Exception handling in PL/pgSQL (PostgreSQL's procedural language) allows you to manage errors gracefully within your functions and procedures. The EXCEPTION block is used to catch and handle errors when something goes wrong during execution.

Here’s a simple example that demonstrates how to use exception handling in a PostgreSQL procedure.

**Business Case Scenario**

Suppose we have a scenario where we are updating employee salaries, but we want to handle the case where an employee ID does not exist (to avoid an error).

**Step 1: Create the employees Table**

Here’s the DDL for the employees table:

sql

Copy code

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

name VARCHAR(100),

salary NUMERIC

);

-- Insert sample data

INSERT INTO employees (name, salary) VALUES

('Alice Johnson', 50000),

('Bob Smith', 60000),

('Charlie Brown', 30000);

**Step 2: Procedure with Exception Handling**

We’ll write a procedure to update an employee’s salary. If an employee with the given ID doesn’t exist, the procedure will catch the error and raise a custom message instead of failing.

sql

Copy code

CREATE OR REPLACE PROCEDURE update\_salary(emp\_id INT, new\_salary NUMERIC)

LANGUAGE plpgsql AS $$

BEGIN

-- Try to update the employee's salary

UPDATE employees

SET salary = new\_salary

WHERE employee\_id = emp\_id;

-- Check if the update affected no rows (meaning the employee ID doesn't exist)

IF NOT FOUND THEN

-- Raise an exception manually

RAISE EXCEPTION 'Employee with ID % does not exist.', emp\_id;

END IF;

EXCEPTION

-- Handle any exception that occurs during the update

WHEN OTHERS THEN

-- RAISE NOTICE to show a friendly error message

RAISE NOTICE 'An error occurred: %', SQLERRM; -- SQLERRM holds the error message

END;

$$;

**Explanation**

* **BEGIN...END**: The main block where the procedure tries to update the employee’s salary.
* **IF NOT FOUND THEN**: This checks if the UPDATE affected no rows, which happens if the employee ID doesn’t exist. If no rows were updated, we raise a custom exception.
* **RAISE EXCEPTION**: Used to raise a custom error if a condition is met (in this case, if the employee ID is not found).
* **EXCEPTION**: This block catches any errors that occur in the BEGIN block.
  + **WHEN OTHERS THEN**: A generic exception handler that catches all exceptions.
  + **RAISE NOTICE**: Outputs a message to the console/log with the error description using SQLERRM (which holds the error message).

**Step 3: Call the Procedure and Handle Errors**

1. **Updating salary for an existing employee** (this should work without any issue):

sql

Copy code

CALL update\_salary(1, 55000); -- This should successfully update Alice Johnson's salary

1. **Updating salary for a non-existent employee** (this will trigger the exception):

sql

Copy code

CALL update\_salary(99, 55000); -- Employee ID 99 doesn't exist, so an exception will be raised

**Output**

For the first call (with employee ID 1), it will update the salary and complete successfully.

For the second call (with employee ID 99, which doesn’t exist), you’ll see the following message:

vbnet

Copy code

NOTICE: An error occurred: Employee with ID 99 does not exist.

**Step 4: Verify the Changes**

You can check the data in the employees table to verify the update:

sql

Copy code

SELECT \* FROM employees;

**Sample Output**

| **employee\_id** | **name** | **salary** |
| --- | --- | --- |
| 1 | Alice Johnson | 55000 |
| 2 | Bob Smith | 60000 |
| 3 | Charlie Brown | 30000 |

You’ll see that Alice’s salary was updated, and no changes were made for the non-existent employee ID 99.

**Summary**

In this example:

* We used exception handling (EXCEPTION block) to catch and manage errors in a controlled way.
* We raised a custom exception when a specific condition was met (employee ID not found).
* The WHEN OTHERS clause allowed us to catch any error and print a friendly message.

This method is useful in business applications where operations might fail due to various reasons (like missing data), and you want to handle these gracefully instead of letting the process fail abruptly.

1/2

4o mini