Friday, November 29, 2024 1:22 PM

Thanku for the opportunity for interviewing me , My name is Shailvi I am from Chhapra, Bihar.
I bring two years of valuable experience in Supply Chain Management (SCM) project within the At&t and gained Good Techno - Functional Knowledge on Supply Chain modules..
During this time, I have had the privilege to work on various applications like Oracle EBS, UC4,Toad, Putty in live production environments.
This experience has enabled me to develop a strong foundation in understanding complex business workflows
and collaborating with cross-functional teams to ensure smooth and efficient operations.
I also lead the team, facilitating cross-team connections, managing client relationships on daily basis, and resolving critical issues across various areas.
While working on this project, I have sharpen my skills in PLSQL .
In this span of time, I have dedicated significant time to upskilling and gaining proficiency in PLSQL tools and concept.
In this learning journey, I have created stored procedures, packages, procedure, functions,triggers from the existing or required queries.
I have hands on experience creating stored procedures, table, views. I also spend time on optimizing query.
Currently, I'm eager to leverage my skills in PL/SQL. As I have skill to prove on,
I will try my best to stand on the expectations by doing quality of work.
Apart from this, I have also runner up to final round in idelivered accenture.

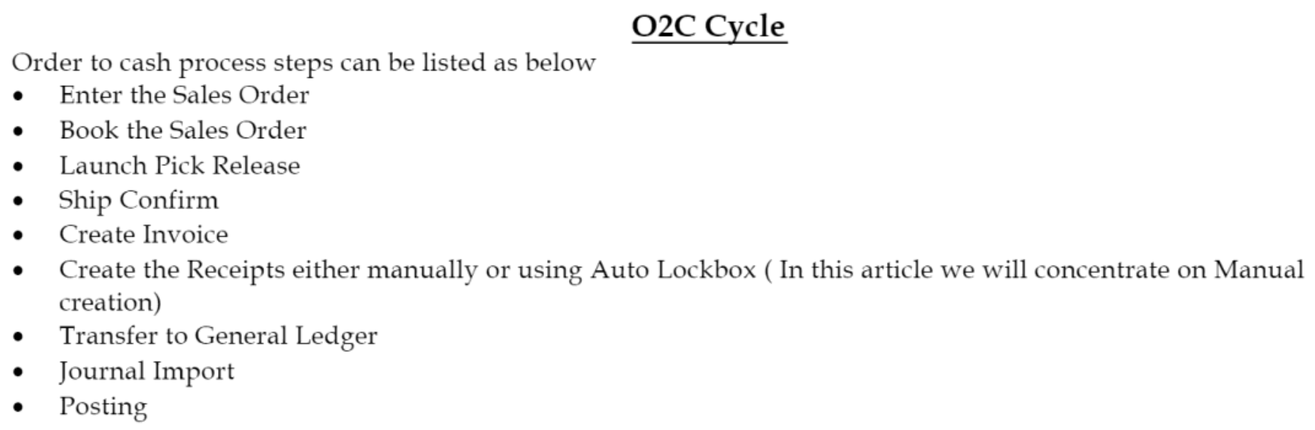
- Make own procedure
- Long query to short query -- run and merge

Thank you for your time and consideration, and I look forward to the opportunity to contribute to your team.

the management of the flow of goods and services, including all the processes of turning raw materials into final products.

- **What is Pick Release:**
  Pick Release is part of Order To cash process in terms of business cycle and is part of Order Management Module in Oracle EBS application.

- **When customer ordered an item first it will come to opus which is front end of AT&T system, then from front end to Yoda which is middleware system and it validates the orders (credit card check, address details etc.,) before coming to AT&T system then from Yoda to oracle .**
- **Yoda will validate the orders and send the orders to oracle to fulfill the order and oracle will do some tasks and send to WMS(warehouse management system) and once WMS fulfill the order then it will send the fulfilment details back to oracle and oracle will send the fulfilment details to Yoda back for the recording purpose.**
- **Once the order is shipped to the customer from WMS it will share the shipment details to oracle and oracle will share the shipment details to Yoda.**



O2C Cycle

Order to cash process steps can be listed as below

- Enter the Sales Order
- Book the Sales Order
- Launch Pick Release
- Ship Confirm
- Create Invoice
- Create the Receipts either manually or using Auto Lockbox (In this article we will concentrate on Manual creation)
- Transfer to General Ledger
- Journal Import
- Posting

In project -
Worked on Tools - Appworx, Putty, Toad, Oracle EBS, PPM Tool,
Technologies - Oracle Apps Development (P3)

I am eager to learn new things and have never give up and self-motivated.
I considered myself as a well-focused, flexible and a confident person.
I am a quick learner and grab knowledge related to project quickly, ready to take challenges.

Problem faced --
The main problem was collection of datasets, we searched on various platforms and we found some of the things but it was not sufficient for our needs then we consult with our seniors.
They suggested us some great idea and platform where we can find out easily the idea and datasets. After that we build our own project.

Some other problems like -- choosing idea, to manage our team and coordinate, gathering team members.

My college minor project -
Train announcements -- Software includes arrival, departure schedule announcements of train.
Technologies - OS,Github, Pydub, Pandas

House Price Predictor - Model to predict the price of house that to be sold by analyzing the dataset.
Technologies - Numpy, pandas, Matplotlib, seaborn, sklearn, Tkinter

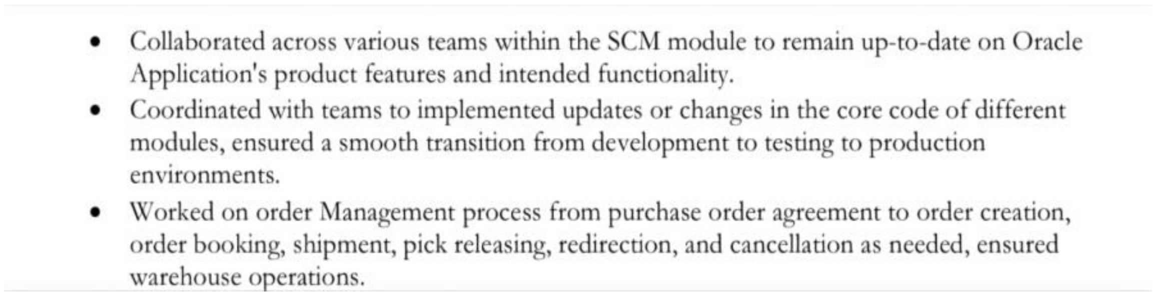
Make a machine learning model to predict the price of a house that to be sold

1. Oracle Certification (Active) in Primary Skill
2. P3 - Proficiency in my Competency
3. DFA (In progress)
4. GEN AI level 2 Training (In progress)
5. Certifications Completion -
Oracle PL/SQL --(In progress)
6. Learning multiple things related to Night Ops applications.
7. Industrial training on Communication technology (In progress)
8. Essential Training -- (In progress)

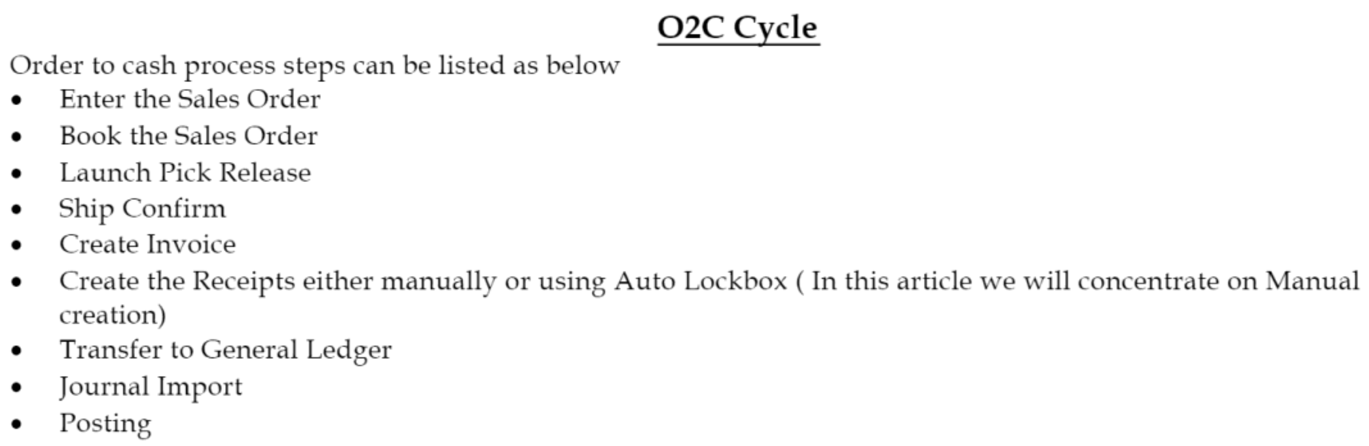
About Project

Monday, December 9, 2024 12:49 PM

Expertise in working with end-users for support, troubleshooting and
giving effective & efficient resolutions for the technical issues.
• Good Techno - Functional Knowledge on Supply Chain modules.
• Engaged in technical development areas of oracle application 11i E-Business suite, •
After a migration from UNIX to LINUX, working on migrating the application to the cloud.
• Conducting weekly load tests to determine server operating limits.
• Built an application from the scratch to provide ease to the business monitoring for various modules, different primaries can check and verify data, which has reduced time complexity.
• In need of any updates/changes in the integral codes of any modules, coordinated with teams from testing that in Dev environment to Testing to migrating it in Production.
• In AT&T, from the point Purchase orders agreement is done and orders are created by the customer. From booking of the order to shipment, pick releasing and redirecting and cancelling as per the requirement and then getting it dropped to the warehouse.
• Coordinating cross-functionally with different teams under SCM module to stay current on product features and intended functionality in Oracle Application.



- Collaborated across various teams within the SCM module to remain up-to-date on Oracle Application's product features and intended functionality.
- Coordinated with teams to implemented updates or changes in the core code of different modules, ensured a smooth transition from development to testing to production environments.
- Worked on order Management process from purchase order agreement to order creation, order booking, shipment, pick releasing, redirection, and cancellation as needed, ensured warehouse operations.



O2C Cycle

Order to cash process steps can be listed as below

- Enter the Sales Order
- Book the Sales Order
- Launch Pick Release
- Ship Confirm
- Create Invoice
- Create the Receipts either manually or using Auto Lockbox (In this article we will concentrate on Manual creation)
- Transfer to General Ledger
- Journal Import
- Posting

UC4 is a powerful application job scheduling tool that meets the needs of operators, programmers, and system administrators throughout the life cycle of an application.
UC4 Job Scheduling is a service that enables the enterprise to schedule and monitor computer batch jobs. The scheduler can initiate and manage jobs automatically by processing prepared job control scripts and statements
AT & T provide each application has create unique MOTS ID, UC4 application MOTS ID: 22748
UC4 has always been known for their industryleading and innovative capabilities in scheduling applications across enterprises. UC4 continues this tradition with the Applications Manager release, once again demonstrating why it is the most innovative task scheduling company today.

Application Manager (aka Appworx) is a software product from Automic that acts as a single source for scheduling and managing jobs across different platforms and applications in a manner that allows tight coupling of jobs between systems. Effectively, when systems are coupled in this manner, all jobs that run on those systems appear as if they are being managed under ONE application; in this case, under Automic Application Manager.

Requirement need

- The jobs use to run in Oracle application and it was a very cumbersome task to hold and release jobs manually and to avoid that one automated tool was needed
- There are many other task which were becoming difficult to handle like scheduling of jobs, Activation and deactivation of jobs
- The most difficult part was at the time of maintenance, where cancelling jobs and monitoring all of them one by one by punching in the request id was very time consuming
- The jobs were also not categorized so a tool was indeed needed where we can bifurcate a bunch of jobs according to the Application

Advantages

- Create jobs to run programs and scripts.
- Create process flows to run a series of jobs.
- Add dependencies to process flow components to establish the correct execution order in process flows.

- Define job parameters.
- Add IF - THEN logic to jobs and process flows to ensure the correct conditions exist before they execute.
- Automate retrieval of values from databases to eliminate data entry errors.
- Schedule jobs and process flows to automate production.
- You can type the first few letters of a task name from the Backlog's ***Search*** field, and Applications Manager will find it rather having need to type in the full name
- Two new replacement values have been added. They are:
- {job}: The job name rather than its alias
- {subflow_id}: The flow id of the individual flow rather than its top level parent
- Applications Manager keeps track of the time each user logs into or out of the Applications Manager system. Monitoring logins is becoming one of the critical elements required to meet compliances

Night Ops expanded as Night Oracle Production Support is the team which is responsible for taking care/monitoring the oracle production support activities

Weekend Maintenance Activity (Overview)
Every weekend we need to bring down the system in order to perform some activities (Patching, Code change, Deployment). These activities are done in quiet time when all the jobs of that instance are on Hold.
Let's say that we have a scheduled maintenance activity starting from 22.00 PM to 6.00 AM (Tentative) EST every Saturday. A call is scheduled to know about all the instances that will fall under maintenance (Sis call). We decide on tasks need to be performed, deployments and free hours.
For instance, if anybody wants to perform a code change in a program, that particular action is to be fulfilled during the silent time which is from 22.00 PM to 6.00 AM EST. The team will hold all the Jobs, perform certain pre-maintenance and post-maintenance activities. During the silent time, the DBA team will get the system without any jobs running so that they can do the patching easily and there are no interruptions of any sessions running. This is the concept of bringing down the system. After the completion and validation of the changes made, we bring up the system and inform all the users.

Forecast: -
- We submit a job called Forecast. Forecast can be added to give the system an idea that we are planning to have maintenance activity due to which agents and queues may go down, jobs might be unscheduled for a specific time so that every related system and users are ready and prepared for this. This is submitted manually.

What is C2W?
C2W is a generic term used to describe Incident Management's process of engaging support personnel during outages. It's a coordinated paging process that brings application-specific experts to a conference-call quickly so they can assist with the resolution.

Need for C2W
Whenever there is an issue or outage, may be related to application, server, database, etc., at first, we try to solve it on the bridge with the concerned team. However, in the case the issue is urgent and need immediate attention (by the SA Team) which may cause business or application impact down the road, we create a C2W.
After C2W is raised, all the teams along with SA come together and solve the problem.

1. Introduction: During the holiday's few DC's will be working and few not. We divert the order from one warehouse to another so it will drop order as per our directory setting.
   Saturday : 01:30 AM IST (4:00PM ET)
   Sunday(revert) : 12:30AM IST (3:00 PM ET)

Requirement: IMM Sync Activity – Sub Inventory Transfer of SKUs from DF, MAIN TO HOLD. (Inv Sync file sent by IMM for both IDC & RDC, we also get Special request from Omar to move some RDC SKU's)

Note: *We are doing this Activity from* ***Tue to Sat at 12 AM ET.***

IMM: (Ingrammicro) Warehouse name (IDC & RDC)

SKU: (Stock Keeping Unit) A number which is assigned to a product in order to identify specific information such as color, [style, brand, size and so on.

Why IMM sync Activity required.
To balance the on-hand differences between Oracle & WMS systems.

MANU ORDERS:
- Manu orders is one of the most important task under which we process all the orders coming to AT&T. There are many important jobs and processes which we monitor in our daily routine for this process.
- There are various types of orders coming in, ex. DF orders, UVERSE orders, similarly there are MANU orders. These are basically bulk orders which are not going to the customers but to the warehouses (only specific quantity daily). Hence optimizing the number of orders that are required to the DCs.
- Everyday 1 AM EST we receive an estimation of orders from OLM, based on the number we start processing requisitions at Oracle end and need to make sure those many orders are also created by 5 AM EST.
- To monitor the MANU order, we basically need Appworx, Toad and monitor the alert emails.

Forecasting At JDA:
- PF's included - OLM_ORACLEAMPMINTERFACES_PF & OLM_DAILY_SP13O_RNS_PF

- On weekdays forecasting activity will start at 12:00 AM ET
- OLM_ORACLEAMPMINTERFACES_PF -- Process starts with this PF at 22:25 PM ET

- Jobs which include in this PF: Application –
    - CCWOHIMO
    - CCWWTMI
    - CWPOMNPOEXP
    - CCWOSOMI
    - CUST_DL_VAN_FUL_EXTRACT
    - CUST_ALC_ONHAND_INV_TO_JDA
    - CW_INTRANSIT_INVENTORY (Inventory job)
- Before JDA does forecasting, they will check on hand inventory also with oracle which is done through OLM_ORACLEAMPMINTERFACES_PF

- **What is Pick Release:**
  Pick Release is part of Order To cash process in terms of business cycle and is part of Order Management Module in Oracle EBS application.

- **When customer ordered an item first it will come to opus which is front end of AT&T system, then from front end to Yoda which is middleware system and it validates the orders (credit card check, address details etc.,) before coming to AT&T system then from Yoda to oracle .**
- **Yoda will validate the orders and send the orders to oracle to fulfill the order and oracle will do some tasks and send to WMS(warehouse management system) and once WMS fulfill the order then it will send the fulfilment details back to oracle and oracle will send the fulfilment details to Yoda back for the recording purpose.**
- **Once the order is shipped to the customer from WMS it will share the shipment details to oracle and oracle will share the shipment details to Yoda.**

DF YODA

- DF Yoda handles wireless orders
- DF Yoda handles business for wireless customers and it sits b/w customer front end and oracle application
- 98% of DF Yoda jobs in uc4 and 2% of jobs at server level
- Below are the Database connection details

Secondary DB connection Sting (For more than 5minutes use this string)

- NHYODA handles wireline orders
- Different types of ordering systems are interfaced with NH YODA and one more kind of front-end system is also there along with NH Yoda which Entertainment for Uverse and DTV
- NHYODA jobs will run in AZURE database
-

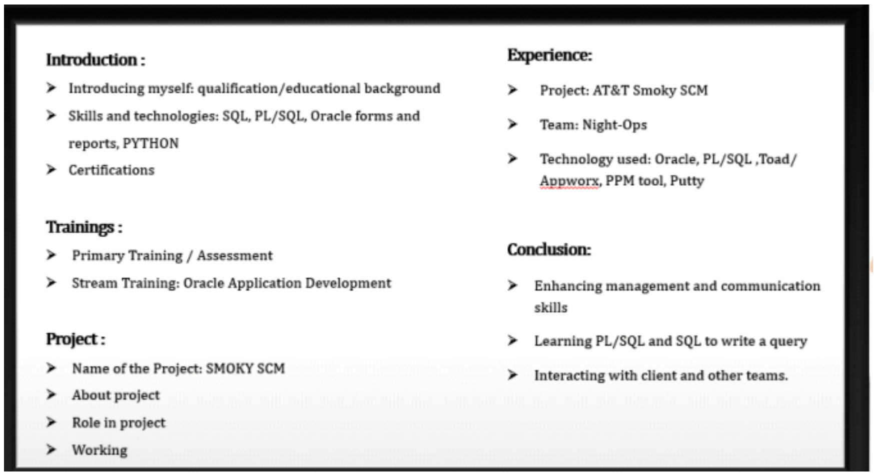
Migration is a process where we are moving code between instances, in AT&T we control all changes thru a process of authorization using a CR and apply automatic script changes thru PPM tool. Some manual migration instructions are included on a BR100 document not covered here.
In this document we'll divide migrations in next 2 topics:
1. CR Management.
2. PPM number.

(PDF) Order to Cash (O2C) Cycle with Table details in Oracle Apps | Joydeep Bonner - Academia.edu

P2P Cycle in Oracle Apps (Step by Step Process and Tables)

idel

Wednesday, February 28, 2024 7:47 PM



Introduction :

- Introducing myself: qualification/educational background
- Skills and technologies: SQL, PL/SQL, Oracle forms and reports, PYTHON
- Certifications

Trainings :

- Primary Training / Assessment
- Stream Training: Oracle Application Development

Project :

- Name of the Project: SMOKY SCM
- About project
- Role in project
- Working

Experience:

- Project: AT&T Smoky SCM
- Team: Night-Ops
- Technology used: Oracle, PL/SQL ,Toad/ Appworx, PPM tool, Putty

Conclusion:

- Enhancing management and communication skills
- Learning PL/SQL and SQL to write a query
- Interacting with client and other teams.

My name is Shailvi. I am from Bihar. I have persuade my B.Tech in Electronics and communication engineering from JECRC, Jaipur.
I got my 1st job in accenture and joined on 10th oct 2022 where I have done my training in oracle and hard lock on AT&T on 8th dec 2022 now currently working in smoky SCM project.

Certifications Completion -
Oracle Cloud Infrastructure 2022 Foundations Associate
Oracle APEX Cloud Developer
Oracle Cloud Platform Application Integration
Oracle Inventory Cloud 2022 Certified

1. Oracle Certification
2. P3 - Proficiency in myCompetency
3. DLF, SFA Trainings
4. TQ (12/12)
5. PACT Training
6. SI Primer Academy Learning (83%)
7. GEN AI Training (In progress)

Supply chain management is the handling of the entire production flow of a good or service — starting from the raw components all the way to delivering the final product to the consumer.

(AT&T) is a provider of telecommunications, media, and technology services.

Raw material -> supplier-> manufacturing ->distribution ->customer->consumer

Role in project -- monitoring and supporting the SMOKY SCM

SQL -
Structured Query Language. SQL is a program created and formulated in the Relational Database Management System to handle structured data.

PL/SQL -
can create and run PL/SQL program units such as procedures, functions, and packages.

Oracle forms -
Oracle Forms is used to develop and deploy Forms applications.

Oracle reports -
Forms can be used for both input and output. Reports, on the other hand, are used for output, i.e., to convey information on

a collection of items.

Python -
Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum.

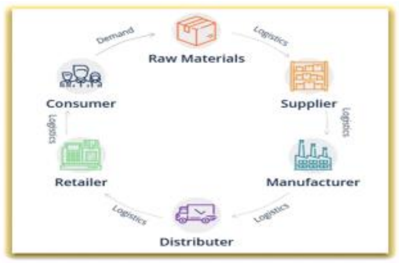
Supply chain technology it enhances collaboration by data sharing and communication among various stakeholders. With technologies like cloud computing, stakeholders can access and share crucial data in real-time, regardless of their geographical location.

Lets take an example of fabric for better understanding --> fabric mades up of cotton,silk,jute etc.which are raw materials. And suppliers supply to the manufacturer for making fabric once fabric is ready then distributor collects it from manufacturer and distributes to the different retailer and customer purchases that fabric from retailer which increase in the demand of that fabric which will automatically increased the production of raw materials and these cycle goes on this is known as supply chain for managing this whole processes we terms that as supply chain management.

What is Supply chain management?
the management of the flow of goods and services, including all the processes of turning raw materials into final products.
and getting them to the ultimate customer. Keith oliver was first coined this in 1982.

What is the meaning of chain in supply chain management?
the network of all the individuals, organizations, resources, activities and technology involved in the creation and sale of a product.

Why do we need SCM?
keeps the mechanisms of supply and demand operating smoothly so that people have access to goods and services.



For same example there are some basic Key components?

1) **Planning** - 1st stage -- used to address how a given good or services will meet the needs of the customers.
   Make sure plan a profitable supply chain
2) **Sourcing** - the whole process of evaluating and selecting the suitable suppliers who help the organization in maintaining its competitiveness in the market. Quality of goods and cost savings are necessary criteria to consider before choosing a sourcing partner.
   the process of strategically choosing the right services and goods that a company needs to run their business.
3) **Manufacturing** -
   comprises all the processes & quality control a business uses to turn raw materials into final products that are ready to be sold to customers.
4) **Transporting** -
   the movement of raw materials, work-in-processes, and final products from one location to another
5) **Delivery** -
   where an actual product leaves the warehouse and gets sent to a retailer, customer,

   Challenges?

1) **globalization** refers to the free movement of goods, services, and people across the world.
   With goods and services travelling across borders, there are more factors to consider, from customs regulations to currency exchange rates. This increased complexity can lead to higher costs and greater risk

2) **Demand volatility -**
   demand volatility is defined as variations in demand for products that include rapidly changing and unpredictable market.
   It requires a proactive and adaptive approach to demand forecasting,

3) Disrupted Transportation and Logistics: Natural disasters can lead to road closures, port shutdowns, and disruptions in transportation networks. This hampers the movement of goods, causing delays and increased transportation costs.

4) Inventory management complexities -
- Rapidly changing customer demand. ...
- Inaccurate data and analysis. ...
- Reordering delays. ...
- Poor production planning.

5) Information and communication barriers -
   Stakeholders can have different backgrounds, cultures, languages, interests, expectations, and preferences, which can affect how they communicate and interpret information

   Strategies & example for effective supply chain management?

responsible for negotiating and contracting for goods and services for the AT&T enterprise. In addition, AT&T procurement responsibilities include delivering goods and services in ways that guarantee quality and value to our clients throughout AT&T.

- Collaboration and Communication.
  - Buid strong relationships with suppliers and partners
  - Utilizing technology for real time information sharing
- Risk Management.
  Map out our supply chain to get a clear understanding of which entities are most vulnerable to risk.
  Implement contingency plans and diverse suppliers accordingly.
- Technology integration.
  make the flow of data and goods more streamlined and efficient
  Adopt advanced analytics for deamnd forecasting.
- Sustainability and Ethical Practices.
  - Incorporating environmentally friendly practices
  - Ensuring ethical sourcing and fair labor practices

Conclusion --

improve financial performance; lead to satisfied customers; reduce delivery times; and build trust, confidence and commitment among suppliers.

PLSQL basic definitions

Wednesday, January 15, 2025 11:05 AM

Programming language used for managing data in databases.

Basic of PL/SQL -

PL/SQL engine compiles PL/SQL code into byte-code and executes the executable code.
Once we submit PL/SQL code to the Oracle Database server, the PL/SQL engine collaborates with the SQL engine to compile and execute the code.
PL/SQL engine runs the procedural elements while the SQL engine processes the SQL statements.

Basis of Comparison	SQL	PL/SQL
Definition	It is a database Structured Query Language.	It is a database programming language using SQL.
Variables	Variables are not available in SQL.	Variables, constraints, and data types features are available in PL/SQL.
Control structures	No Supported Control Structures like for loop, if, and other.	Control Structures are available like, for loop, while loop, if, and other.
Nature of Orientation	It is a Data-oriented language.	It is an application-oriented language.
Operations	Query performs the single operation in SQL.	PL/SQL block performs Group of Operation as a single block resulting in reduced network traffic.
Declarative/ Procedural Language	SQL is a declarative language.	PL/SQL is a procedural language.
Embed	SQL can be embedded in PL/SQL.	PL/SQL can't be embedded in SQL.
Interaction with Server	It directly interacts with the database server.	It does not interact directly with the database server.
Writes	It is used to write queries using DDL (Data Definition Language) and DML (Data Manipulation Language) statements.	The code blocks, functions, procedures triggers, and packages can be written using PL/SQL.
Processing Speed	SQL does not offer a high processing speed for voluminous data.	PL/SQL offers a high processing speed for voluminous data.
Application	You can fetch, alter, add, delete, or manipulate data in a database using SQL.	You can use PL/SQL to develop applications that show information from SQL in a logical manner.

Block - Basic programming unit in PLSQL. PLSQL code is organized in blocks.

Consists of 3 sections -

- **Declaration**
  - Declare variables, allocate memory for cursors, and define data types
- **Executable (Mandatory)**
  - Starts with BEGIN and ends with keyword END
  - Must have 1 statement to execute even if it is NULL

- **Exception-handling**
  - Starts with the keyword EXCEPTION
  - Here we can catch and handle exceptions raised in execution section.

Types - Anonymous, Named

Anonymous Block	**Named Block**
Block without name	Block has name
Used for testing purposes.	Ex - Functions, procedure
Not saved in Oracle DB server, so just for one-time use.	Stored in Oracle database server and reusable

DATA TYPES -

1) Scalar

    Store single values -- number,boolean,character,datetime

2) Composite

    Store multiple values -- record and collection

Number,binary_float,binary_double - sql data types.

- **PLS_INTEGER** (uses hardware arithmetic)-- PL/SQL --> requires less storage than number range from -2,147,483,648 to 2,147,483,647., faster than number(which uses software arithmetic).
    - Subtype - NATURAL (non-negative pls_integer values) , POSITIVE(pls_integer)
- **Boolean** --- TRUE,FALSE AND NULL -- we can use in IF,THEN,CASE,LOOPS
- **Character** -- char(n) - fixed length - range from 1 to 32,767 bytes.
- **Varchar(n)** - varying length character -range from 1 to 32,767 bytes.
  long, raw, long raw, rowid, and UROWID.
- **Date time** -- Date, Timestamp

SubType - subset of another dataType

PLSQL Large object (LOB) Data Types -

-- LOB data types store large amounts of unstructured data, such as text, images, videos and audio.

PLSQL provides several datatypes
BLOB (Binary large object) - store binary large objects, such as images or multimedia files.
CLOB (Character large object) - Stores large character data
NCLOB (National Character large object) - stores large character data using the national character set.
BFILE (Binary File) - Stores a reference to a binary file stored outside of the database.

The GOTO statement allows you to transfer control to a labeled block or statement.

GOTO statement restrictions

1) First, you cannot use a GOTO statement to transfer control into an IF, CASE or LOOP statement, the same for the sub-block
2) Second, you cannot use a GOTO statement to transfer control from one clause to another in the IF statement e.g., from IF clause to ELSIF or ELSE clause, or from one WHEN clause to another in the CASE statement.
3) Third, you cannot use a GOTO statement to transfer control out of a subprogram or into an exception handler.
4) Fourth, you cannot use a GOTO statement to transfer control from an exception handler back into the current block.

- PL/SQL NULL statement does nothing but serves as a placeholder statement.

LOOP -

LOOP statement is a control structure that repeatedly executes a block of code until a specific condition is met or until you manually exit the loop.

The CONTINUE statement allows you to exit the current loop iteration and immediately continue on to the next iteration of that loop.

Select * into var1 --> * **>** var1 --> not enough values and * **<** var1 --> too many values

Cursor

Thursday, January 9, 2025 5:37 PM

Cursor is defined as private worked area where the SQL statements (Select & DML) are executed.
It is a pointer that points a result set of query.

$$\begin{tikzcd} & \textbf{Cursor} \arrow[dl] \arrow[dr] & \\ \textbf{Implicit Cursor} & & \textbf{Explicit Cursor} \arrow[d] \\ & & \textbf{Named, FOR, Cursor Parameters, REF (Strongly \& Weakly Typed REF Cursor)} \end{tikzcd}$$

Implicit Cursors	**Explicit Cursors**
Implicit cursors are automatically created when select statements are executed.	Explicit cursors needs to be defined explicitly by the user by providing a name.
They are capable of fetching a single row at a time.	Explicit cursors can fetch multiple rows.
Closes automatically after execution.	Need to close after execution.
Implicit cursors are less efficient.	Comparative to Implicit cursors, explicit cursors are more efficient.

Implicit Cursors are defined as:
BEGIN
SELECT attr_name from table_name
where CONDITION;
END
	Explicit cursors are defined as:
	DECLARE
	CURSOR cur_name IS
	SELECT attr_name from table_name
	where CONDITION;
	BEGIN
	…

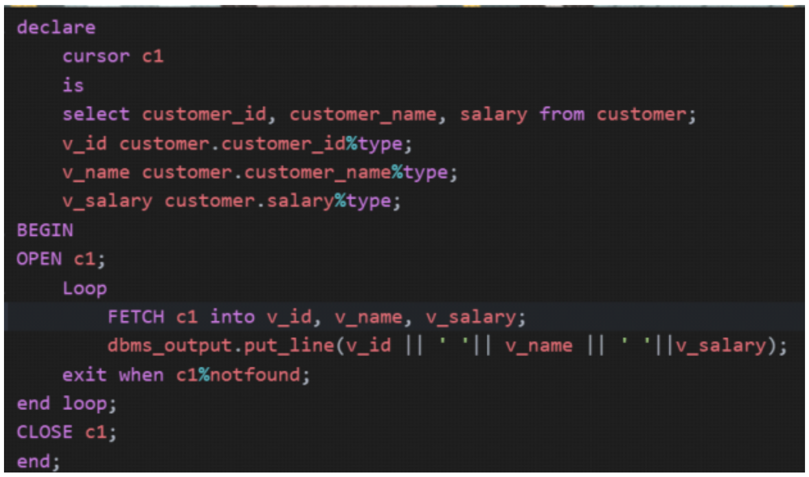
Cursor attributes use prefix "SQL". Structure for implicit cursors: SQL%attr_name Few implicit cursors attributes are: SQL%FOUND, SQL%NOTFOUND, SQL%ROWCOUNT	Structure for explicit cursors: cur_name%attr_name Few explicit cursors are: cur_name%FOUND, cur_name%NOTFOUND, cur_name%ROWCOUNT, %isopen

Maximum limit - 50 -1000 - depending on data configuration, memory

Explicit cursors :-

Simple Cursor ex-

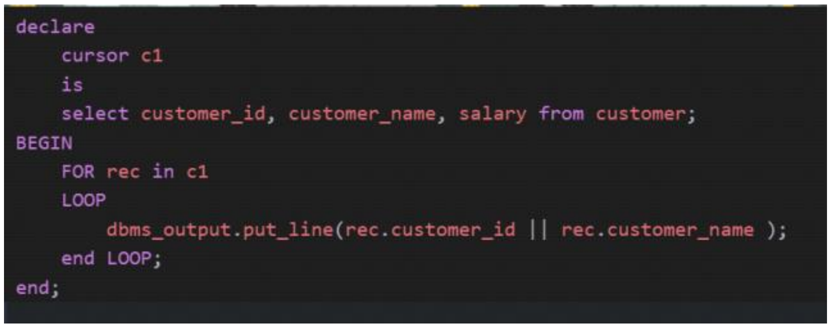
```
declare
   cursor c1
   is
   select customer_id, customer_name, salary from customer;
   v_id customer.customer_id%type;
   v_name customer.customer_name%type;
   v_salry customer.salary%type;
BEGIN
OPEN c1;
   Loop
                  FETCH c1 into v_id, v_name, v_salry;
                  dbms_output.put_line(v_id || ' '|| v_name || ' '||v_salary);
          exit when c1%notfound;
end loop;
CLOSE c1;
end;
```



```
declare
    cursor c1
    is
    select customer_id, customer_name, salary from customer;
    v_id customer.customer_id%type;
    v_name customer.customer_name%type;
    v_salary customer.salary%type;
BEGIN
OPEN c1;
    Loop
        FETCH c1 into v_id, v_name, v_salary;
        dbms_output.put_line(v_id || ' '|| v_name || ' '||v_salary);
    exit when c1%notfound;
end loop;
CLOSE c1;
end;
```

To overcome above complex code (instead of using OPEN/FETCH/CLOSE) we can use FOR cursor LOOP -

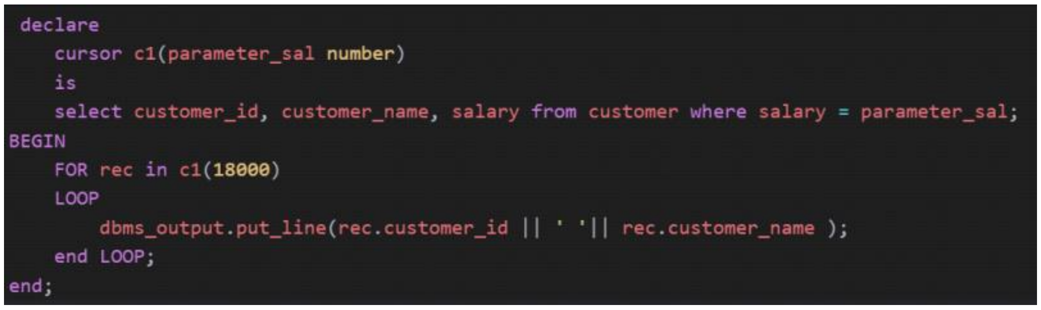
```
declare
   cursor c1
   is
   select customer_id, customer_name, salary from customer;
BEGIN
         FOR rec in c1
   LOOP
                  dbms_output.put_line(rec.customer_id || rec.customer_name );
   end LOOP;
end;
```



```
declare
    cursor c1
    is
    select customer_id, customer_name, salary from customer;
BEGIN
    FOR rec in c1
    LOOP
        dbms_output.put_line(rec.customer_id || rec.customer_name );
    end LOOP;
end;
```

Above process is used for particular 'where' clause for same 1 table, **Cursor parameters** is used in more than one place for the same where clause.

```
 declare
   cursor c1(parameter_sal number)
   is
   select customer_id, customer_name, salary from customer where salary = parameter_sal;
BEGIN
         FOR rec in c1(18000)
   LOOP
                  dbms_output.put_line(rec.customer_id || ' '|| rec.customer_name );
   end LOOP;
end;
```



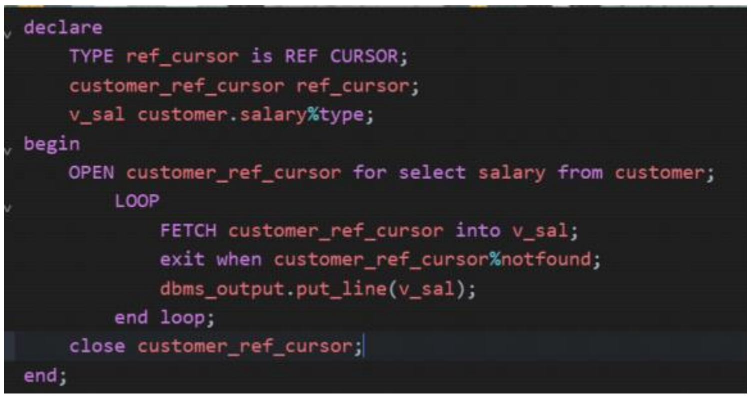
```
 declare
    cursor c1(parameter_sal number)
    is
    select customer_id, customer_name, salary from customer where salary = parameter_sal;
BEGIN
    FOR rec in c1(18000)
    LOOP
        dbms_output.put_line(rec.customer_id || ' '|| rec.customer_name );
    end LOOP;
end;
```

*To know how many Cursor is opened - select * from v$open_cursor where user_name ='schema_name';*

If we have to fetch data from 2 table using same single code (avoiding to make 2 cursor) -- so we use REF CURSOR
REF Cursors are opened with an 'OPEN FOR' statement.

Weak REF Cursor - Don't have any return type

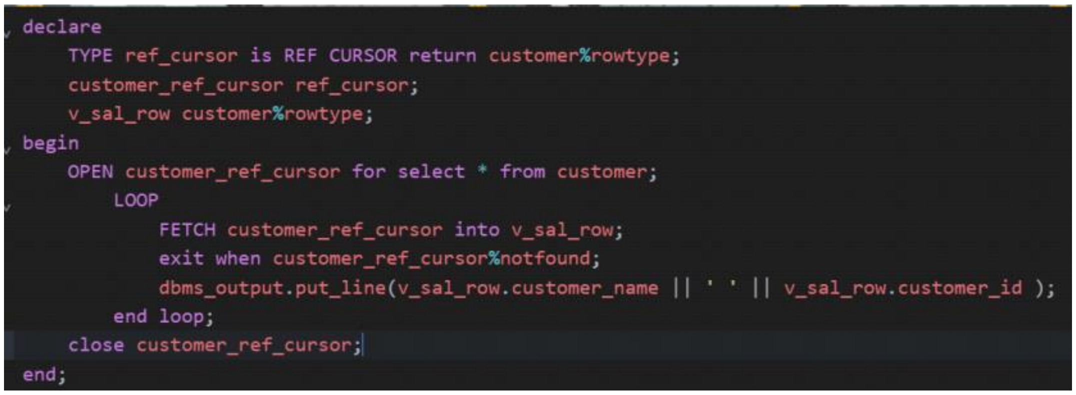
```
declare
        TYPE ref_cursor is REF CURSOR;
        customer_ref_cursor ref_cursor;
        v_sal customer.salary%type;
begin
   OPEN customer_ref_cursor for select salary from customer;
           LOOP
         FETCH customer_ref_cursor into v_sal;
                      exit when customer_ref_cursor%notfound;
                      dbms_output.put_line(v_sal);
           end loop;
        close customer_ref_cursor;
end;
```



```
declare
    TYPE ref_cursor is REF CURSOR;
    customer_ref_cursor ref_cursor;
    v_sal customer.salary%type;
begin
    OPEN customer_ref_cursor for select salary from customer;
        LOOP
            FETCH customer_ref_cursor into v_sal;
            exit when customer_ref_cursor%notfound;
            dbms_output.put_line(v_sal);
        end loop;
    close customer_ref_cursor;
end;
```

Strong REF Cursor - Any REF cursor which has a fixed return type

```
declare
        TYPE ref_cursor is REF CURSOR return customer%rowtype;
        customer_ref_cursor ref_cursor;
        v_sal_row customer%rowtype;
begin
   OPEN customer_ref_cursor for select * from customer;
           LOOP
         FETCH customer_ref_cursor into v_sal_row;
                      exit when customer_ref_cursor%notfound;
                      dbms_output.put_line(v_sal_row.customer_name || ' ' || v_sal_row.customer_id );
           end loop;
        close customer_ref_cursor;
end;
```



```
declare
    TYPE ref_cursor is REF CURSOR return customer%rowtype;
    customer_ref_cursor ref_cursor;
    v_sal_row customer%rowtype;
begin
    OPEN customer_ref_cursor for select * from customer;
        LOOP
            FETCH customer_ref_cursor into v_sal_row;
            exit when customer_ref_cursor%notfound;
            dbms_output.put_line(v_sal_row.customer_name || ' ' || v_sal_row.customer_id );
        end loop;
    close customer_ref_cursor;
end;
```

--write a PL/SQL block to implement a loop that fetches all employees and prints their names and salaries until a salary exceeds 100,000?

```
declare
  cursor cur is select * from employees;
begin
  for rec in cur loop
  dbms_output.put_line('Name of the employee ' || rec.first_name ||' and Salary is ' ||rec.salary);
      IF rec.salary > 100000 THEN
        DBMS_OUTPUT.PUT_LINE('Salary exceeds 100,000, exiting loop.');
        EXIT;
      END IF;

  end loop;
end;
```

--Fetch Employee Details Using a Cursor

```
declare

  cursor cur is select employee_id,first_name,salary from employees;
  v_name employees.first_name%type;
  v_sal employees.salary%type;
  v_emp_id employees.employee_id%type;
begin
  open cur;
  loop
  fetch cur into v_emp_id, v_name,v_sal;
  dbms_output.put_line('Employees Details are '|| ' Employee id: '|| v_emp_id||' Name of the emoloyee '||v_name||' Salary of the employee '||v_sal);
  exit when cur%notfound;
  end loop;
  close cur;
end;
```

--Cursor with FOR LOOP

```
begin
  for rec in (select * from employees ) loop
  dbms_output.put_line('Details are '|| rec.first_name ||' '|| rec.last_name || ' '||rec.salary );
  end loop;
end;
```

--Using a Cursor to Update Records

```
declare
  cursor curs is select employee_id,salary from employees where salary >8000;
  v_employee_id employees.employee_id%TYPE;
  v_salary employees.salary%TYPE;
begin
  OPEN CURS;
  LOOP
  FETCH CURS INTO v_employee_id, v_salary;
  EXIT WHEN CURS%NOTFOUND;
  --UPDATE SALARY
  update employees set salary = v_salary + 500 where employee_id = v_employee_id;
  DBMS_OUTPUT.PUT_LINE ('UPDATED SALARY '|| v_employee_id || v_Salary );
  END LOOP;
  CLOSE CURS;
  commit;
end;
select * from employees where salary >8000;
```

-- cursor with parameters

```
DECLARE
 -- Declare a cursor with a parameter to fetch employees by department
 CURSOR emp_cursor(p_dept_id NUMBER) IS
  SELECT employee_id, first_name, salary
  FROM employees
  WHERE department_id = p_dept_id;

 -- Declare variables to hold the fetched values
 v_employee_id employees.employee_id%TYPE;
 v_first_name employees.first_name%TYPE;
 v_salary employees.salary%TYPE;
BEGIN
 -- Open the cursor and pass the department ID as a parameter
```

```
  OPEN emp_cursor(10);  -- Fetch employees from department 10

  -- Fetch and display the results
  LOOP
    FETCH emp_cursor INTO v_employee_id, v_first_name, v_salary;
    EXIT WHEN emp_cursor%NOTFOUND;  -- Exit when no more rows are fetched
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_employee_id || ', Name: ' || v_first_name || ', Salary: ' || v_salary);
  END LOOP;

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

--Cursor with Bulk Collect Example
--Write a PL/SQL block that uses a cursor with a BULK COLLECT to fetch multiple employee records at once and store them into collections for processing.

```
DECLARE
  -- Declare the cursor to fetch employee details
  CURSOR emp_cursor IS
    SELECT employee_id, first_name, salary
    FROM employees;

  -- Declare collections to hold the fetched data
  TYPE emp_id_array IS TABLE OF employees.employee_id%TYPE;
  TYPE emp_name_array IS TABLE OF employees.first_name%TYPE;
  TYPE emp_salary_array IS TABLE OF employees.salary%TYPE;

  emp_ids emp_id_array;
  emp_names emp_name_array;
  emp_salaries emp_salary_array;
BEGIN
  -- Use BULK COLLECT to fetch the data into collections
  OPEN emp_cursor;
  FETCH emp_cursor BULK COLLECT INTO emp_ids, emp_names, emp_salaries;
  CLOSE emp_cursor;

  -- Display the fetched data using the collections
  FOR i IN 1..emp_ids.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_ids(i) || ', Name: ' || emp_names(i) || ', Salary: ' || emp_salaries(i));
  END LOOP;
END;
/
```

Procedure

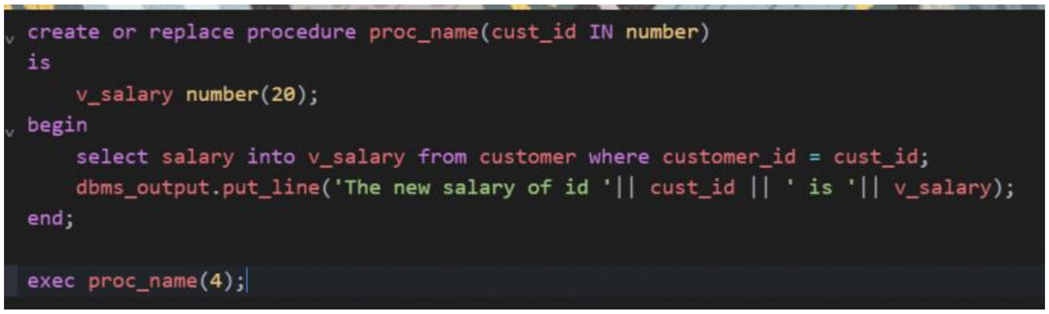
Thursday, January 9, 2025 9:15 PM

PL/SQL procedure is a named block stored as a schema object in the Oracle Database.

IN
An IN parameter is read-only. We can reference an IN parameter inside a procedure, but we cannot change its value. Oracle uses IN as the default mode.
OUT - An OUT parameter is writable. we set a returned value for the OUT parameter and return it to the calling program. Note that a procedure ignores the value that you supply for an OUT parameter.
INOUT
An INOUT parameter is both readable and writable. The procedure can be read and modified.

Simple Procedure -

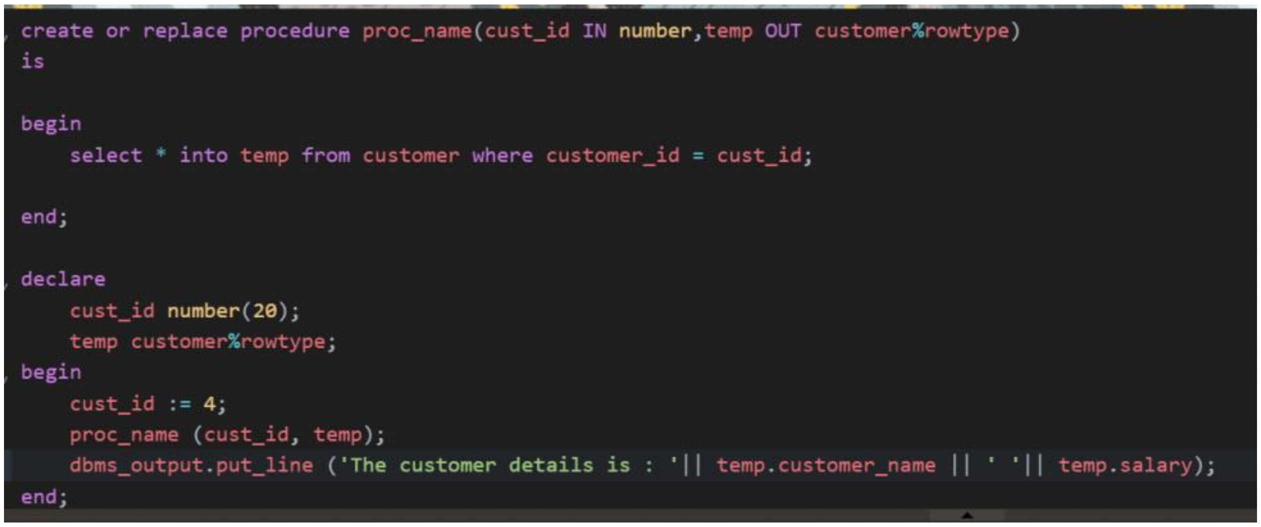
```
create or replace procedure proc_name(cust_id IN number)
is
   v_salary number(20);
begin
   select salary into v_salary from customer where customer_id = cust_id;
        dbms_output.put_line('The new salary of id '|| cust_id || ' is '|| v_salary);
end;

exec proc_name(4);
```



```
create or replace procedure proc_name(cust_id IN number)
is
    v_salary number(20);
begin
    select salary into v_salary from customer where customer_id = cust_id;
    dbms_output.put_line('The new salary of id '|| cust_id || ' is '|| v_salary);
end;

exec proc_name(4);
```

Code 2 -

```
create or replace procedure proc_name(cust_id IN number,temp OUT customer%rowtype)
is

begin
   select * into temp from customer where customer_id = cust_id;

end;

declare
        cust_id number(20);
        temp customer%rowtype;
begin
   cust_id := 4;
        proc_name (cust_id, temp);
        dbms_output.put_line ('The customer details is : '|| temp.customer_name || ' '|| temp.salary);
end;
```



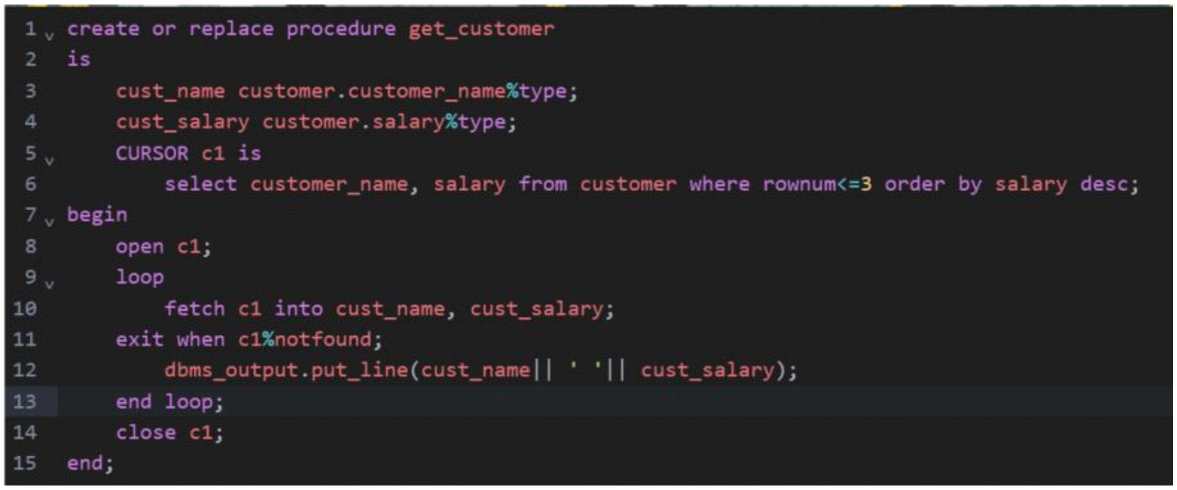
```
create or replace procedure proc_name(cust_id IN number,temp OUT customer%rowtype)
is

begin
    select * into temp from customer where customer_id = cust_id;

end;

declare
    cust_id number(20);
    temp customer%rowtype;
begin
    cust_id := 4;
    proc_name (cust_id, temp);
    dbms_output.put_line ('The customer details is : '|| temp.customer_name || ' '|| temp.salary);
end;
```

*-- Select * from all_procedures where owner = 'HR';*
*-- select * from user_procedures;*
-- select text from all_source;

PROCEDURE with CURSOR

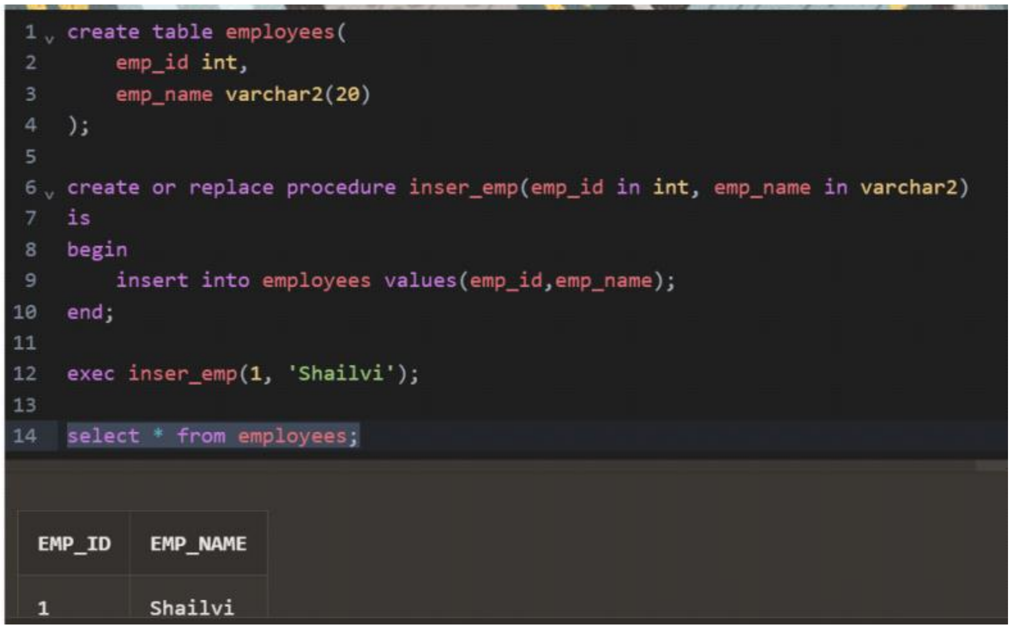
```
create or replace procedure get_customer
is
        cust_name customer.customer_name%type;
        cust_salary customer.salary%type;
        CURSOR c1 is
      select customer_name, salary from customer where rownum<=3 order by salary desc;
begin
        open c1;
   loop
           fetch c1 into cust_name, cust_salary;
        exit when c1%notfound;
           dbms_output.put_line(cust_name|| ' '|| cust_salary);
   end loop;
   close c1;
end;
```

exec get_customer;



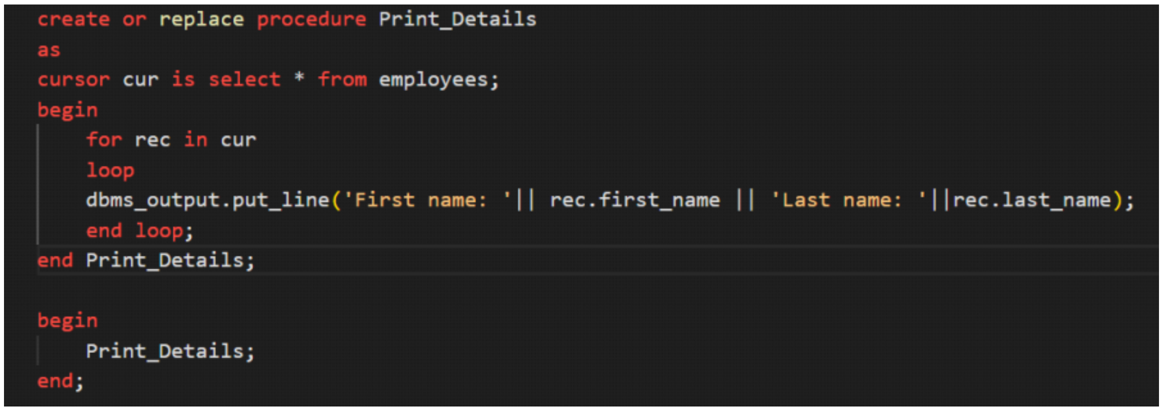
```
create or replace procedure get_customer
is
    cust_name customer.customer_name%type;
    cust_salary customer.salary%type;
    CURSOR c1 is
        select customer_name, salary from customer where rownum<=3 order by salary desc;
begin
    open c1;
    loop
        fetch c1 into cust_name, cust_salary;
    exit when c1%notfound;
        dbms_output.put_line(cust_name|| ' '|| cust_salary);
    end loop;
    close c1;
end;
```

Code 3 - insert statement in PLSQL procedure



```
create table employees(
    emp_id int,
    emp_name varchar2(20)
);

create or replace procedure inser_emp(emp_id in int, emp_name in varchar2)
is
begin
    insert into employees values(emp_id,emp_name);
end;

exec inser_emp(1, 'Shailvi');

select * from employees;
```

EMP_ID	EMP_NAME
1	Shailvi



```
create or replace procedure Print_Details
as
cursor cur is select * from employees;
begin
    for rec in cur
    loop
    dbms_output.put_line('First name: '|| rec.first_name || 'Last name: '||rec.last_name);
    end loop;
end Print_Details;

begin
    Print_Details;
end;
```

--How would you create a stored procedure to calculate the total salary of employees in a specific department?

```
CREATE OR REPLACE PROCEDURE get_salary1(dept_id in number, total_salary OUT number)
is
begin
  SELECT SUM(salary) into total_Salary from employees where department_ID=dept_id;
    DBMS_OUTPUT.PUT_LINE('Total Salary for Department ' || dept_id || ' is: ' || total_salary);
end get_salary1;

declare
P_Salary number;
begin
get_salary1(100,P_Salary);
DBMS_OUTPUT.PUT_LINE('Total Salary for Department ' || P_Salary);
end;
```

Functions

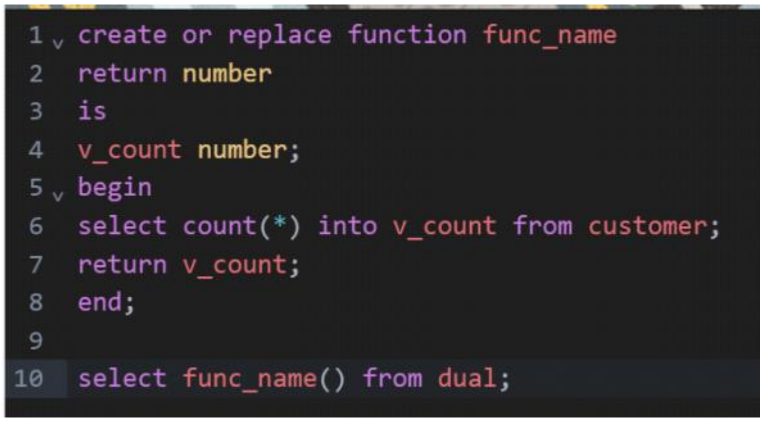
Friday, January 10, 2025 4:31 PM

Function	Procedure
Functions always return a value after the execution of queries.	**The procedure can return a value using “IN OUT” and “OUT” arguments.**
In SQL, those functions having a DML statement can not be called from SQL statements. But autonomous transaction functions can be called from SQL queries.	A procedure can not be called using SQL queries.
Each and every time functions are compiled they provide output according to the given input.	Procedures are compiled only once but they can be called many times as needed without being compiled each time.
A Function can not return multiple result sets.	A procedure is able to return multiple result sets.
The function can be called using Stored Procedure.	While procedures cannot be called from function.
A function used only to read data.	**A procedure can be used to read and modify data.**
The return statement of a function returns the control and function’s result value to the calling program.	While the return statement of the procedure returns control to the calling program, it can not return the result value.
The function does not support try-catch blocks.	Procedure supports try-catch blocks for error handling.
A function can be operated in the SELECT statement.	While it can’t be operated in the SELECT statement.
Functions do not permit transaction management.	It allows transaction management.
In functions, we can use only a table variable. Temporary tables can not be created in function.	In procedures, we can use temporary tables or table variables to store temporary data.

Complexity - functions for value retrieval

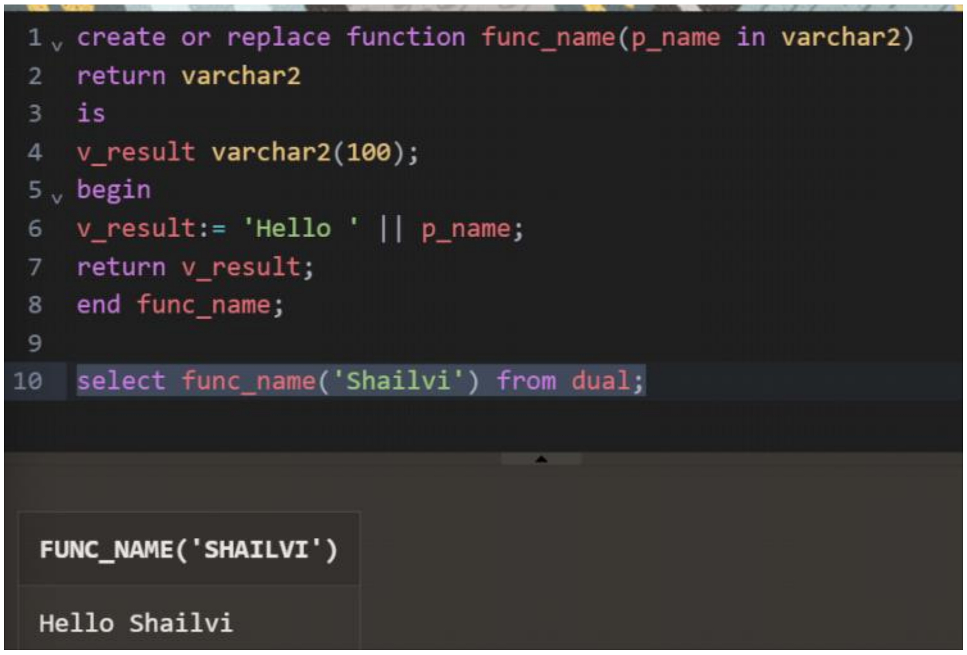
From <https://www.geeksforgeeks.org/difference-between-function-and-procedure/>

```
create or replace function func_name
return number
is
v_count number;
begin
select count(*) into v_count from customer;
return v_count;
end;

select func_name() from dual;
```



```
create or replace function func_name
return number
is
v_count number;
begin
select count(*) into v_count from customer;
return v_count;
end;

select func_name() from dual;
```

Code -2



```
create or replace function func_name(p_name in varchar2)
return varchar2
is
v_result varchar2(100);
begin
v_result:= 'Hello ' || p_name;
return v_result;
end func_name;

select func_name('Shailvi') from dual;
```

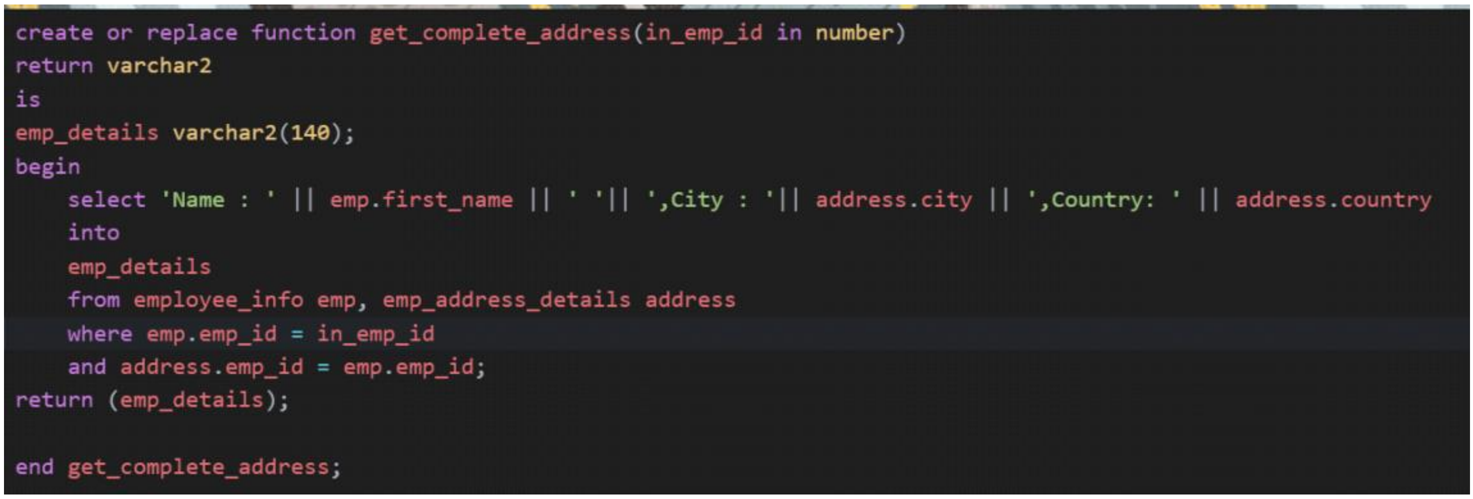
```
FUNC_NAME('SHAILVI')
Hello Shailvi
```

Code 3 -

```
create table employee_info
(
   emp_id number(5) primary key,
   first_name varchar2(20),
   last_name varchar2(20)
);

create table emp_address_details(
   emp_add_id number(5) primary key,
   emp_id number(5) references employee_info(emp_id),
   city varchar2(15),
   state varchar2(15),
   country varchar2(20),
   ZIP_code varchar2(10)
);

insert into employee_info values (10,'Rakesh','Sharma');
insert into employee_info values (20,'John','Faula');

insert into emp_address_details values (101,10,'Vegas','Nevada', 'US','88902');
insert into emp_address_details values (102,20,'Carson','Nevada', 'US','99902');
```

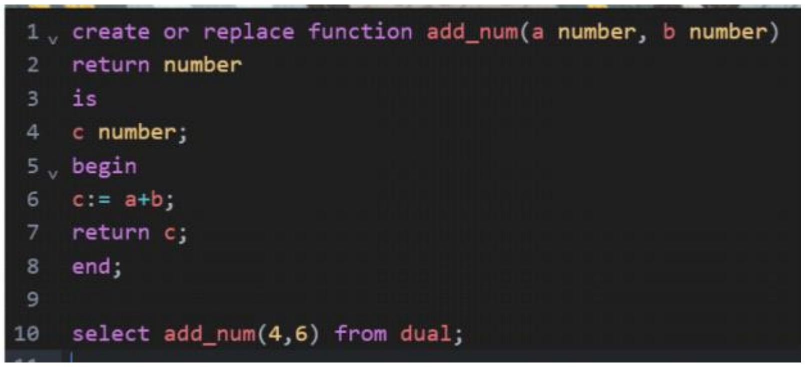
Create a function to get_complete_address

```
create or replace function get_complete_address(in_emp_id in number)
return varchar2
is
emp_details varchar2(140);
begin
        select 'Name : ' || emp.first_name || ' '|| ',City : '|| address.city || ',Country: ' || address.country
        into
        emp_details
        from employee_info emp, emp_address_details address
        where emp.emp_id = in_emp_id
        and address.emp_id = emp.emp_id;
return (emp_details);

end get_complete_address;


select emp_id,first_name, get_complete_address(emp_id) address from employee_info;
```



```
create or replace function get_complete_address(in_emp_id in number)
return varchar2
is
emp_details varchar2(140);
begin
    select 'Name : ' || emp.first_name || ' '|| ',City : '|| address.city || ',Country: ' || address.country
    into
    emp_details
    from employee_info emp, emp_address_details address
    where emp.emp_id = in_emp_id
    and address.emp_id = emp.emp_id;
return (emp_details);

end get_complete_address;
```

Code 3: add 2 numbers



```
create or replace function add_num(a number, b number)
return number
is
c number;
begin
c:= a+b;
return c;
end;

select add_num(4,6) from dual;
```

--How would you write a function to return the total number of employees in a department?

```
create or replace function fetch_no_of_employees(dept_id in number)
return number
is
emp_count number;
begin
   select count(*) into emp_count from employees where department_id = dept_id;
   dbms_output.put_line('Total number of employees is '|| emp_count);
  RETURN emp_count;
EXCEPTION
```

```
   WHEN NO_DATA_FOUND THEN
      RETURN 0;  -- Return 0 if no employees are found in the department
   WHEN OTHERS THEN
      RETURN NULL;  -- Return NULL if there is an error
END;
select fetch_no_of_employees(100) from dual;
```

--Write a PL/SQL function that takes an employee's ID and returns the employee's full name (concatenation of first and last name).

```
create or replace function emp_full_name(emp_id in number)
return varchar2
is
   v_name employees.first_name%type;
begin
   select first_name || ' ' || last_name into v_name from employees where employee_id = emp_id;
   dbms_output.put_line(v_name);
   return v_name;
end emp_full_name;

select emp_full_name(100) from dual;
```

--Write a function to calculate the average salary of employees in a given department.

```
CREATE OR REPLACE FUNCTION avg_sal(dept_id in number)
return number
is
emp_count number;
emp_sal_sum number;
avg_salary number;
begin
   select count(*), sum(salary) into emp_count, emp_sal_sum from employees where department_id = dept_id;
   avg_salary := emp_sal_sum/emp_count;
   dbms_output.put_line(avg_salary);
   return avg_salary;
end;

select avg_sal(100) from dual;
```

Packages

Friday, January 10, 2025 9:41 PM

PL/SQL packages are a way to organize and encapsulate related **procedures**, **functions**, **variables**, **triggers**, and other PL/SQL items into a single item.
A PL/SQL package is a collection of related **Procedures**, **Functions**, **Variables**, and other elements that are grouped for **Modularity** and **Reusability**.
Typically, a package has a specification and a body. A package specification is mandatory while the package body can be required or optional, depending on the package specification.

Example of Package Specification:

```
CREATE OR REPLACE PACKAGE my_package AS
   PROCEDURE my_procedure(p_param1 NUMBER);
   FUNCTION calculate_sum(x NUMBER, y NUMBER) RETURN NUMBER;
   -- Other declarations...
END my_package;
```

Example of Package Body:

```
CREATE OR REPLACE PACKAGE BODY my_package AS
   PROCEDURE my_procedure(p_param1 NUMBER) IS
   BEGIN
      -- Implementation code...
   END my_procedure;
FUNCTION calculate_sum(x NUMBER, y NUMBER) RETURN NUMBER IS
   BEGIN
      -- Implementation code...
   END calculate_sum;
   -- Other implementation details...
END my_package;
```

Using Oracle PL/SQL Packages in Code

```
DECLARE
   result NUMBER;
BEGIN
   -- Call a procedure from the package
   my_package.my_procedure(42);
-- Call a function from the package
   result := my_package.calculate_sum(10, 20);

-- Other code...
END;
```

```
-- Enable the display of server output
SET SERVEROUTPUT ON;
-- Create a PL/SQL package specification
CREATE OR REPLACE PACKAGE math_operations AS
  -- Procedure to add two numbers with an output parameter
  PROCEDURE add_numbers(x NUMBER, y NUMBER, result OUT NUMBER);
  -- Function to multiply two numbers
  FUNCTION multiply_numbers(x NUMBER, y NUMBER) RETURN NUMBER;
END math_operations;
/
-- Create the body of the math_operations package
CREATE OR REPLACE PACKAGE BODY math_operations AS
  -- Implementation of the add_numbers procedure
  PROCEDURE add_numbers(x NUMBER, y NUMBER, result OUT NUMBER) IS
  BEGIN
    result := x + y;
  END add_numbers;
  -- Implementation of the multiply_numbers function
  FUNCTION multiply_numbers(x NUMBER, y NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN x * y;
  END multiply_numbers;
END math_operations;
/
-- PL/SQL block to test the math_operations package
DECLARE
  -- Declare variables to store results
  sum_result NUMBER;
  product_result NUMBER;
BEGIN
  -- Call the procedure and pass output parameter
  math_operations.add_numbers(5, 7, sum_result);
  -- Display the result of the add_numbers procedure
  DBMS_OUTPUT.PUT_LINE('Sum Result: ' || sum_result);
  -- Call the function and retrieve the result
  product_result := math_operations.multiply_numbers(3, 4);
  -- Display the result of the multiply_numbers function
  DBMS_OUTPUT.PUT_LINE('Product Result: ' || product_result);
END;
/
```

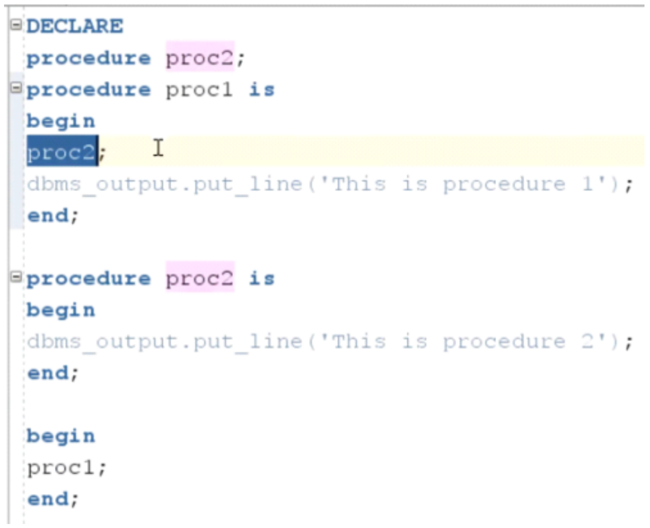
Code -1

```
create or replace package first_package
is
procedure greet(p_name in varchar2);
function hello_func(p_name in varchar2) return varchar2;
end;


create or replace package body first_package is
procedure greet(p_name in varchar2)
is
begin
        dbms_output.put_line('Hello '|| p_name);
end greet;
function hello_func(p_name in varchar2)
return varchar2
is
v_result varchar2(100);
begin
        v_result:= 'Hello '||p_name;
return v_result;
end hello_func;
end;

-- procedure execution
execute first_package.greet('PLSQL procedure');
--function execution
select first_package.hello_func('PLSQL Function') from dual;
```

Code -2

```
create or replace package pkg_overload_add_numbers
is
        procedure add_num(a number, b number);
        procedure add_num(a number, b number, c number);
end;

create or replace package body pkg_overload_add_numbers
is
        procedure add_num(a number, b number)
                is
                begin
                        dbms_output.put_line('Sum of two numbers is: '|| to_char(a+b));
                end;
        procedure add_num(a number, b number, c number)
                is
                begin
                        dbms_output.put_line('Sum of two numbers is: '|| to_char(a+b+c));
                end;
end;

exec pkg_overload_add_numbers.add_num(4,6,5);
```

Code 3
Forward declarations/reference -



```
DECLARE
procedure proc2;
procedure proc1 is
begin
proc2;
dbms_output.put_line('This is procedure 1');
end;

procedure proc2 is
begin
dbms_output.put_line('This is procedure 2');
end;

begin
proc1;
end;
```

-- Create a package to fetch employee details by employee ID (using SELECT query) and update their salary.
--Ask the candidate to design a package that includes a procedure for fetching employee details (using SELECT) and another procedure for updating employee salary.

```
create or replace package fetching_details
as
procedure fetch_details (emp_id in number, p_name out varchar2, p_salary out number);
procedure update_salary (emp_id in number, p_slary out number);
end fetching_details;

-- creating body of the pkg
create or replace package body fetching_details
as
procedure fetch_details (emp_id in number, p_name out varchar2, p_salary out number)
is
begin
select first_name ||' '|| last_name into p_name,p_salary from employees where employee_id = emp_id;
end fetch_details;

create or replace package body fetching_details
```

```
as
procedure update_salary (emp_id in number, p_salary out number)
is
begin
   update salary set salary = p_salary where employee_id = emp_id;
end update_salary;
end fetching_details;
/
-- Testing 1
declare
v_name varchar2;
v_Sal  number;
begin
fetching_details.fetch_details(100,v_name,v_sal);
  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_sal);
 end;

-- Testing 2
declare
v_Sal  number;
begin
fetching_details.update_salary(100,v_sal);
 DBMS_OUTPUT.PUT_LINE('Updated Salary' ||v_Sal);
end;
```

Triggers

Saturday, January 11, 2025 12:24 PM

Oracle Trigger :- Named PL/SQL blocks which are stored in the database and executed automatically when a triggering event takes place.

Events -	**Types -**	**Uses -**
DML, DDL, System, User event	◆ DML Triggers ◆ DDL Triggers (Auditing changes) ◆ System/ Database Event Triggers (e.g. log off/log on) ◆ Instead-of Triggers ◆ Compound Triggers	◆ Enforce business rules ◆ Gain strong control over the security ◆ Collect statistical Information ◆ Automatically generate values ◆ Prevent invalid Transactions

The act of executing a trigger is also known as firing a trigger. We say that the trigger is fired.

statement-trigger fires once regardless of the number of rows affected by the triggering event.

Row Level Triggers	**Statement Level Triggers**
Row level triggers executes once for each and every row in the transaction.	Statement level triggers executes only once for each single transaction.
Specifically used for data auditing purpose.	Used for enforcing all additional security on the transactions performed on the table.
"FOR EACH ROW" clause is present in CREATE TRIGGER command.	"FOR EACH STATEMENT" clause is omitted in CREATE TRIGGER command.
Example: If 1500 rows are to be inserted into a table, the row level trigger would execute 1500 times.	Example: If 1500 rows are to be inserted into a table, the statement level trigger would execute only once

Code 1 - DML triggers using insert

```
select * from customer;

create table customer_details as select * from customer;

create table customer_bkp_trig as select * from customer where 1=2;

alter table customer_bkp_trig add date_of_deletion date;

alter table customer_bkp_trig add who_deleted varchar2(30);

select * from customer_bkp_trig;
select * from customer_details;

create or replace trigger customer_trigger
before delete on customer_details
for each row
begin
insert into customer_bkp_trig values (
   :old.customer_id, :old.customer_name, :old.address, :old.city, :old.country,:old.salary,sysdate,user);
end;

delete from customer_details where customer_name='Arun';
```

An INSTEAD OF trigger is a trigger that allows you to update data in tables via their view which cannot be modified directly through DML statements.

From <https://www.oracletutorial.com/plsql-tutorial/oracle-instead-of-triggers/>

In Oracle, you can create an INSTEAD OF trigger for a view only. You cannot create an INSTEAD OF trigger for a table.

From <https://www.oracletutorial.com/plsql-tutorial/oracle-instead-of-triggers/>

disable a trigger for testing and troubleshooting purposes. To disable a trigger, we use the ALTER TRIGGER DISABLE statement:
ALTER TRIGGER trigger_name DISABLE/disable all/enable/enable all;

From <https://www.oracletutorial.com/plsql-tutorial/oracle-disable-triggers/>

```
DROP TRIGGER [schema_name.]trigger_name;
DROP TRIGGER IF EXISTS trigger_name;
```

From <https://www.oracletutorial.com/plsql-tutorial/oracle-drop-trigger/>

Mutating Table Error in Oracle

From <https://www.oracletutorial.com/plsql-tutorial/mutating-table-error-in-oracle/>

When a table is mutating, it is changing. If the change is taking place and you try to make another change in the middle of the first change, Oracle will issue a mutating table error with the error code ORA-04091.

From <https://www.oracletutorial.com/plsql-tutorial/mutating-table-error-in-oracle/>

Create a Trigger to Track Changes (Audit) on the employees Table

```
CREATE TABLE salary_log1 (
   who_did_it VARCHAR2(50),
   when_did_it TIMESTAMP,
   old_salary NUMBER,
   new_salary NUMBER,
   emp_id NUMBER
);
/
CREATE OR REPLACE TRIGGER SALTRIG
AFTER UPDATE OF salary
ON employees
FOR EACH ROW
BEGIN
   INSERT INTO salary_log1 (
      who_did_it,
      when_did_it,
      old_salary,
      new_salary,
      emp_id
   )
   VALUES (
      USER,                    -- Captures the current user
      SYSDATE,                 -- Captures the current timestamp
      :OLD.salary,             -- Old salary value
      :NEW.salary,             -- New salary value
      :NEW.employee_id         -- Employee ID
   );
END;
/

UPDATE employees
SET salary = salary+500
WHERE employee_id = 100;

select * from salary_log1;
```

Records/Collections

Saturday, January 11, 2025 8:37 PM

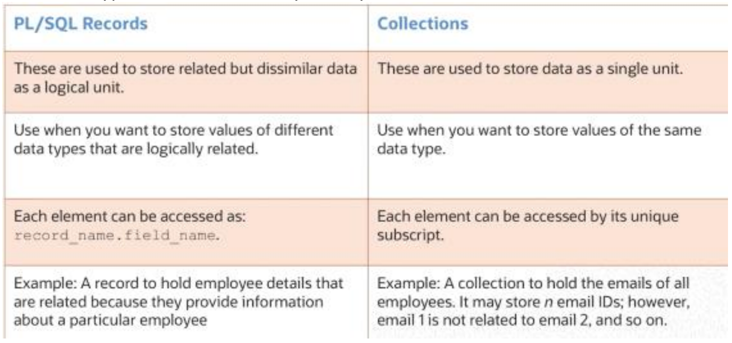
Composite Data Types in PL/SQL

Definition: Composite data types in PL/SQL allow grouping multiple values into a single variable. They help in handling structured data efficiently. The two main types are:

1. Records – Store multiple related fields of different data types as a single entity.

2. Collections – Store multiple values of the same data type. These include:

Compostive data types = Records , collection
Collection types - associative array, varray, nested tables



PL/SQL Records	Collections
These are used to store related but dissimilar data as a logical unit.	These are used to store data as a single unit.
Use when you want to store values of different data types that are logically related.	Use when you want to store values of the same data type.
Each element can be accessed as: `record_name.field_name`.	Each element can be accessed by its unique subscript.
Example: A record to hold employee details that are related because they provide information about a particular employee	Example: A collection to hold the emails of all employees. It may store *n* email IDs; however, email 1 is not related to email 2, and so on.

A PL/SQL record is a composite data structure that consists of multiple fields; each has its own value

Types of Records - Table based, Cursor based, Programmer defined

Table Based record - when we have to create record based on existing ones
DECLARE record_name table_name%ROWTYPE;

Cursor Based record - when we have to create record based on existing ones
DECLARE record_name cursor_name%ROWTYPE;

Programmer-defined record - when we have to want to create a record whose structure is not based on the existing ones

To declare a programmer-defined record, you use the following steps:
1. Define a record type that contains the structure you want in your record.
2. Declare a record based on the record type.

Nesting records are a powerful way to structure your program data and hide complexity in your code.

*create table bkp as select * from customer where 2=3; - no data there due to where*

*select * from customer;*
*select * from bkp;*

Code1 -

```
declare
v_cust_name varchar2(30);
v_salary number(10);
begin
   select customer_name,salary into v_cust_name, v_salary from customer
   where customer_id =4;
dbms_output.put_line(v_cust_name||v_salary);
end;
```

We need to write correct datatype which is in customer table so for overcome this we need to create type

```
declare
v_cust_name customer.customer_name%type;
```

```
v_salary customer.salary%type;
begin
   select customer_name,salary into v_cust_name, v_salary from customer
   where customer_id =1;
dbms_output.put_line(v_cust_name||v_salary);
end;
```

Doubt -

```
declare
        v_cust_rec customer%rowtype;
begin
   select * into v_cust_rec from customer where customer_id=5;
        dbms_output.put_line(v_cust_rec.customer_name);
end;
```

--Record : TYPE - want to create our own complex data type by using columns of table

```
declare
        type cust1_record_type is record(
   customer_id number(10),
   customer_name varchar2(100) );
        cust_rec cust1_record_type;
begin
   cust_rec.customer_id :=19;
        cust_rec.customer_name := 'John';
        dbms_output.put_line(cust_rec.customer_name|| ' '|| cust_rec.customer_id);
end;
```

Code 3-

```
declare
        type customer_rec_type is record (
   customer_id number(10),
   salary number(9)
   );
        customer_rec customer_rec_type;
begin
           select customer_id,salary into customer_rec from customer where customer_id=2;
                dbms_output.put_line(customer_rec.customer_id || ' '||customer_rec.salary);
end;
```

Code 4 : Collections - VARRAY

```
declare
        type v_array_type is varray(4) of varchar2(30);
        address v_array_type:= v_array_type(null,null,null,null);
begin
        address(1):= 'Katra';
        address(2):= 'Bazar';
        address(3):= 'Chhapra';
        address(4):= 'Bihar';
        dbms_output.put_line('The city of Shailvi is: '||address(3));
end;
```

If you try to access address(8) then will get an error of Subscript outside of limit

Code 5 :

```
declare
        type v_array_type is varray(4) of varchar2(30);
        address v_array_type:= v_array_type(null,null,null,null);
begin
        address(1):= 'Katra';
        address(2):= 'Bazar';
        address(3):= 'Chhapra';
        address(4):= 'Bihar';
        dbms_output.put_line(address.limit);
        dbms_output.put_line(address.count);
   dbms_output.put_line(address.first);
        dbms_output.put_line(address.last);
        dbms_output.put_line(address.prior(2));
        dbms_output.put_line(address.next(2));
--address.extend(3); -- extend 3 - will give error because size is 4
--dbms_output.put_line(address.count);
```

```
--address.trim(); --last element trimmed
--address.delete(); -- all delete - we can't delete by indexing we need to make nested table to use index
end;
```

Code 5 : Nested Tables

```
declare
        type v_nested_table_type is table of varchar2(40);
        v_color v_nested_table_type:= v_nested_table_type(null,null,null); -- given 3 null
begin
        v_color(1):='Red';
        v_color(2):='Blue';
        v_color(3):='White';
        v_color(4):='Doodo'; -- giving 1 extra values will lead to error - subscript beyond count
        dbms_output.put_line(v_color(3));
end;


declare
        type v_nested_table_type is table of varchar2(40);
        v_color v_nested_table_type:= v_nested_table_type(null,null,null);
begin
        v_color(1):='Red';
        v_color(2):='Blue';
        v_color(3):='White';
        dbms_output.put_line(v_color.count);
        v_color.extend(4);
        v_color(4):='Doodo';
        dbms_output.put_line(v_color(4));
        dbms_output.put_line(v_color.limit);
        dbms_output.put_line(v_color.count);
        dbms_output.put_line(v_color.first); --index of 1st element
        dbms_output.put_line(v_color.last);

--- in nested table we can delete by defining index number and that index wont print in o/p get an error
--
if v_color.exists(2) then
         dbms_output.put_line(v_color(2));
        end if;
        v_color.delete(2);
        dbms_output.put_line(v_color(1)||v_color(3));
end;
```

BULK COLLECT

```
create table bulk_table(id number(10));

create table bulk_bind(id number(10));

--make anonymous block
begin
        for i in 1..1000 --plsql block
        loop
                insert into bulk_table values(i); --sql block so taking time for context switching
        end loop;
        commit;
        end;

select count(*) from bulk_table;

declare
        type rt is table of bulk_table%rowtype;
        bulk rt;
begin
        select * bulk collect into bulk from bulk_table;
        forall i in 1..bulk.count
                insert into bulk_bind values bulk(i);
        commit;
end; -- in real time project millions of data it will take less time to execute

select count(*) from bulk_bind;
```

```
declare
        v_cust_name customer.customer_name%type;
        v_salary customer.salary%type;
begin
        select customer_name,salary into v_cust_name,v_salary from customer; --where customer_id=92;
        dbms_output.put_line('First name : '|| v_cust_name);
        dbms_output.put_line('Salary : '||v_salary);
EXCEPTION
        when no_data_found then
                dbms_output.put_line('No data is found for this customer');
        when too_many_rows then
                dbms_output.put_line('Many rows are returned from base table');
end;

--if we want to fetch multiple records we can use BULK COLLECT or CURSOR
-- Cursor please check cursor notes
-- BULK COLLECT - nested table -- to avoid context switching

declare
        type nt_salary_type is table of number(10);
        nt_salary nt_salary_type := nt_salary_type();
begin
        select salary BULK COLLECT INTO nt_salary from customer;
        for i in nt_salary.first..nt_salary.last
        loop
        dbms_output.put_line(nt_salary(i));
        end loop;
end;

-- example on bulk collect

declare
        type customer_t is table of customer%ROWTYPE;
        l_customer customer_t;
begin
           select * BULK COLLECT INTO l_customer from customer;
                dbms_output.put_line('Total count : '||l_customer.count);
        FOR i in 1..l_customer.count
      loop
           dbms_output.put_line(l_customer(i).customer_name);
                END LOOP;
end;
```

Exceptions

Monday, January 13, 2025 11:46 AM

When an exception occurs in the executable section, the execution of the current block stops, and control transfers to the exception handling section.

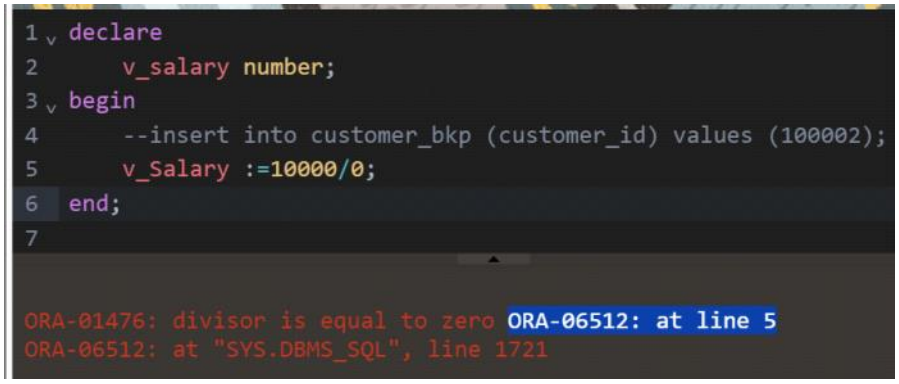
PL/SQL has three exception categories:

- **Internally defined exceptions** are errors that arise from the Oracle Database environment. The runtime system raises the internally defined exceptions automatically. ORA-27102 (**out of memory**) is one example of Internally defined exceptions. Note that Internally defined exceptions **do not have names, but an error code.**
- **Predefined exceptions** are errors that **occur during the execution of the program**. The predefined exceptions are internally defined exceptions that **PL/SQL has given names e.g., NO_DATA_FOUND, TOO_MANY_ROWS.**
- **User-defined exceptions** are custom exceptions defined by users. User-defined exceptions must be raised explicitly.

The **SQLCODE** function accepts no argument and **returns a number code** of the most recent exception.
If the exceptions are internal, SQLCODE returns a negative number except for the NO_DATA_FOUND exception which has the number code +100.
If the exception is user-defined, SQLCODE returns +1 or the number that you associated with the exception via the pragma EXCEPTION_INIT.
The SQLCODE is only usable in the exception handling section. If you use the SQLCODE function outside an exception handler, it always returns zero.

Named Exception -

```
declare
        v_salary number;
begin
        --insert into customer_bkp (customer_id) values (100002);
        v_Salary :=10000/0;
end;
```



```
declare
    v_salary number;
begin
    --insert into customer_bkp (customer_id) values (100002);
    v_Salary :=10000/0;
end;

ORA-01476: divisor is equal to zero ORA-06512: at line 5
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

Using named exception -

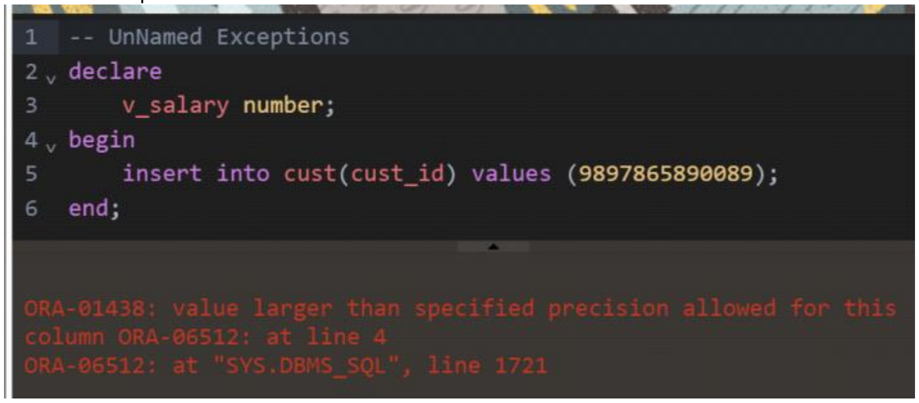
```
declare
        v_salary number;
begin
        --insert into customer_bkp (customer_id) values (100002);
        v_Salary :=10000/0;
EXCEPTION
   when ZERO_DIVIDE then
           dbms_output.put_line('Divisor is equal to zero, Operations is not allowed');
        when others then
      dbms_output.put_line('Exception happened, Check the code');
end;
```

Code 3 -

```
declare
        v_salary number(2);
begin
```

```
        v_salary := 34567;
EXCEPTION
    when value_error then
            dbms_output.put_line('Value error Exception happened, check the code');
        when others then
      dbms_output.put_line('Exception happened, Check the code');
end;
```

Unnamed Exception



```
-- UnNamed Exceptions
declare
    v_salary number;
begin
    insert into cust(cust_id) values (9897865890089);
end;
```

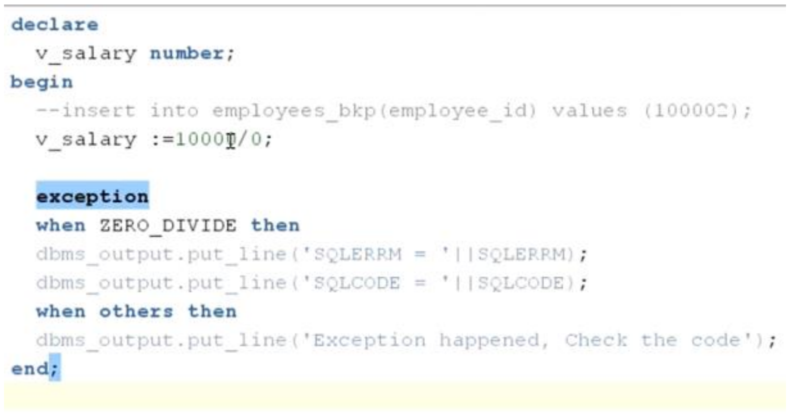
```
ORA-01438: value larger than specified precision allowed for this
column ORA-06512: at line 4
ORA-06512: at "SYS.DBMS_SQL", line 1721
```

To overcome above

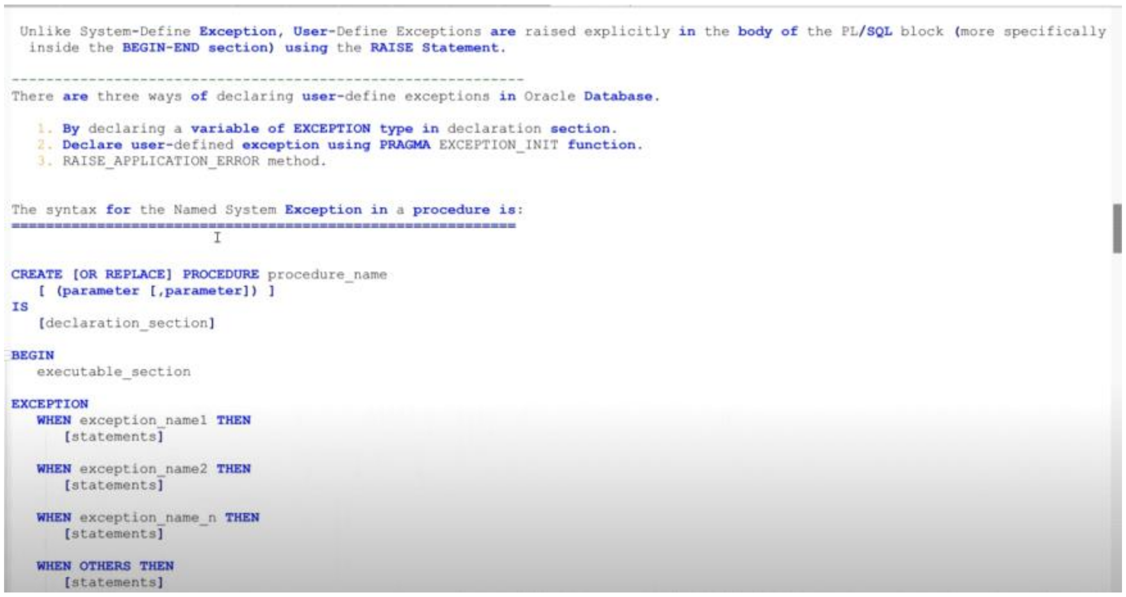
```
-- UnNamed Exceptions

declare
        v_salary number;
        ex_cust_id_value_limit exception;
        PRAGMA EXCEPTION_INIT (ex_cust_id_value_limit,-01438);
begin
        insert into cust(cust_id) values (9897865890089);
        exception
      when ex_cust_id_value_limit then
            dbms_output.put_line('Exception happened,and it has been handled');

end;
```

User- defined exception -

```
-- User Defined Exception
create table emp_excep
(
   emp_id number(6),
   emp_name varchar2(40)
);
alter table emp_excep add primary key (emp_id);

create or replace Procedure add_new_employee
        (employee_id_in In number, first_name_in varchar2)
is
begin
        insert into emp_excep(emp_id,emp_name)
        values (employee_id_in, first_name_in);
        commit;
EXCEPTION
        when dup_val_on_index then
                raise_application_error (-20001,'you are trying to insert a duplicate emp_id');
        when others then
                raise_application_error (-20002,'An error has occurred inserting a employee');
end;

exec add_new_employee(2000,'Arun');
select * from emp_excep;
```



```
declare
  v_salary number;
begin
  --insert into employees_bkp(employee_id) values (100002);
  v_salary :=10000/0;

  exception
  when ZERO_DIVIDE then
  dbms_output.put_line('SQLERRM = '||SQLERRM);
  dbms_output.put_line('SQLCODE = '||SQLCODE);
  when others then
  dbms_output.put_line('Exception happened, Check the code');
end;
```



```
 Unlike System-Define Exception, User-Define Exceptions are raised explicitly in the body of the PL/SQL block (more specifically
  inside the BEGIN-END section) using the RAISE Statement.

-----------------------------------------------------------
There are three ways of declaring user-define exceptions in Oracle Database.

   1. By declaring a variable of EXCEPTION type in declaration section.
   2. Declare user-defined exception using PRAGMA EXCEPTION_INIT function.
   3. RAISE_APPLICATION_ERROR method.


The syntax for the Named System Exception in a procedure is:
============================================================


CREATE [OR REPLACE] PROCEDURE procedure_name
   [ (parameter [,parameter]) ]
IS
   [declaration_section]

BEGIN
   executable_section

EXCEPTION
   WHEN exception_name1 THEN
      [statements]

   WHEN exception_name2 THEN
      [statements]

   WHEN exception_name_n THEN
      [statements]

   WHEN OTHERS THEN
      [statements]
```

--1. Question: Handle an exception when a SELECT query does not return any rows (NO_DATA_FOUND).

```
create or replace procedure exe (emp_id in number)
is
v_name employees.first_name%type;
begin
select first_name || ' ' || last_name into v_name from employees  where employee_id = emp_id;
exception
   when no_data_found then
      dbms_output.put_line('No Data Found');
   when others then
      dbms_output.put_line('Then value is ');
end exe;

begin
   exe(1);
end;
```

--2. Question: Handle TOO_MANY_ROWS exception and display a meaningful message.

```
create or replace procedure wksp ( mng_id in number)
is
v_name employees.first_name%type;
begin
select first_name into v_name from employees where manager_id = mng_id;
exception
when too_many_rows then
   dbms_output.put_line('Too many rows are there');
end wksp;

begin
wksp(108);
```

```
end;
```

--3. **Question: Use RAISE_APPLICATION_ERROR to throw a custom exception.**

```
DECLARE
  v_dept_id NUMBER := 1;  -- Invalid department ID
  v_count   NUMBER;
BEGIN
  -- Check if the department_id exists in the employees table
  SELECT COUNT(*)
  INTO v_count
  FROM employees
  WHERE department_id = v_dept_id;

  IF v_count = 0 THEN
    -- If no rows are found, raise a custom error
    RAISE_APPLICATION_ERROR(-20001, 'Invalid department ID: ' || v_dept_id);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Department ID is valid.');
  END IF;

EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

--4. Question: Handle multiple exceptions in a PLSQL block.

```
DECLARE
  emp_name employees.first_name%TYPE;
BEGIN
  SELECT first_name
  INTO emp_name
  FROM employees
  WHERE department_id = 10;  -- Assuming this might return multiple rows

  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('No employee found in the given department.');
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('There are too many employees in the department.');
END;
```

question

Sunday, January 19, 2025 11:57 AM

Month	**Product**	**Sales**
Jan	Apple	100
Jan	Banana	150
Feb	Apple	200
Feb	Banana	120

You want to pivot this data to get the sales of each product for each month:
SQL allows us to **rotate data from rows to columns**.

```
SELECT *
FROM (
   SELECT Month, Product, Sales
   FROM SalesData
) AS SourceTable
PIVOT (
   SUM(Sales)
   FOR Product IN ([Apple], [Banana])
) AS PivotTable;
```

Result:

Month	**Apple**	**Banana**
Jan	100	150
Feb	200	120

```
SELECT (ColumnNames)
FROM (TableName)
PIVOT
 (
AggregateFunction(ColumnToBeAggregated)
 FOR PivotColumn IN (PivotColumnValues)
 ) AS (Alias) //Alias is a temporary name for a table
```

Basis	**View**	**Table**
Definition	A **view** is a virtual table that derives its data from one or more **base tables** through a **SQL query**.	A **table** is a physical object that stores data in the form of rows and columns.
Dependency	A view depends on underlying **tables** or other views for data retrieval.	A **table** is an independent data object that directly stores information.
Database space	Views do not occupy physical storage space. They only store the query structure.	Tables consume physical space to store data in a database.

Backup table -
let us safely restore the original data if something goes wrong

```
CREATE TABLE Table_Name AS SELECT * FROM Source_Table_Name;
```

create table customer1 as select * from customer;

select * from customer2; - all data will be there

create table customer2 as select * from customer where 1=0; -- no data found

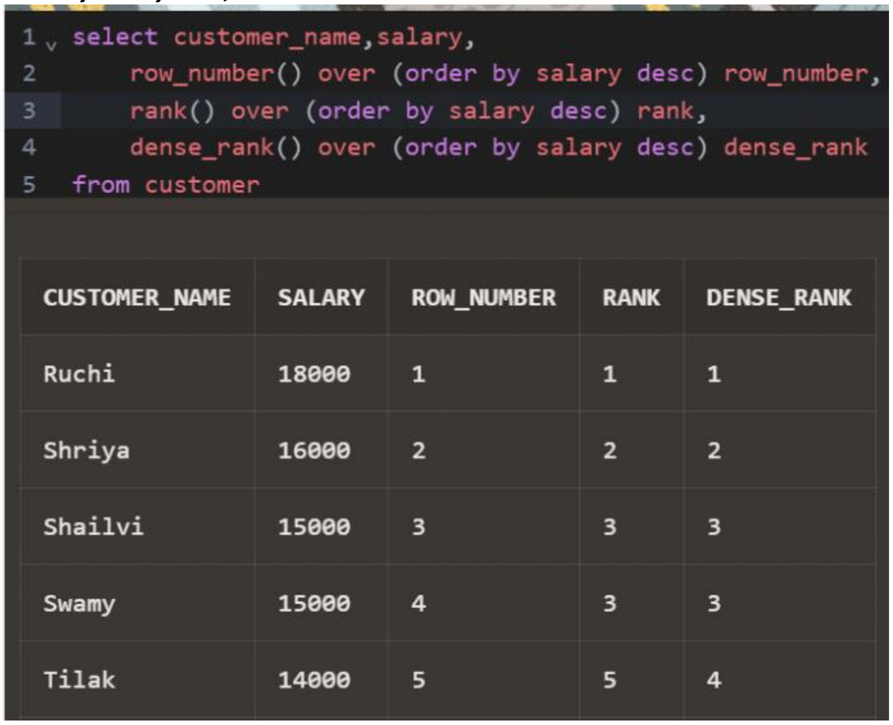
The most commonly used SQL aggregate functions are:

- `MIN()` - returns the smallest value within the selected column
- `MAX()` - returns the largest value within the selected column
- `COUNT()` - returns the number of rows in a set
- `SUM()` - returns the total sum of a numerical column
- `AVG()` - returns the average value of a numerical column

Rank and Dense_rank
 they are windows function/analytic function

Rank - Ties will assign with the same rank, with then next rankings skipped
Dense_rank - Ties will assign with the same rank, but next ranking will be consecutive

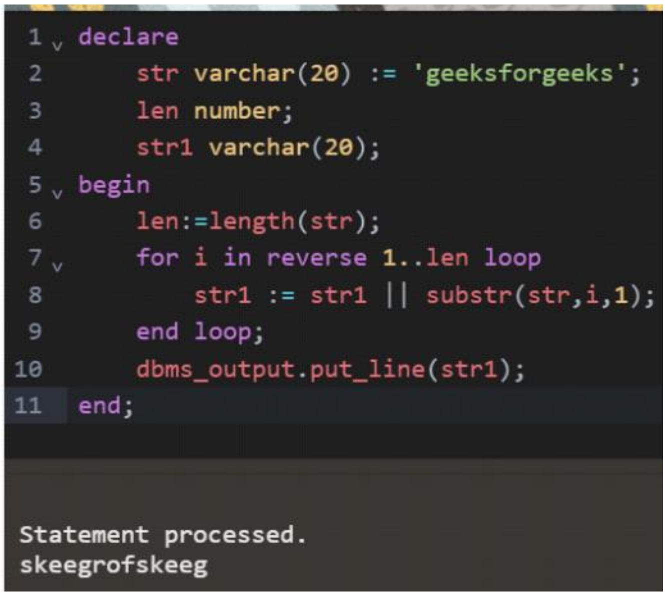
select customer_name,salary,
   row_number() over (order by salary desc) row_number,
   rank() over (order by salary desc) rank,
   dense_rank() over (order by salary desc) dense_rank
from customer
order by salary desc;



```
select customer_name,salary,
    row_number() over (order by salary desc) row_number,
    rank() over (order by salary desc) rank,
    dense_rank() over (order by salary desc) dense_rank
from customer
```

CUSTOMER_NAME	SALARY	ROW_NUMBER	RANK	DENSE_RANK
Ruchi	18000	1	1	1
Shriya	16000	2	2	2
Shailvi	15000	3	3	3
Swamy	15000	4	3	3
Tilak	14000	5	5	4

Reverse a string
SELECT REVERSE(Customer_Name)
FROM Customer;



```
declare
    str varchar(20) := 'geeksforgeeks';
    len number;
    str1 varchar(20);
begin
    len:=length(str);
    for i in reverse 1..len loop
        str1 := str1 || substr(str,i,1);
    end loop;
    dbms_output.put_line(str1);
end;
```

```
Statement processed.
skeegrofskeeg
```

PL/SQL project 1

Tuesday, January 21, 2025 12:38 PM

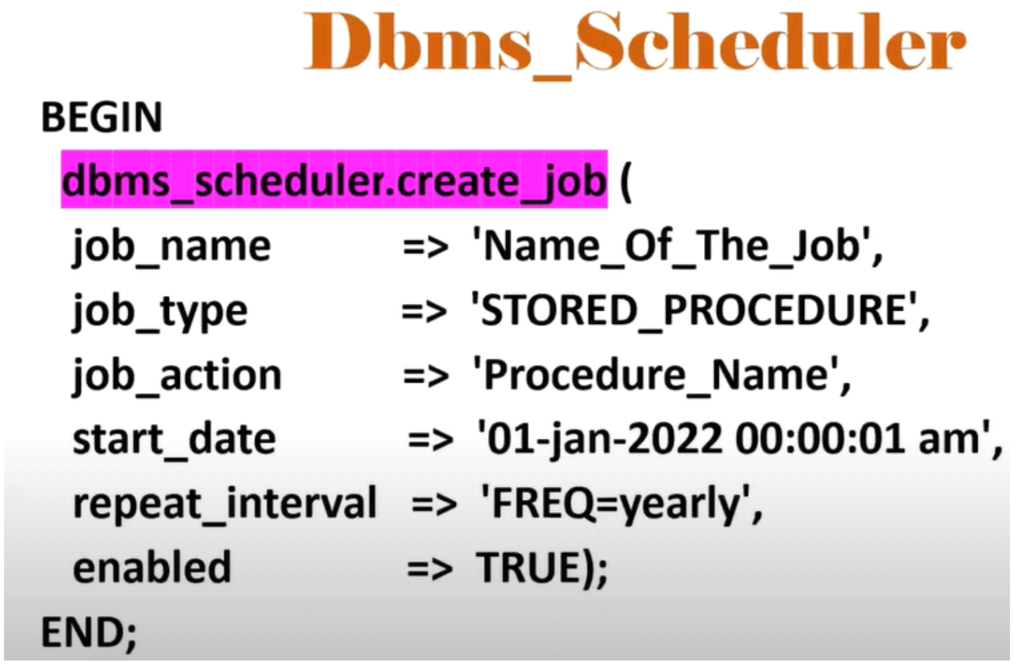
Proj -1

```
create table aadhar_details
(aadhar_number number,
   Name varchar(100),
   Address varchar(100),
   Mobile_number number,
   email varchar2(50));

-- insertion of new record
insert into aadhar_details values(446899997865,'Shiva','Katra',9876364573,'abc.gmail.com');

create or replace procedure aadhar_updation
(p_aadhar_number aadhar_details.aadhar_number%type,
p_name aadhar_details.Name%type,
p_address aadhar_details.Address%type,
p_mobile_number aadhar_details.Mobile_number%type,
p_email aadhar_details.email%type,
   P_Status OUT varchar2
)
As BEGIN
        update aadhar_details
        set Name = p_name,
                Address = p_address,
                Mobile_number = p_mobile_number,
                email = p_email
        where aadhar_number = p_aadhar_number;
If SQL%FOUND then
   commit;
        P_Status := 'Aadhar Details are updated successfully.';
else
   rollback;
   P_Status := 'Error in server, please try after sometime.';
end if;
end aadhar_updation;

-- Execute procedure create anonymous block
declare
        result varchar2(100);
begin
        aadhar_updation(
   p_aadhar_number => 446899997865,
p_name => 'Shivam',
p_address => 'Katra',
p_mobile_number => 94457320656,
p_email => 'abcde@gmail.com',
   P_Status => result);
dbms_output.put_line(result);
```

```
end;

select * from aadhar_details;
```

PLSQL PROJECT 2

Wednesday, January 22, 2025 11:36 AM



Dbms_Scheduler

```
BEGIN
 dbms_scheduler.create_job (
  job_name         =>  'Name_Of_The_Job',
  job_type         =>  'STORED_PROCEDURE',
  job_action       =>  'Procedure_Name',
  start_date       =>  '01-jan-2022 00:00:01 am',
  repeat_interval  =>  'FREQ=yearly',
  enabled          =>  TRUE);
END;
```

```
-- we cant directly use DDL commands in procedure and function so we use dynamic SQL
-- DBMS_Scheduler - this package provided by oracle to schedule any package, funciton, procedure
-- Project - New Employee Leaves Credit
-- Casual & Sick leave - 1 month - 0.5 day || Earned leaves - 1 month - 1 Day
        -- 0.25 (cl+sl), 0.5 (EL)
create table employee (
   EmpID number, DOJ date, Casual_leaves number(10,2), Sick_leaves number(10,2), Earned_leaves
number(10,2)
);

insert into employee values(100, '01-JAN-2025',6,6,12);

select * from employee;

create or replace procedure New_employees_leaves_credit(P_EmpID number, P_Message OUT
Varchar2)
as
   L_joining_Day number;
   L_joining_Month number;
        L_Casual number;
        L_Sick number;
        L_earned number;
begin
   select TO_CHAR(SYSDATE,'DD'), TO_CHAR(SYSDATE,'MM') into  L_joining_Day,L_joining_Month from
dual;
   If L_joining_Day <=15 then
          L_Casual := (12-L_joining_Month)*0.5 + 0.5;
      L_Sick := (12-L_joining_Month)*0.5 + 0.5;
      L_earned := (12-L_joining_Month)+1;
```

```
    else
            L_Casual := (12-L_joining_Month)*0.5 + 0.25;
            L_Sick := (12-L_joining_Month)*0.5 + 0.25;
        L_earned := (12-L_joining_Month)+0.5;
    End if;
        Insert into employee(EmpID , DOJ , Casual_leaves , Sick_leaves , Earned_leaves ) values
                                                    (P_EmpID,sysdate,L_Casual,L_Sick,L_earned);
        commit;
        P_Message := 'Leaves are credited successfully for EMPID : '||P_EmpID;
end New_employees_leaves_credit;

-- Execute Procedure
Declare
        Message Varchar2(200);
Begin
        New_employees_leaves_credit(103,Message);
        Dbms_output.put_line(Message);
End;
```

PLSQL Project 3

Wednesday, January 22, 2025 11:36 AM

```
create or replace procedure Sample_procedure
as
        L_Seating_Type Varchar2(10) := 'Sleeper';
        L_Bus_Number number;
        L_Date_of_Journey DATE;
begin
        dbms_output.put_line('Seating Type: '|| L_Seating_Type);
end Sample_procedure;

--execute procedure
begin
        Sample_procedure;
end;

--sys_refcursor is a weak REF cursor, and it is a pointer to query result set.

Create table Bus_Details(
   Bus_number number,
   Bus_Operator Varchar2(100),
   Journey_Date Date,
   Source Varchar2(100),
   Destination Varchar2(100),
   Price number,
   Seats_Available number(2),
   Seating_Type Varchar2(100),
   Bus_Type Varchar2(100),
   Bus_availability char(1),
   Departure_Time Varchar2(100),
   Arrival_Time Varchar2(100),
   Bus_rating number,
   TV_Availability char(1),
   Charging_point char(1),
   Social_distancing char(1));

Insert into Bus_Details values (19087,'CGR Travels','23-Jan-2024','Wakad','Indore',700,32,'Seater','Non-
AC','Y','22:05','11:15',5,'Y','Y','Y');
Insert into Bus_Details values (19567,'DDF Travels','25-
Jan-2024','Wakad','Indore',900,36,'Sleeper','AC','Y','21:05','10:15',4,'Y','Y','Y');
Select * from Bus_Details;

Create or replace Procedure Bus_Details_Display
   (P_DOJ IN DATE,
   P_Source IN Varchar2,
   P_Destination IN Varchar2,
   P_result OUT SYS_REFCURSOR)
AS
BEGIN
        OPEN P_Result FOR
```

```
                    select * from Bus_Details
                                                where Bus_availability  = 'Y' AND Journey_Date = P_DOJ AND Source =
                                                P_Source AND Destination = P_Destination;
END Bus_Details_Display;

-- execution
declare
    Result SYS_REFCURSOR;
          Bus_Dtls_Record Bus_Details%ROWTYPE;
begin
    Bus_Details_Display(
    P_DOJ => '23-Jan-2024',
    P_Source => 'Wakad',
    P_Destination => 'Indore',
    P_Result => Result
    );
Loop
    Fetch Result into Bus_DTLS_Record;
          EXIT WHEN Result%notfound;
          dbms_output.put_line('Bus Operator : '|| Bus_Dtls_Record.Bus_Operator||'--Departure Time : '||
          Bus_Dtls_Record.Departure_Time);
          End Loop;
          close Result;
end;
end;
```

Proj -4

Thursday, January 23, 2025 12:27 PM

PLSQL prac ques

Tuesday, February 25, 2025 1:00 PM

1. How would you write a PL/SQL block to fetch employee details based on their department ID and print the employee names?

Step-by-Step Answer:

Step 1: Define the PL/SQL Block

- We need to use an anonymous PL/SQL block to retrieve employee data based on a department ID.

Step 2: Declare Variables

- Declare the necessary variables (e.g., cursor, employee name, department ID).

Step 3: Open a Cursor to Fetch Employee Data

- Use a **cursor** to retrieve employee data based on the department ID.

Step 4: Use Loop to Fetch and Print Data

- Loop through the cursor to fetch employee details and print the employee names.

Step 5: Handle Exceptions

- Handle any potential exceptions like no data found.

Code Example:

plsql

Copy

```
DECLARE
   -- Declare cursor to select employee names and department
   CURSOR emp_cursor IS
      SELECT employee_name
      FROM employees
      WHERE department_id = :dept_id;  -- Bind variable for department ID
emp_name employees.employee_name%TYPE;  -- Variable to hold employee name
BEGIN
   -- Ask user for the department ID
   :dept_id := &dept_id;  -- Accept department ID as input
-- Open the cursor
   OPEN emp_cursor;
-- Loop through each record
   LOOP
      FETCH emp_cursor INTO emp_name;
      EXIT WHEN emp_cursor%NOTFOUND;  -- Exit when no more rows are found
      DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);  -- Print employee name
   END LOOP;
-- Close the cursor
   CLOSE emp_cursor;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No employees found for department ' || :dept_id);
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Explanation:

- This block prompts the user to input a department ID and then fetches and prints the names of employees in that department.
- The **cursor** is used to iterate through the employee records.
- Exception handling is included to display a message if no data is found or if there is any error.

2. How would you create a stored procedure to calculate the total salary of employees in a specific department?

Step-by-Step Answer:

Step 1: Define the Stored Procedure

- Create a **stored procedure** that takes the department ID as an input parameter and calculates the total salary.

Step 2: Declare Variables

- Declare a variable to store the total salary.

Step 3: Query Employee Data

- Use a **SELECT** statement to get the sum of salaries for the specified department.

Step 4: Return the Result

- The result can either be returned through an output parameter or printed.

Code Example:

plsql

```
Copy
CREATE OR REPLACE PROCEDURE calculate_total_salary(
   dept_id IN NUMBER,  -- Input department ID
   total_salary OUT NUMBER  -- Output total salary
) AS
BEGIN
   -- Calculate the total salary by summing up salaries for the given department
   SELECT SUM(salary) INTO total_salary
   FROM employees
   WHERE department_id = dept_id;
-- Display the total salary
   DBMS_OUTPUT.PUT_LINE('Total Salary for Department ' || dept_id || ' is: ' || total_salary);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('No data found for department ' || dept_id);
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Explanation:

- The procedure **calculate_total_salary** takes dept_id as an input parameter and returns the total salary through an output parameter total_salary.
- The **SUM()** function is used to calculate the total salary.
- Exception handling is implemented in case of no data or errors.

3. How would you write a trigger that automatically updates the employee's bonus when the salary is updated in the employees table?

Step-by-Step Answer:

Step 1: Define the Trigger

- Create a **BEFORE UPDATE trigger** on the employees table that checks if the salary has changed.

Step 2: Update the Bonus

- In the trigger body, calculate and update the **bonus** based on the new salary.

Step 3: Handle Exceptions

- Include exception handling for any potential errors during the update process.

Code Example:

```
plsql
Copy
CREATE OR REPLACE TRIGGER update_bonus_on_salary_change
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN
   -- Check if salary is updated
   IF :OLD.salary != :NEW.salary THEN
      -- Update the bonus based on the new salary (e.g., 10% of the new salary)
      :NEW.bonus := :NEW.salary * 0.10;
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error in trigger: ' || SQLERRM);
END;
```

Explanation:

- The **trigger** fires **before** an update on the employees table.
- It checks if the **salary** has been changed and if so, updates the **bonus** to 10% of the new salary.
- **Exception handling** is included to log any errors that occur during the trigger execution.

4. How would you write a function to return the total number of employees in a department?

Step-by-Step Answer:

Step 1: Define the Function

- Create a **function** that takes a department ID as an argument and returns the total number of employees in that department.

Step 2: Query Employee Data

- Use a **SELECT COUNT()** query to get the number of employees.

Code Example:

```
plsql
Copy
CREATE OR REPLACE FUNCTION get_employee_count(dept_id IN NUMBER)
RETURN NUMBER IS
   emp_count NUMBER;
```

```
BEGIN
   -- Count the number of employees in the given department
   SELECT COUNT(*) INTO emp_count
   FROM employees
   WHERE department_id = dept_id;
RETURN emp_count;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN 0;  -- Return 0 if no employees are found in the department
   WHEN OTHERS THEN
      RETURN NULL;  -- Return NULL if there is an error
END;
```

Explanation:

- The function **get_employee_count** returns the total number of employees in a given department.
- It uses the **COUNT(*)** function to count the rows in the employees table.
- Exception handling returns 0 if no employees are found or NULL if an error occurs.

5. How would you write a PL/SQL block to implement a loop that fetches all employees and prints their names and salaries until a salary exceeds 100,000?

Step-by-Step Answer:

Step 1: Declare Variables

- Declare a **cursor** to fetch employee details and a variable for the salary.

Step 2: Use a Loop to Fetch Data

- Loop through the cursor and check if the salary exceeds 100,000, in which case exit the loop.

Code Example:

```
 DECLARE
   CURSOR emp_cursor IS
      SELECT employee_name, salary
      FROM employees;
emp_name employees.employee_name%TYPE;
   emp_salary employees.salary%TYPE;
BEGIN
   OPEN emp_cursor;
LOOP
      FETCH emp_cursor INTO emp_name, emp_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
-- Print employee name and salary
      DBMS_OUTPUT.PUT_LINE('Employee: ' || emp_name || ' - Salary: ' || emp_salary);
-- Exit the loop if salary exceeds 100,000
      IF emp_salary > 100000 THEN
         DBMS_OUTPUT.PUT_LINE('Salary exceeds 100,000, exiting loop.');
         EXIT;
      END IF;
   END LOOP;
CLOSE emp_cursor;
END;
```

Explanation:

- The PL/SQL block opens a cursor that fetches employee names and salaries from the employees table.
- It loops through the records and prints each employee's name and salary.
- The loop exits when the salary exceeds 100,000, and a message is displayed.

```
--write a PL/SQL block to implement a loop that fetches all employees and prints their names and salaries until a salary exceeds 100,000?
declare
   cursor cur is select * from employees;
begin
   for rec in cur loop
   dbms_output.put_line('Name of the employee ' || rec.first_name ||' and Salary is ' ||rec.salary);
         IF rec.salary > 100000 THEN
            DBMS_OUTPUT.PUT_LINE('Salary exceeds 100,000, exiting loop.');
            EXIT;
         END IF;

   end loop;
end;
```