# Programa RDEMPS

**COBOL**

IDENTIFICATION DIVISION.

PROGRAM-ID. RDEMPS.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT EMPLOYEE-OUT

ASSIGN TO EMPLOYEE

ORGANIZATION IS LINE SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE STATUS IS ECODE.

DATA DIVISION.

FILE SECTION.

FD EMPLOYEE-OUT

RECORD CONTAINS 272 CHARACTERS.

01 EMPLOYEE-OUT-REC PIC X(272).

WORKING-STORAGE SECTION.

01 WS-EOF-INDICATOR PIC X(1) VALUE 'N'.

01 ECODE PIC X(2).

EXEC SQL

INCLUDE EMPS-CPY

END-EXEC.

EXEC SQL

INCLUDE SQLCA

END-EXEC.

PROCEDURE DIVISION.

OPEN-FILES.

OPEN OUTPUT EMPLOYEE-OUT

EXEC SQL

DECLARE EMPLOYEE-CURSOR CURSOR FOR

SELECT \* FROM EMPLOYEE-TABLE

END-EXEC

EXEC SQL

OPEN EMPLOYEE-CURSOR

END-EXEC

PERFORM READ-EMPLOYEE UNTIL WS-EOF-INDICATOR = 'Y'.

CLOSE-FILES.

CLOSE EMPLOYEE-OUT.

EXEC SQL

CLOSE EMPLOYEE-CURSOR

END-EXEC.

STOP RUN.

READ-EMPLOYEE.

EXEC SQL

FETCH EMPLOYEE-CURSOR INTO

:EMPLOYEE-ID,

:EMPLOYEE-LASTNAME,

:EMPLOYEE-FIRSTNAME,

:EMPLOYEE-SALARY,

:EMPLOYEE-DEPARTMENT,

:EMPLOYEE-LEVEL

END-EXEC.

IF SQLCODE < 0 OR SQLCODE = 100

MOVE 'Y' TO WS-EOF-INDICATOR

ELSE

MOVE 'N' TO WS-EOF-INDICATOR

END-IF.

IF WS-EOF-INDICATOR = 'N'

WRITE EMPLOYEE-OUT-REC FROM EMPLOYEE-RECORD

END-IF.

EXIT.

**PYTHON**

import pymysql # Assuming MySQL database

# Database connection details (replace with your actual details)

host = "localhost"

user = "username"

password = "password"

database = "EMPS-CPY"

# SQL statements

open\_cursor\_sql = """

DECLARE EMPLOYEE\_CURSOR CURSOR FOR

SELECT \* FROM EMPLOYEE\_TABLE

"""

fetch\_employee\_sql = """

FETCH EMPLOYEE\_CURSOR INTO (%s, %s, %s, %s, %s, %s)

"""

close\_cursor\_sql = "CLOSE EMPLOYEE\_CURSOR"

def main():

# Connect to the database

connection = pymysql.connect(host=host, user=user, password=password, database=database)

cursor = connection.cursor()

try:

# Open database cursor

cursor.execute(open\_cursor\_sql)

# Loop through employee records

while True:

# Fetch employee data

employee\_id, last\_name, first\_name, salary, department, level = cursor.fetchone()

# Check for end of data (using None for null values)

if not any([employee\_id, last\_name, first\_name, salary, department, level]):

break

# Implement your logic to process employee data (e.g., write to a file)

# ...

# Close database cursor

cursor.execute(close\_cursor\_sql)

except pymysql.Error as err:

print(f"Error: {err}")

finally:

# Close connection

connection.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Explanation:**

* We import the pymysql library for connecting to a MySQL database (replace with the appropriate library for your database type).
* We define connection details and SQL statements for opening/closing the cursor and fetching data.
* The main function establishes a connection, opens a cursor, and loops through employee records using fetchone.
* We check for the end of data by verifying if all fetched values are None.
* An empty placeholder (...) is included for your logic to process the employee data (e.g., writing to a file).
* Error handling is implemented using a try-except block with pymysql.Error for database errors.
* The connection is closed using a finally block to ensure proper resource management.

# Programa PAYROLL

**COBOL**

IDENTIFICATION DIVISION.

PROGRAM-ID. PAYROLL.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT EMPLOYEE-IN

ASSIGN TO EMPLOYEE

ORGANIZATION IS LINE SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE STATUS IS WS-FS1.

SELECT PAYROLL-OUT

ASSIGN TO PAYROLL

ORGANIZATION IS LINE SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE STATUS IS WS-FS2.

DATA DIVISION.

FILE SECTION.

FD EMPLOYEE-IN

RECORD CONTAINS 272 CHARACTERS.

01 EMPLOYEE-RECORD.

COPY EMPS-NO-01.

FD PAYROLL-OUT

RECORD CONTAINS 282 CHARACTERS.

01 PAYROLL-RECORD.

COPY EMPS-NO-01.

05 MONTHLY-SALARY PIC S9(7)V99 COMP-3.

WORKING-STORAGE SECTION.

01 WS-VAR.

05 WS-FS1 PIC 9(02).

05 WS-FS2 PIC 9(02).

05 WS-EOF-SW PIC X(01).

88 WS-EOF VALUE 'Y'.

88 WS-NOT-EOF VALUE 'N'.

01 ANNUAL-SALARY PIC S9(7)V99 COMP-3.

PROCEDURE DIVISION.

OPEN-FILES.

OPEN INPUT EMPLOYEE-IN.

OPEN OUTPUT PAYROLL-OUT.

READ-EMPLOYEE.

SET WS-NOT-EOF TO TRUE.

PERFORM UNTIL WS-EOF

READ EMPLOYEE-IN

AT END SET WS-EOF TO TRUE

NOT AT END

MOVE CORRESPONDING EMPLOYEE-RECORD

TO PAYROLL-RECORD

COMPUTE MONTHLY-SALARY =

EMPLOYEE-SALARY OF PAYROLL-RECORD

/ 12

WRITE PAYROLL-RECORD

END-READ

END-PERFORM.

CLOSE-FILES.

CLOSE EMPLOYEE-IN.

CLOSE PAYROLL-OUT.

STOP RUN.

**PYTHON**

def calculate\_payroll(employee\_data):

"""

Calculates the monthly salary for a given employee record.

Args:

employee\_data (dict): A dictionary containing employee data.

Returns:

dict: A dictionary with the original employee data and the calculated monthly salary.

"""

monthly\_salary = employee\_data["EMPLOYEE\_SALARY"] / 12

return {\*\*employee\_data, "MONTHLY\_SALARY": monthly\_salary}

def main():

# Assuming you have a way to read employee data from EMPLOYEE-IN (e.g., a list)

employee\_data\_list = []

# ... your logic to read from EMPLOYEE-IN

# Process each employee record and calculate monthly salary

processed\_employees = [calculate\_payroll(employee) for employee in employee\_data\_list]

# Assuming you have a way to write the processed data to PAYROLL-OUT (e.g., a function)

write\_payroll\_data(processed\_employees)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Explanation:**

* We define a function calculate\_payroll that takes an employee record (dictionary) as input and calculates the monthly salary by dividing the annual salary by 12. It returns a dictionary with the original data and the calculated monthly salary.
* The main function outlines the overall process:
  + It assumes you have a way to read employee data from EMPLOYEE-IN (replace the placeholder with your actual logic).
  + It iterates through the employee data using a list comprehension and calls calculate\_payroll for each record.
  + Finally, it assumes you have a function write\_payroll\_data to write the processed data to PAYROLL-OUT (replace the placeholder with your implementation).

# Programa LISTROLL

**COBOL**

IDENTIFICATION DIVISION.

PROGRAM-ID. LISTROLL.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT PAYROLL-IN

ASSIGN TO PAYROLL

ORGANIZATION IS LINE SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE STATUS IS WS-FS1.

DATA DIVISION.

FILE SECTION.

FD PAYROLL-IN

RECORD CONTAINS 282 CHARACTERS.

01 PAYROLL-RECORD.

COPY EMPS-NO-01.

05 MONTHLY-SALARY PIC S9(7)V99 COMP-3.

WORKING-STORAGE SECTION.

01 SALARY-DISP PIC Z9(7).99.

01 DISP-LINE PIC X(250).

01 WS-VAR.

05 WS-FS1 PIC 9(02).

05 WS-EOF-SW PIC X(01).

88 WS-EOF VALUE 'Y'.

88 WS-NOT-EOF VALUE 'N'.

PROCEDURE DIVISION.

OPEN-FILES.

OPEN OUTPUT PAYROLL-IN.

READ-EMPLOYEE.

SET WS-NOT-EOF TO TRUE.

PERFORM UNTIL WS-EOF

READ PAYROLL-IN

AT END SET WS-EOF TO TRUE

NOT AT END

MOVE MONTHLY-SALARY TO SALARY-DISP

STRING EMPLOYEE-DEPARTMENT DELIMITED BY SIZE,

SPACE,

EMPLOYEE-LEVEL DELIMITED BY SIZE,

SPACE,

EMPLOYEE-LASTNAME DELIMITED BY SIZE,

SPACE,

EMPLOYEE-FIRSTNAME DELIMITED BY SIZE,

SPACE,

SALARY-DISP DELIMITED BY SIZE

INTO DISP-LINE

DISPLAY DISP-LINE

END-READ

END-PERFORM.

CLOSE-FILES.

CLOSE PAYROLL-IN.

STOP RUN.

**PYTHON**

def main():

# Assuming you have a way to read processed employee data (from PAYROLL-OUT)

processed\_employees = []

# ... your logic to read from processed employee data

# Loop through each employee record

for employee in processed\_employees:

# Extract relevant data

department = employee["EMPLOYEE\_DEPARTMENT"]

level = employee["EMPLOYEE\_LEVEL"]

last\_name = employee["EMPLOYEE\_LASTNAME"]

first\_name = employee["EMPLOYEE\_FIRSTNAME"]

monthly\_salary = employee["MONTHLY\_SALARY"]

# Format salary for display

formatted\_salary = "{:,.2f}".format(monthly\_salary) # Use f-strings or formatting function

# Construct display line

display\_line = f"{department:<15}{level:<5}{last\_name:<20}{first\_name:<20}{formatted\_salary:>15}"

# Print the display line

print(display\_line)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Explanation:**

* The main function assumes you have a way to read the processed employee data obtained from PAYROLL (replace the placeholder with your actual logic).
* It iterates through each employee record in the processed data.
* We extract relevant data like department, level, names, and monthly salary from the employee record dictionary.
* The monthly\_salary is formatted for display using f-strings or an appropriate formatting function (replace with your preferred method).
* A display line is constructed by combining department, level, names, and the formatted salary with proper spacing using string formatting methods.
* Finally, the display line is printed to the console.

# JCL PAYROLL

**COBOL**

//PAYROLL JOB (383),PAYROLL,CLASS=A,MSGCLASS=X

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*

//\* Calculo de la nomina mensual a partir del sueldo anual

//\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP1 EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=\*

//SYSIN DD \*

DELETE EMPLOYEE.DAT PURGE

SET MAXCC = 0

/\*

//\*

//\* Lectura de la tabla de empleados de la empresa (COBOL-DB2)

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP2 EXEC PGM=IKJEFT01

//SYSTSPRT DD SYSOUT=A

//EMPLOYEE DD DSN=EMPLOYEE.DAT,DISP=(NEW,CATLG,DELETE),

// SPACE=(TRK,(1,1),RLSE),

// DCB=(RECFM=FB,LRECL=272,BLKSIZE=0)

//SYSTSIN DD \*

DSN SYSTEM(DB2SSID)

RUN PROGRAM(RDEMPS) PLAN(RDEMPLN)

END

/\*

//SYSUDUMP DD SYSOUT=A

//\*

//\* Ordenacion de la lista de empleados por dptmo, area y appellidos

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP3 EXEC PGM=SORT

//SYSOUT DD SYSOUT=A

//SORTIN DD DSN=EMPLOYEE.DAT,DISP=SHR

//SORTOUT DD DSN=EMPLOYEE.SORTED,DISP=(NEW,CATLG,DELETE),

// SPACE=(TRK,(1,1),RLSE),

// DCB=(RECFM=FB,LRECL=272,BLKSIZE=0)

//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)

//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(1,1),RLSE)

//SYSTDIN DD \*

SORT FIELDS=(21,5,CH,A,18,2,CH,A,6,150,CH,A)

/\*

//\*

//\* Calculo de la nomina a partir del sueldo anual

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP4 EXEC PGM=PAYROLL

//SYSOUT DD SYSOUT=A

//EMPLOYEE DD DSN=EMPLOYEE.SORTED,DISP=SHR

//PAYROLL DD DSN=PAYROLL.ALL.MMAAAA,DISP=(NEW,CATLG,DELETE),

// SPACE=(TRK,(1,1),RLSE),

// DCB=(RECFM=FB,LRECL=278,BLKSIZE=0)

//\*

//\* Listado de la nomina

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP5 EXEC PGM=LISTROLL

//SYSOUT DD SYSOUT=A

//PAYROLL DD DSN=PAYROLL.ALL.MMAAAA,DISP=SHR

//\*

//\* Envio del fichero de nomina a un sistema externo via FTP

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//STEP6 EXEC PGM=FTP,REGION=6M

//SYSOUT DD SYSOUT=A

//NETRC DD \*

FTPUSER FTPPASSWORD

BIN

CD /destination/path

PUT PAYROLL.ALL.MMAAAA

BYE

/\*

**PYTHON**

Divide el jcl en 3 scrips para phyton:

**1. Data Access Script (data\_access.py):**

Python

import pymysql # Assuming MySQL database

# Database connection details (replace with your actual details)

host = "localhost"

user = "username"

password = "password"

database = "EMPS-CPY"

# SQL statements

open\_cursor\_sql = """

DECLARE EMPLOYEE\_CURSOR CURSOR FOR

SELECT \* FROM EMPLOYEE\_TABLE

"""

fetch\_employee\_sql = """

FETCH EMPLOYEE\_CURSOR INTO (%s, %s, %s, %s, %s, %s)

"""

close\_cursor\_sql = "CLOSE EMPLOYEE\_CURSOR"

def get\_employee\_data():

"""

Retrieves employee data from the database.

Returns:

List of dictionaries: Each dictionary represents an employee record.

"""

connection = pymysql.connect(host=host, user=user, password=password, database=database)

cursor = connection.cursor()

try:

cursor.execute(open\_cursor\_sql)

employee\_data = []

while True:

employee\_id, last\_name, first\_name, salary, department, level = cursor.fetchone()

if not any([employee\_id, last\_name, first\_name, salary, department, level]):

break

employee\_data.append({

"EMPLOYEE\_ID": employee\_id,

"EMPLOYEE\_LASTNAME": last\_name,

"EMPLOYEE\_FIRSTNAME": first\_name,

"EMPLOYEE\_SALARY": salary,

"EMPLOYEE\_DEPARTMENT": department,

"EMPLOYEE\_LEVEL": level

})

cursor.execute(close\_cursor\_sql)

return employee\_data

except pymysql.Error as err:

print(f"Error: {err}")

finally:

connection.close()

if \_\_name\_\_ == "\_\_main\_\_":

employee\_data = get\_employee\_data()

print(employee\_data) # Example: print the retrieved data for debugging

**2. Data Processing Script (data\_processing.py):**

Python

import pandas as pd

def process\_employee\_data(employee\_data):

"""

Processes and sorts employee data.

Args:

employee\_data (list of dicts): List of employee records (dictionaries).

Returns:

pandas.DataFrame: Sorted employee data.

"""

# Convert to pandas DataFrame

df = pd.DataFrame(employee\_data)

# Sort by department, area, and last name

df = df.sort\_values(by=["EMPLOYEE\_DEPARTMENT", "EMPLOYEE\_LEVEL", "EMPLOYEE\_LASTNAME"], ascending=True)

return df

if \_\_name\_\_ == "\_\_main\_\_":

# Assuming you have employee data retrieved from data\_access.py (replace with your actual data)

employee\_data = [

{"EMPLOYEE\_ID": 1, "EMPLOYEE\_LASTNAME": "Smith", "EMPLOYEE\_FIRSTNAME": "John", "EMPLOYEE\_SALARY": 120000, "EMPLOYEE\_DEPARTMENT": "IT", "EMPLOYEE\_LEVEL": 2},

# ... more employee records

]

processed\_data = process\_employee\_data(employee\_data)

print(processed\_data.to\_string()) # Example: print the processed data for debugging

**3. Salary Calculation Script (salary\_calculation.py):**

Python

def calculate\_payroll(employee\_data):

"""

Calculates monthly salary for each employee record.

Args:

employee\_data (pandas.DataFrame): Employee data as a DataFrame.

Returns:

pandas.DataFrame: Employee data with monthly salary added.

"""

employee\_data["MONTHLY\_SALARY"] = employee\_data["EMPLOYEE\_SALARY"] / 12

return employee\_data

if \_\_name\_\_ == "\_\_main\_\_":

# Assuming you have processed employee data from data\_processing.py (replace with your actual data)

processed\_data = pd.DataFrame({

"EMPLOYEE\_ID": [1, 2, 3],

"EMPLOYEE\_LASTNAME": ["Smith", "Jones", "Williams"],

"EMPLOYEE\_FIRSTNAME": ["John", "Peter", "Mary"],

"EMPLOYEE\_SALARY": [120000,

Como la solución no nos convence pedimos a Gemini que nos genere un solo scrip que tenga las 3 funcionalidades del jcl. A parte le solicitamos que en vez de ftp genere una Excel en el directorio C:/drive.

import pymysql # Assuming MySQL database

import pandas as pd

import openpyxl # For Excel file generation

def get\_employee\_data():

"""

Retrieves employee data from the database.

Returns:

List of dictionaries: Each dictionary represents an employee record.

"""

connection = pymysql.connect(host="localhost", user="username", password="password", database="EMPS-CPY")

cursor = connection.cursor()

try:

cursor.execute(open\_cursor\_sql)

employee\_data = []

while True:

employee\_id, last\_name, first\_name, salary, department, level = cursor.fetchone()

if not any([employee\_id, last\_name, first\_name, salary, department, level]):

break

employee\_data.append({

"EMPLOYEE\_ID": employee\_id,

"EMPLOYEE\_LASTNAME": last\_name,

"EMPLOYEE\_FIRSTNAME": first\_name,

"EMPLOYEE\_SALARY": salary,

"EMPLOYEE\_DEPARTMENT": department,

"EMPLOYEE\_LEVEL": level

})

cursor.execute(close\_cursor\_sql)

return employee\_data

except pymysql.Error as err:

print(f"Error: {err}")

finally:

connection.close()

def process\_employee\_data(employee\_data):

"""

Processes and sorts employee data.

Args:

employee\_data (list of dicts): List of employee records (dictionaries).

Returns:

pandas.DataFrame: Sorted employee data.

"""

df = pd.DataFrame(employee\_data)

df = df.sort\_values(by=["EMPLOYEE\_DEPARTMENT", "EMPLOYEE\_LEVEL", "EMPLOYEE\_LASTNAME"], ascending=True)

return df

def calculate\_payroll(employee\_data):

"""

Calculates monthly salary for each employee record.

Args:

employee\_data (pandas.DataFrame): Employee data as a DataFrame.

Returns:

pandas.DataFrame: Employee data with monthly salary added.

"""

employee\_data["MONTHLY\_SALARY"] = employee\_data["EMPLOYEE\_SALARY"] / 12

return employee\_data

def generate\_report(processed\_data):

"""

Generates the final payroll report in an Excel file.

Args:

processed\_data (pandas.DataFrame): Employee data with monthly salary.

"""

# Create an Excel workbook and worksheet

wb = openpyxl.Workbook()

ws = wb.active

# Set column headers

ws.cell(row=1, column=1).value = "EMPLOYEE\_ID"

ws.cell(row=1, column=2).value = "EMPLOYEE\_LASTNAME"

ws.cell(row=1, column=3).value = "EMPLOYEE\_FIRSTNAME"

ws.cell(row=1, column=4).value = "EMPLOYEE\_DEPARTMENT"

ws.cell(row=1, column=5).value = "EMPLOYEE\_LEVEL"

ws.cell(row=1, column=6).value = "MONTHLY\_SALARY"

# Add employee data to the worksheet

row\_num = 2

for index, row in processed\_data.iterrows():

ws.cell(row=row\_num, column=1).value = row["EMPLOYEE\_ID"]

ws.cell(row=row\_num, column=2).value = row["EMPLOYEE\_LASTNAME"]

ws.cell(row=num=row\_num, column=3).value = row["EMPLOYEE\_FIRSTNAME"]

ws.cell(row=row\_num, column=4).value = row["EMPLOYEE\_DEPARTMENT"]

ws.cell(row=row\_num, column=5).value = row["EMPLOYEE\_LEVEL"]

ws.cell(row=row\_num, column=6).value = row["MONTHLY\_SALARY"]

row\_num += 1

# Save the Excel file to the C: drive

wb.save("payroll\_report.xlsx") # Replace with your desired filename

if \_\_name\_\_ == "\_\_main\_\_":

# Retrieve employee data from the database

employee\_data = get\_employee\_data()

# Process and sort the employee data

processed\_data = process\_employee\_data(employee\_data)