute if none of the conditions are true
END IF;
```

Switch Case

```
CASE
WHEN condition1 THEN
- - statements to execute if condition1 is true
WHEN condition2 THEN
- - statements to execute if condition2 is true
ELSE
- - statements to execute if none of the conditions are true
END CASE;
```

6.6.2 Looping Control

Simple Loop

```
LOOP
- - statements to execute
EXIT WHEN condition; - - condition to exit the loop
END LOOP;
```

While Loop

WHILE condition LOOP
- - statements to execute while condition is true
END LOOP;

For Loop

Ascending

FOR counter IN start..end LOOP
- - statements to execute for each value of counter
END LOOP;

Descending

FOR counter IN REVERSE end..start LOOP
- - statements to execute for each value of counter
END LOOP;

6.7 Raise Application Error

Raise Errors

RAISE_APPLICATION_ERROR is a procedure used to raise an error that halts code execution, with a custom error message. Each error_code (between -20000 and -20999) is associated with an error message retrieved by SQLERRM, while SQLCODE captures the error code itself.

RAISE_APPLICATION_ERROR(error_code, error_message);

Though commonly used to handle user-defined exceptions, RAISE_APPLICATION_ERROR can also be used internally by the system for predefined exceptions, supporting error control in both system and custom PL/SQL operations.

6.8 Exceptions

Definition

Exceptions help manage errors and improve readability compared to directly using RAISE_APPLICATION_ERROR. Under the hood, exceptions are built on RAISE_APPLICATION_ERROR. There are two main types of exceptions:

- **Predefined Exceptions**: These are system-defined exceptions, such as:
  - NO_DATA_FOUND: Raised when a SELECT statement returns no rows.
  - TOO_MANY_ROWS: Raised when a SELECT statement returns more than one row.
- **User-defined Exceptions**: Defined by the user using the EXCEPTION DataType.

Syntax Exception

Exception management happens in the EXCEPTION block:

```
DECLARE
User_EXC EXCEPTION; - - Declare a custom exception
BEGIN
IF [Condition] THEN
RAISE User_EXC; - - Raise the custom exception
END IF;
- - Rest of code
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('No data found.');
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE('Too many rows returned.');
WHEN User_EXC THEN
DBMS_OUTPUT.PUT_LINE('Custom error occurred.');
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
```

Exceptions with SQLCODE

```
DECLARE
sqlcode_1 := -20001;
sqlcode_2 := -20002;
BEGIN
- - Check for specific conditions and raise custom errors with codes
IF [Condition] THEN
RAISE_APPLICATION_ERROR(sqlcode_1, 'error message1');
[Condition] THEN
RAISE_APPLICATION_ERROR(sqlcode_2, 'error message2');
END IF;
EXCEPTION
WHEN OTHERS THEN
- - Use SQLCODE to check error codes directly
IF SQLCODE = sqlcode_1 THEN
DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
ELSIF SQLCODE = sqlcode_2 THEN
DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
ELSE
DBMS_OUTPUT.PUT_LINE('Unexpected error: ' || SQLERRM);
END IF;
END;
```

Note

What is OTHERS?
It's best practice to add OTHERS as the last exception handler, as it catches any exceptions not explicitly defined. This ensures any unexpected errors are managed gracefully.
When to Use RAISE_APPLICATION_ERROR vs. Exceptions?
Although exceptions are built on RAISE_APPLICATION_ERROR, they offer better readability and manageability in complex code. Use exceptions for organized error handling, while RAISE_APPLICATION_ERROR provides a more direct and minimalistic approach.

6.9 Cursors

cursor

```
DECLARE
    -- Declare a cursor that selects data
    CURSOR cr IS
        select query

    -- Variables to hold fetched data
    var1 table.col1%TYPE;
    var2 table.col2%TYPE;
    .....
    varn table.coln%TYPE;
BEGIN
    -- Open the cursor
    OPEN emp_cursor;
    FETCH cr INTO var1,var2,...,varn
    -- Loop through the result set
    WHILE(cr%FOUND) LOOP
        --traitements
        FETCH cr INTO var1,var2,...,varn

    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
END;
```

6.10 Triggers

6.11 Functions

6.12 Procedure