

# Documentación del Procedimiento: Implementación de Base de Datos Oracle XE 11g con SQL Developer

## Integrantes

- Marco Abarca R.
- Alex Jadan
- Luis Sarango

## Índice

- Introducción
- 1. Análisis del Dataset y Diseño del Modelo Relacional
- 2. Creación de Modelos y Scripts SQL
  - Modelo Conceptual
  - Modelo Lógico
  - Modelo Físico
- 3. Importación de Datos CSV
- 4. Implementación de Triggers, Procedimientos y Funciones
  - Trigger para Cálculo de `total_gross_pay`
  - Trigger para Validación de `base_salary`
  - Procedimiento para Actualizar Nombre de Departamento
- 5. Desarrollo de Consultas Complejas y Análisis de Rendimiento
  - Consultas sin Índices
    - Consulta 1: Salario promedio por departamento y tipo de salario en un año específico
    - Consulta 2: Empleados con el mayor `total_gross_pay` por departamento en un trimestre específico

- Consulta 3: Historial de salarios de un empleado específico a lo largo de los años
  - Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral
- Creación de Índices
- Consultas con Índices
  - Consulta 1: Salario promedio por departamento y tipo de salario en un año específico (Con Índices)
  - Consulta 2: Empleados con el mayor `total_gross_pay` por departamento en un trimestre específico (Con Índices)
  - Consulta 3: Historial de salarios de un empleado específico a lo largo de los años (Con Índices)
  - Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral (Con Índices)
- Análisis de Resultados de Rendimiento
- Conclusión
- Referencias Biográficas
  - Edgar F. Codd: El Padre del Modelo Relacional
  - Los Creadores de SQL: Chamberlin y Boyce
  - Los Fundadores de Oracle: Ellison, Miner y Oates
  - Referencias

## Introducción

Este documento detalla el proceso de diseño, implementación y optimización de una base de datos en Oracle XE 11g utilizando SQL Developer, a partir de un conjunto de datos proporcionado en formato CSV. El objetivo es construir un modelo de datos relacional robusto, importar eficientemente los datos, e incorporar lógica de negocio mediante triggers y procedimientos almacenados. Además, se analizará el rendimiento de consultas complejas, con y sin la aplicación de índices, para demostrar su impacto en la eficiencia de la base de datos.

La información aquí presentada busca ser una guía clara y concisa, permitiendo una comprensión rápida de los pasos clave y las decisiones tomadas durante el desarrollo. Se incluyen marcadores para capturas de pantalla que complementarán visualmente cada etapa del proceso.

# 1. Análisis del Dataset y Diseño del Modelo Relacional

El primer paso crucial fue comprender la estructura del archivo que escogimos: `employee_earnings_300000.csv`. Este dataset contiene una gran cantidad de registros relacionados con las ganancias de empleados, incluyendo información sobre su identidad, departamento, puesto de trabajo y diversos componentes salariales. La inspección inicial reveló la necesidad de normalizar los datos para evitar redundancias y asegurar la integridad referencial.

Se identificaron las siguientes entidades principales:

- **EMPLOYEES:** Para almacenar la información básica de cada empleado.
- **DEPARTMENTS:** Para los nombres y números de los departamentos.
- **JOB\_TITLES:** Para los códigos y títulos de los puestos de trabajo.
- **EARNINGS:** Para los registros de ganancias de los empleados, que incluyen detalles salariales por año y trimestre.

La relación entre estas entidades es fundamental. Un empleado pertenece a un departamento y ocupa un puesto de trabajo. Los registros de ganancias están directamente asociados a un empleado, y a su vez, a un departamento y un puesto de trabajo, lo que permite un análisis detallado de las remuneraciones.

	CARTO6_ID	OBJECTID	CALENDAR_YEAR	QUARTER	LAST_NAME	FIRST_NAME	TITLE	JOB_CODE	DEPARTMENT_NAME	DEPARTMENT_NUMBER	BASE_SALARY	SALARY_TYPE	OVERTIME_GROSS
1	166	2197191	2020	2	Whipple Glover	Juanita	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
2	167	2197192	2020	3	Mackins	Karina	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
3	168	2197193	2020	2	Ammons	Lavinia	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
4	169	2197194	2020	1	Laird	Linda	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
5	170	2197195	2020	3	Higgins	Lisa	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
6	171	2197196	2020	1	Corbin	Louise	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
7	172	2197197	2020	1	Vega	Maria	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
8	173	2197198	2020	2	Pearson	Marianne	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
9	174	2197199	2020	2	Clarke	Marie	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
10	175	2197200	2020	2	Rush	Maryann	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
11	176	2197201	2020	3	Santore	Mary Jo	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
12	177	2197202	2020	3	Scott	Nancy	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
13	178	2197203	2020	3	Sanders	Nathaniel	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
14	179	2197204	2020	1	Iaguinto	Regina	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
15	180	2197205	2020	3	Young	Renda	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
16	181	2197206	2020	4	Gilyard	Sade	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
17	182	2197207	2020	1	Mathis	Sony	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
18	183	2197208	2020	3	Sims	Tami	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
19	184	2197209	2020	3	Jordan	Thelma	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
20	185	2197210	2020	2	Baskerville	Valerie	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
21	186	2197211	2020	1	Lancaster	Tolanda	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
22	187	2197212	2020	1	Evans	Pronne	School Crossing Guard	6D44	PPD Police	11	1270400	Salaried	
23	188	2197213	2019	4	Gallagher	Joan	School Crossing Guard	6D44	PPD Police	11	1270600	Salaried	
24	189	2197214	2019	2	Gallagher	Joan	School Crossing Guard	6D44	PPD Police	11	1270600	Salaried	
25	190	2197215	2019	3	Gallagher	Joan	School Crossing Guard	6D44	PPD Police	11	1270600	Salaried	
26	191	2197216	2020	1	Gallagher	Joan	School Crossing Guard	6D44	PPD Police	11	1270600	Salaried	

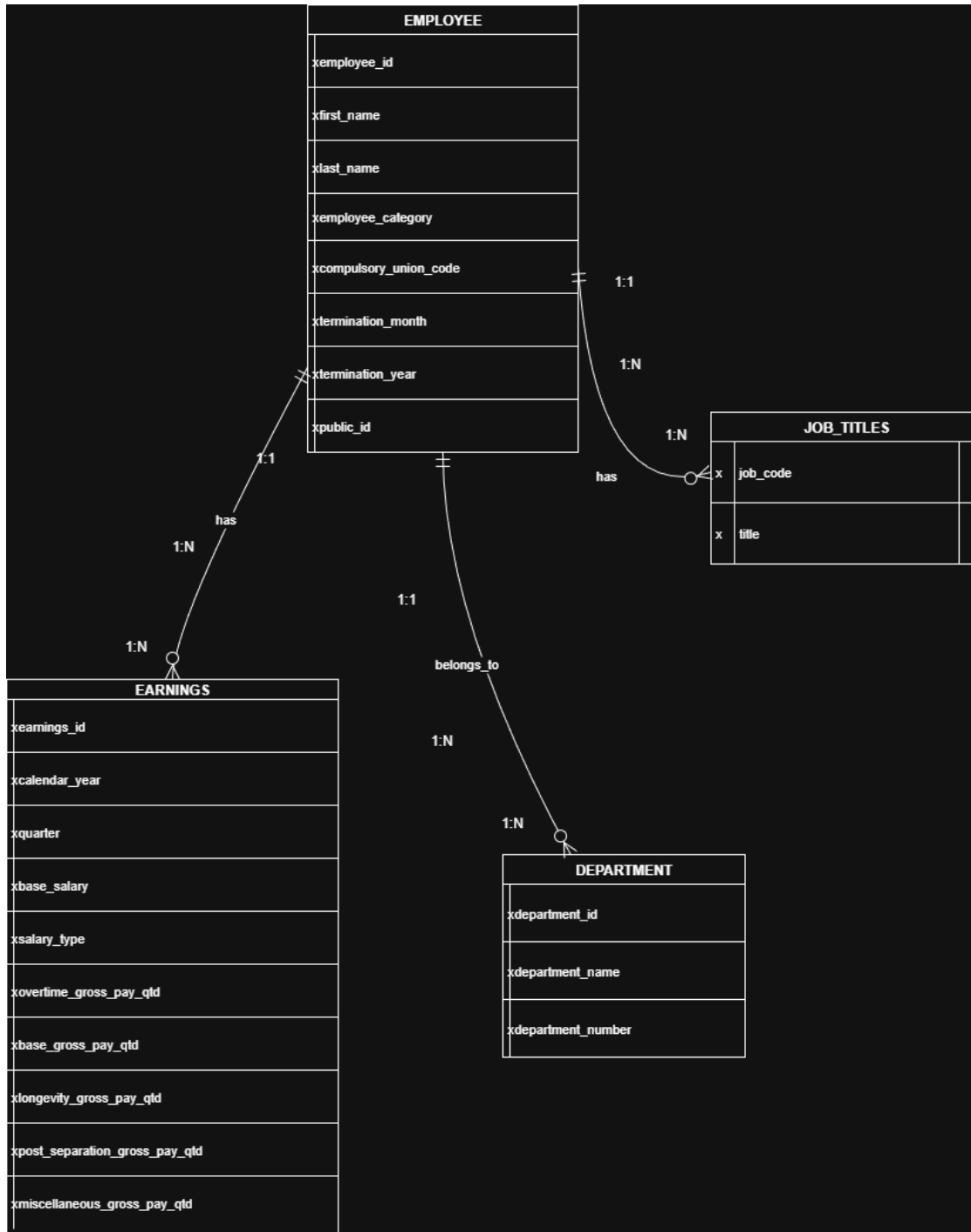
*Descripción: Vista de las primeras filas del archivo CSV, mostrando las columnas y el formato de los datos.*

## 2. Creación de Modelos y Scripts SQL

El diseño de la base de datos se articuló a través de modelos conceptual, lógico y físico. Estos modelos sirvieron como planos para la construcción de la base de datos, asegurando una estructura coherente y optimizada.

## Modelo Conceptual

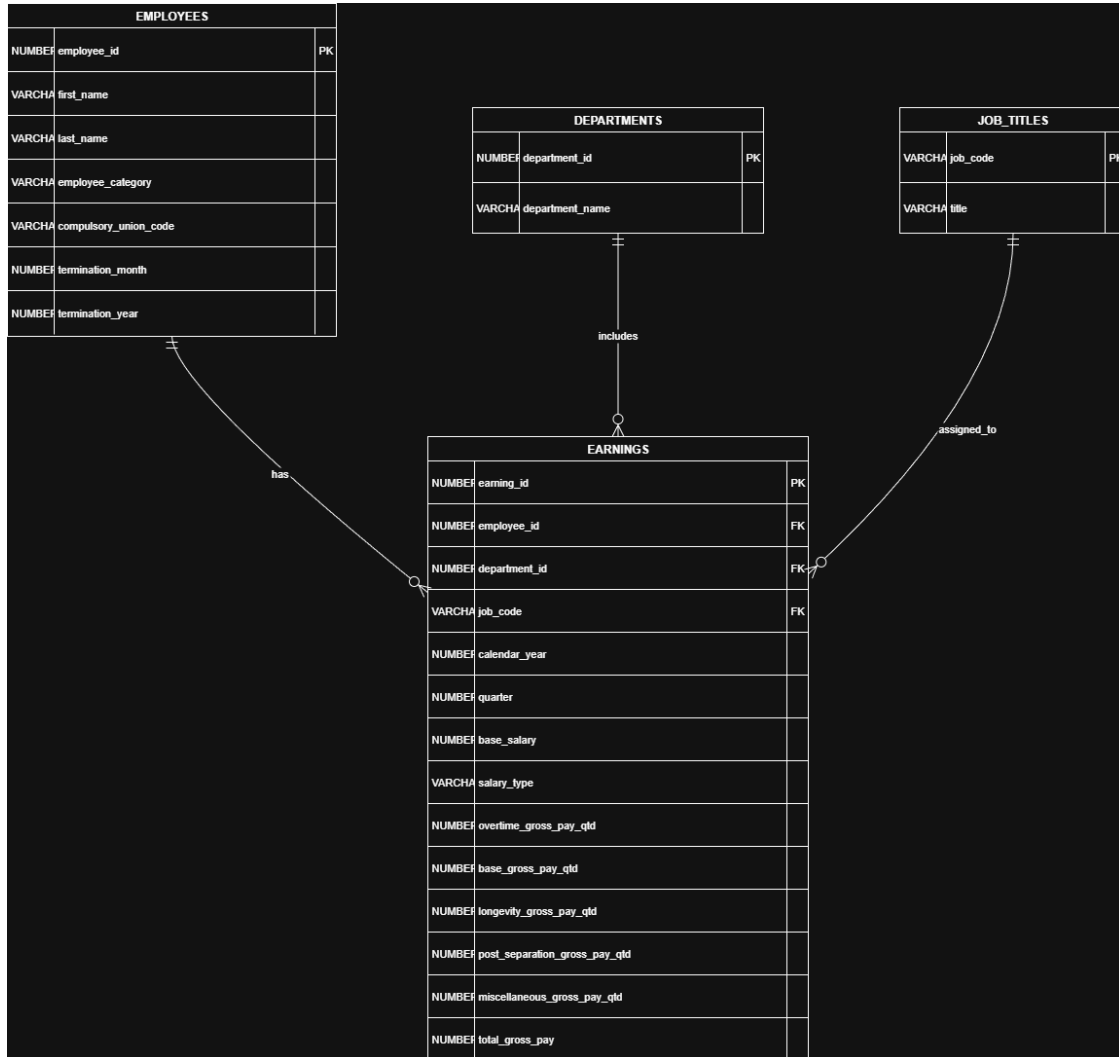
Representa las entidades y sus relaciones a un alto nivel, enfocándose en el 'qué' del negocio. Aquí se definieron las entidades EMPLOYEE, DEPARTMENT, JOB\_TITLE y EARNINGS y sus interconexiones básicas.



Descripción: Diagrama que ilustra las entidades principales y sus relaciones generales.

## Modelo Lógico

Este modelo refina el conceptual, añadiendo atributos específicos a cada entidad y detallando las relaciones, pero aún de forma independiente de la tecnología de base de datos. Se definieron las claves primarias y foráneas lógicas.



*Descripción: Diagrama que muestra las entidades con sus atributos y las relaciones detalladas.*

## Modelo Físico

El modelo físico es la implementación concreta para Oracle XE 11g. Se tradujeron las entidades y atributos a tablas y columnas SQL, especificando tipos de datos, restricciones de integridad (claves primarias, foráneas, NOT NULL) y la definición de secuencias para IDs autoincrementales. Los scripts SQL para la creación de estas tablas se generaron cuidadosamente para asegurar la compatibilidad y eficiencia con Oracle.

```
CREATE TABLE EMPLOYEES (  
    employee_id NUMBER ,  
    first_name VARCHAR2(100),  
    last_name VARCHAR2(100),  
    employee_category VARCHAR2(50),  
    compulsory_union_code VARCHAR2(10),  
    termination_month NUMBER(2),  
    termination_year NUMBER(4)  
);
```

```
CREATE TABLE DEPARTMENTS (  
    department_id NUMBER ,  
    department_name VARCHAR2(200)  
);
```

```
CREATE TABLE JOB_TITLES (  
    job_code VARCHAR2(20) ,  
    title VARCHAR2(200)  
);
```

```
CREATE TABLE EARNINGS (  
    earning_id NUMBER,  
    employee_id NUMBER NOT NULL,  
    department_id NUMBER NOT NULL,  
    job_code VARCHAR2(20) NOT NULL,  
    calendar_year NUMBER(4) NOT NULL,  
    quarter NUMBER(1) NOT NULL,  
    base_salary NUMBER(10, 2),  
    salary_type VARCHAR2(50),  
    overtime_gross_pay_qtd NUMBER(10, 2),  
    base_gross_pay_qtd NUMBER(10, 2),  
    longevity_gross_pay_qtd NUMBER(10, 2),  
    post_separation_gross_pay_qtd NUMBER(10, 2),  
    miscellaneous_gross_pay_qtd NUMBER(10, 2),  
    total_gross_pay NUMBER(10, 2)  
);
```

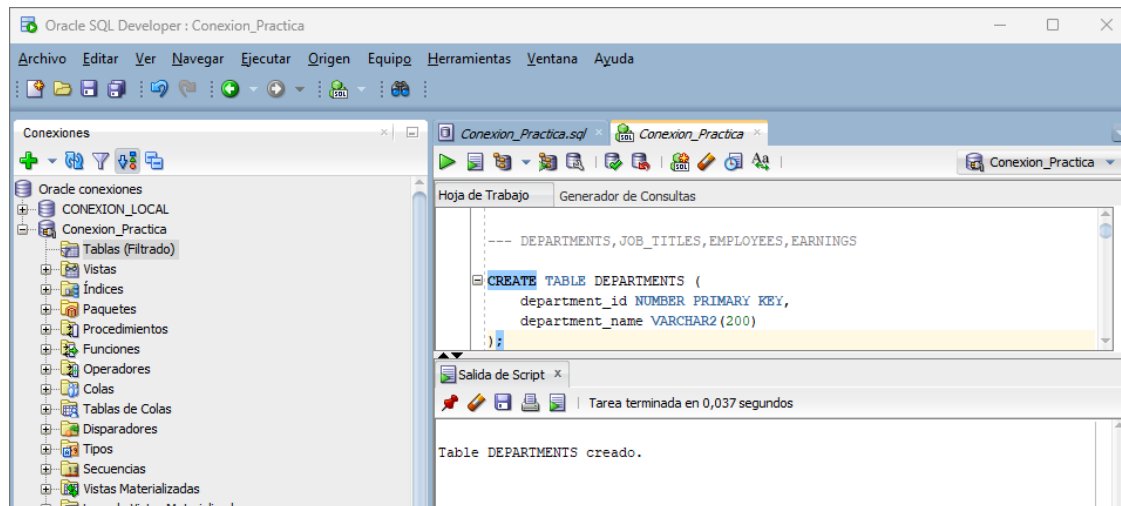
```
CREATE SEQUENCE earnings_seq  
START WITH 1  
INCREMENT BY 1  
NOCACHE  
NOCYCLE;
```

```
CREATE OR REPLACE TRIGGER earnings_before_insert -- trigger de control  
BEFORE INSERT ON EARNINGS  
FOR EACH ROW  
BEGIN  
    IF :NEW.earning_id IS NULL THEN  
        SELECT earnings_seq.NEXTVAL
```

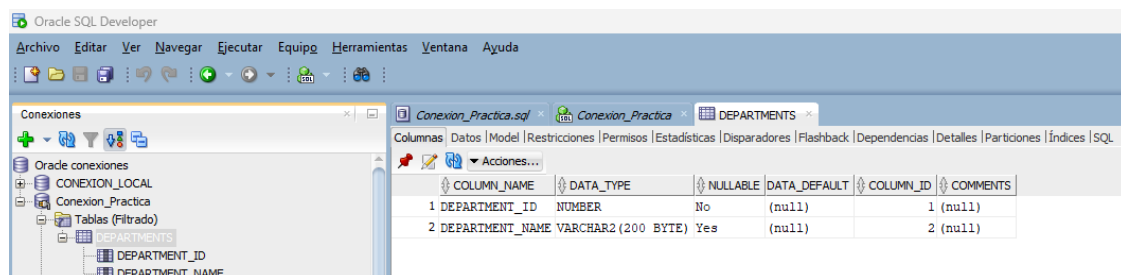
```

        INTO :NEW.earning_id
      FROM dual;
    END IF;
  END;

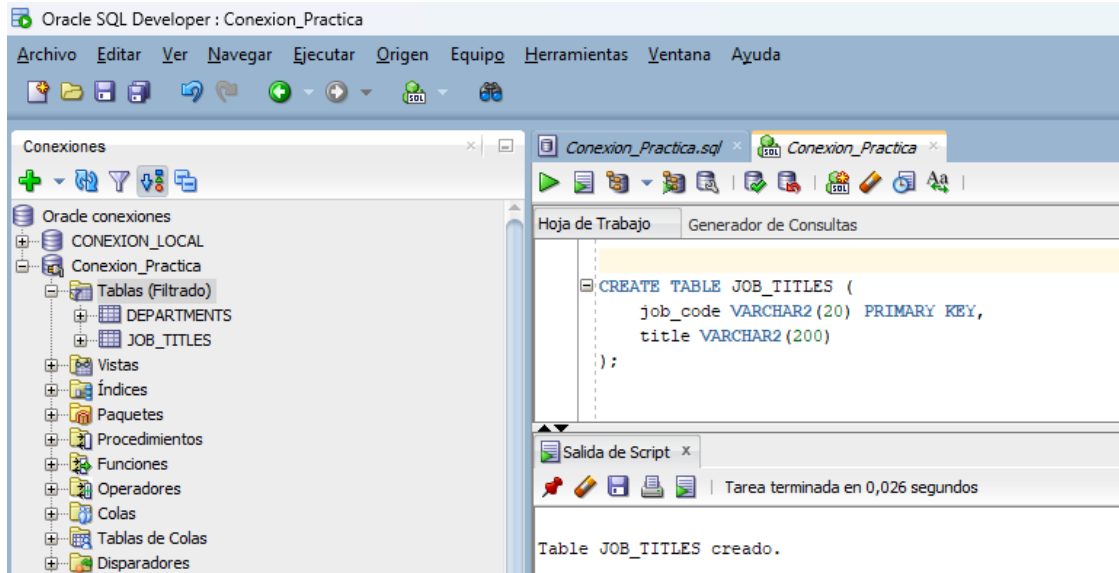
```



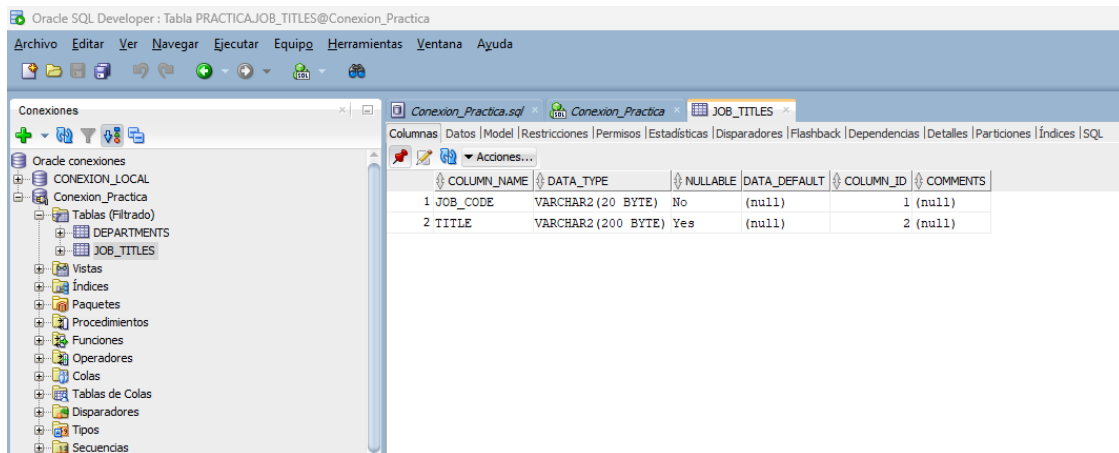
*Descripción: Creación de la Tabla DEPARTMENTS*



*Descripción: Propiedades de la Tabla DEPARTMENTS*

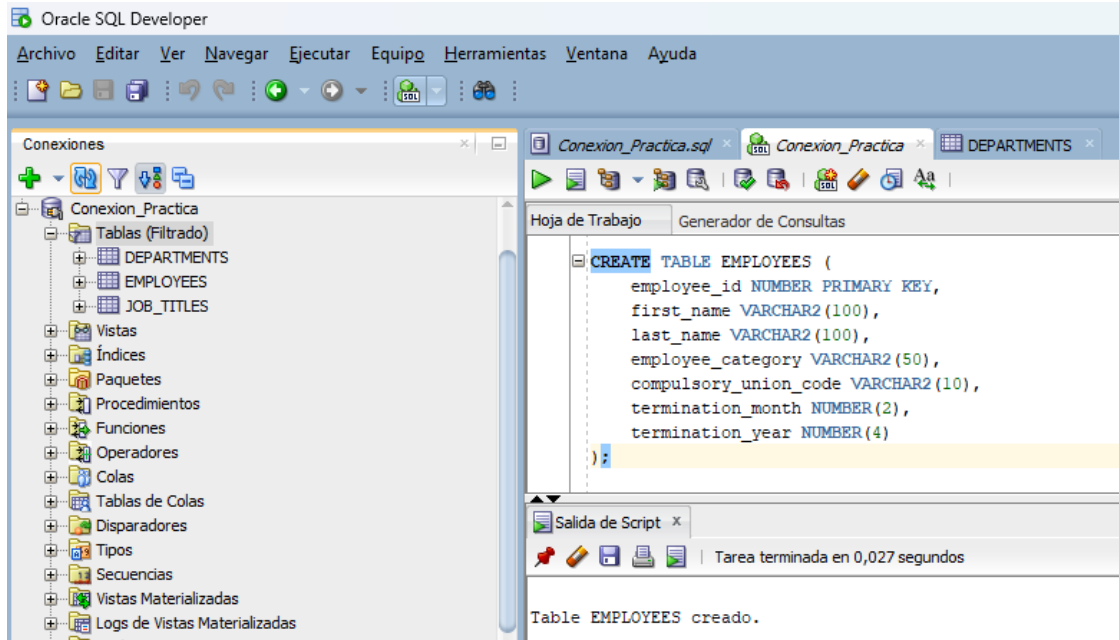


*Descripción: Creación de la Tabla JOB\_TITLES*

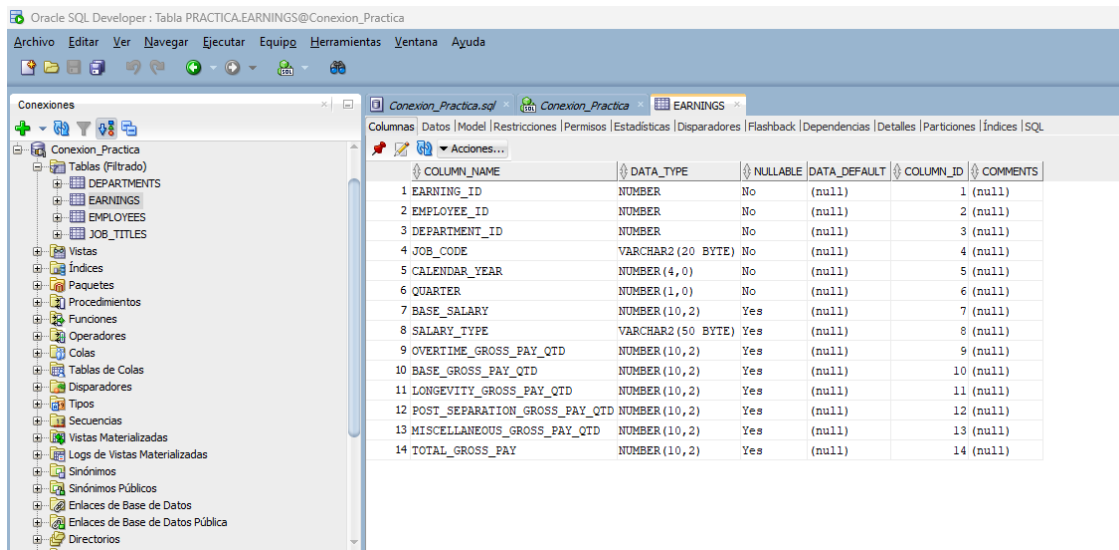


*Descripción: Propiedades de la Tabla JOB\_TITLES*





*Descripción: Creacion de la Tabla EMPLEADOS*



*Descripción: Propiedades de la Tabla EARNINGS*

```
21 CREATE TABLE EARNINGS (  
22     earning_id NUMBER PRIMARY KEY,  
23     employee_id NUMBER NOT NULL,  
24     department_id NUMBER NOT NULL,  
25     job_code VARCHAR2(20) NOT NULL,  
26     calendar_year NUMBER(4) NOT NULL,  
27     quarter NUMBER(1) NOT NULL,  
28     base_salary NUMBER(10, 2),  
29     salary_type VARCHAR2(50),  
30     overtime_gross_pay_qtd NUMBER(10, 2),  
31     base_gross_pay_qtd NUMBER(10, 2),  
32     longevity_gross_pay_qtd NUMBER(10, 2),  
33     post_separation_gross_pay_qtd NUMBER(10, 2),  
34     miscellaneous_gross_pay_qtd NUMBER(10, 2),  
35     total_gross_pay NUMBER(10, 2),  
36     CONSTRAINT fk_employee  
37         FOREIGN KEY (employee_id)  
38         REFERENCES EMPLOYEES (employee_id),  
39     CONSTRAINT fk_department  
40         FOREIGN KEY (department_id)  
41         REFERENCES DEPARTMENTS (department_id),  
42     CONSTRAINT fk_job_title  
43         FOREIGN KEY (job_code)  
44         REFERENCES JOB_TITLES (job_code)  
45 );  
46  
47 CREATE SEQUENCE earnings_seq  
48 START WITH 1  
49 INCREMENT BY 1  
50 NOCACHE  
51 NOCYCLE;
```

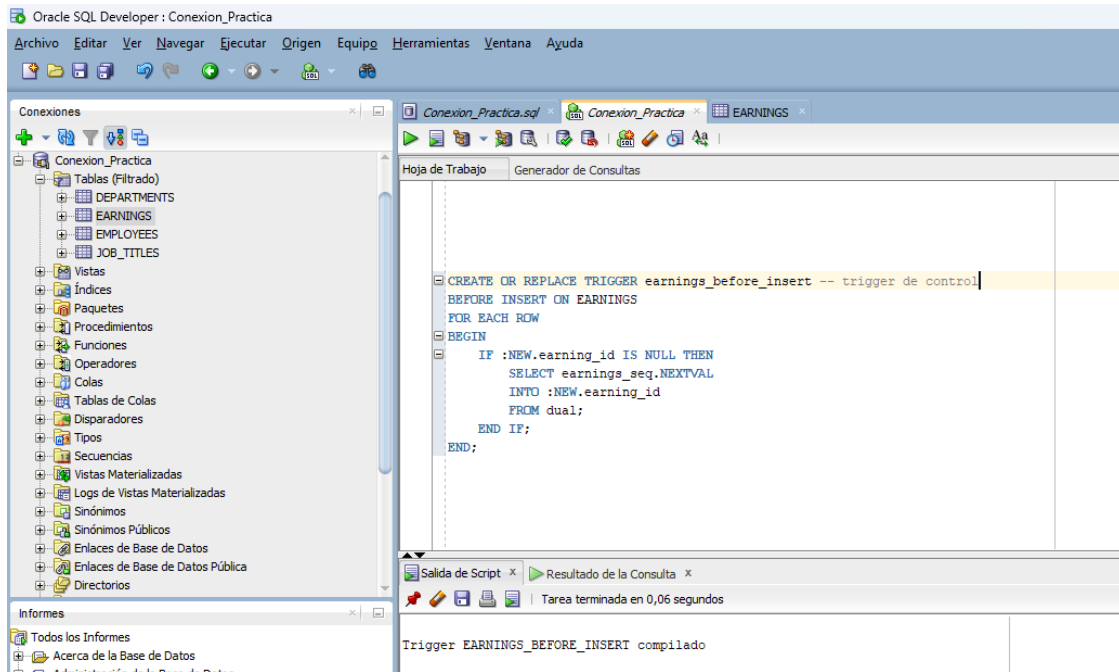
Salida de Script x

Tarea terminada en 0,246 segundos

Table EARNINGS creado.

Sequence EARNINGS\_SEQ creado.

Trigger EARNINGS\_BEFORE\_INSERT compilado



*Descripción: Creacion de la Tabla EARNINGS*

**Descripción General: Fragmento del script SQL con las sentencias CREATE TABLE para las tablas EMPLOYEES, DEPARTMENTS, JOB\_TITLES y EARNINGS.**

### 3. Importación de Datos CSV

La importación de 300,000 registros desde el CSV a la estructura relacional diseñada fue un paso crítico. Se optó por un enfoque en dos fases para manejar las dependencias de claves foráneas y asegurar la correcta normalización de los datos:

- 1. Carga a una Tabla Temporal de Respaldo (Como metodo de fallback):**  
Primero, el CSV completo se importó a una tabla temporal (TEMP\_EMPLOYEE\_EARNINGS) con todas las columnas definidas como VARCHAR2. Esto minimizó los problemas de tipo de datos durante la carga inicial.

```
-- tabla temporal

-- Esta tabla temporal se utiliza para cargar los datos crudos del CSV antes de su procesamiento y normalización.
CREATE TABLE TEMP_EMPLOYEE_EARNINGS (
  the_geom VARCHAR2(200),
  cartodb_id NUMBER,
  the_geom_webmercator VARCHAR2(200),
  objectid NUMBER,
  calendar_year NUMBER,
  quarter NUMBER,
  last_name VARCHAR2(100),
  first_name VARCHAR2(100),
  title VARCHAR2(200),
  job_code VARCHAR2(20),
  department_name VARCHAR2(200),
  department_number NUMBER,
  base_salary VARCHAR2(50), -- Importar como VARCHAR2 para manejar formatos variados
  salary_type VARCHAR2(50),
  overtime_gross_pay_qtd VARCHAR2(50),
  base_gross_pay_qtd VARCHAR2(50),
  longevity_gross_pay_qtd VARCHAR2(50),
);
```

Script Output x Query Result x

Task completed in 0.298 seconds

Trigger EARNINGS\_BEFORE\_INSERT compiled

Table TEMP\_EMPLOYEE\_EARNINGS created.

*Descripción: Script SQL para la creación de la tabla TEMP\_EMPLOYEE\_EARNINGS.*

Data Import Wizard - Step 1 of 5

### Data Preview

Source: Local File

File: C:\Users\mnabarca1\Documents\Separar dataset\employee\_earnings\_300000.csv

File Format

☒ Header After Skip Skip Rows: 0

Format: csv ☒ Preview Row Limit: 100

Encoding: UTF8

Delimiter: , Line Terminator: standard: CR LF, CR or LF

Left Endosure: " Right Endosure: "

File Contents

the_geom	cartodb_id	the_geom...	objectid	calendar_year	quarter	last_name	first_name	title	job_code	department.
	1		2197027	2023	4	Guess	Monte	Deputy Sher...	6K06	SHF Sheriff
	2		2197028	2022	4	Massi Jr	John	Police Serge...	6A04	PPD Police
	3		2197029	2022	4	Guess	Monte	Deputy Sher...	6K06	SHF Sheriff
	4		2197030	2019	4	Timms	James	Police Officer	1 6A02	PPD Police
	5		2197031	2022	3	Mcquillin	Charles	Firefighter	6B01	PPD Fire
	6		2197032	2022	4	Feinberg	Louis	Tipstaff 1 (G...	T251	FJD 1st Judi.
	7		2197033	2022	1	Kennedy	Sharon	Data Servic...	1D41	REV Revenue
	8		2197034	2019	4	Betts	Perry	Police Officer	1 6A02	PPD Police
	9		2197035	2025	2	Rauscher	Stephen	Firefighter	6B01	PPD Fire
	10		2197036	2019	4	Wallington	Michael	School Cros...	6D44	PPD Police

Help < Back Next > Finish Cancel

Data Import Wizard - Step 2 of 5

Import Method

Data Preview

Import Method

Choose Columns

Column Definition

Finish

Specify the method for importing data. For Staging External Table method, an external table will be created as a staging table for importing the Target Table. For other import methods, data is imported directly into the table.

Import Method: 

Insert Script

☐ Send Create Script to SQL Worksheet

Table Name: 

TEMP\_EMPLOYEE\_EARNINGS

☐ Import Row Limit: 

100

File Contents

the_geom	cartodb_id	the_geom...	objectid	calendar_year	quarter	last_name	first_name	title	job_code	department.
	63		2197089	2025	1	Dyches	India	Service Rep...	1A37	DPH Health
	64		2197090	2024	4	Foster	Kimmy	Firefighter	6B01	PPD Fire
	65		2197091	2019	2	King	Frencella	School Cros...	6D44	PPD Police
	66		2197092	2019	2	Jones	Renata	School Cros...	6D44	PPD Police
	67		2197093	2019	3	Jones	Renata	School Cros...	6D44	PPD Police
	68		2197094	2020	2	Jones	Renata	School Cros...	6D44	PPD Police
	69		2197095	2019	3	Thomas	Frances	School Cros...	6D44	PPD Police
	70		2197096	2019	2	Whalen	Carolina	School Cros...	6D44	PPD Police
	71		2197097	2019	4	Bass	Christine	School Cros...	6D44	PPD Police
	72		2197098	2019	2	Williams	Courtney	School Cros...	6D44	PPD Police
	73		2197099	2019	2	Parker	Danielle	School Cros...	6D44	PPD Police
	74		2197100	2019	2	Bruce	Debra	School Cros...	6D44	PPD Police
	75		2197101	2019	4	Cain	Dorothy	School Cros...	6D44	PPD Police
	76		2197102	2019	4	Brown	Eleanor	School Cros...	6D44	PPD Police
	77		2197103	2019	4	Cameron	Eulene	School Cros...	6D44	PPD Police
	78		2197104	2019	4	Bassler	Jane	School Cros...	6D44	PPD Police
	79		2197105	2019	2	Patterson	Karen	School Cros...	6D44	PPD Police

Help

< Back

Next >

Finish

Cancel

Data Import Wizard - Step 3 of 5

Choose Columns

Data Preview

Import Method

Choose Columns

Column Definition

Finish

Select the columns to import from the data set and arrange them in the order you want.

Available Columns

Selected Columns

the\_geom

cartodb\_id

the\_geom\_webmercator

objectid

calendar\_year

quarter

last\_name

first\_name

title

job\_code

department\_name

department\_number

base\_salary

salary\_type

overtime\_gross\_pay\_qtd

base\_gross\_pay\_qtd

longevity\_gross\_pay\_qtd

post\_separation\_gross\_pay\_qtd

miscellaneous\_gross\_pay\_qtd

employee\_category

compulsory\_union\_code

termination\_month

File Contents

the_geom	cartodb_id	the_geom...	objectid	calendar_year	quarter	last_name	first_name	title	job_code	department.
	63		2197089	2025	1	Dyches	India	Service Rep...	1A37	DPH Health
	64		2197090	2024	4	Foster	Kimmy	Firefighter	6B01	PPD Fire
	65		2197091	2019	2	King	Frencella	School Cros...	6D44	PPD Police

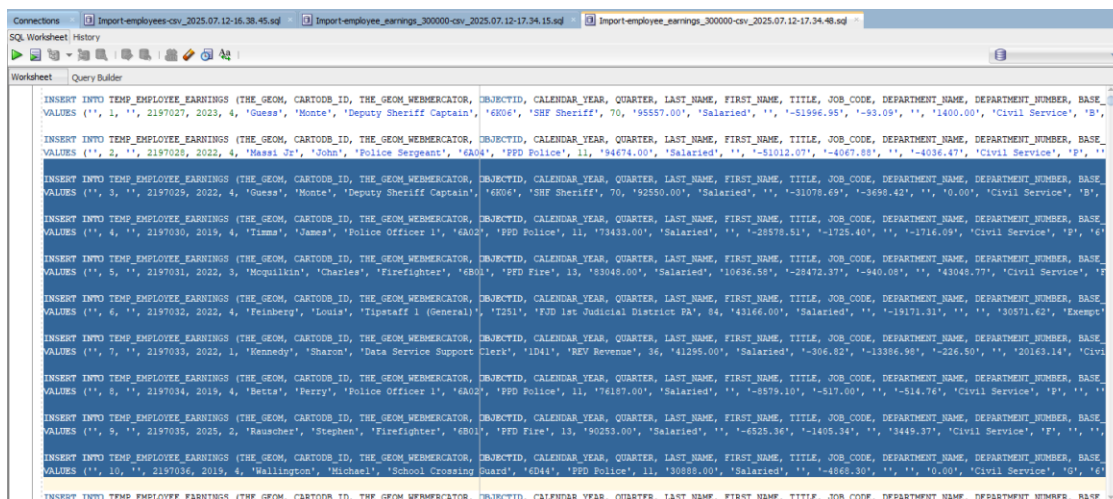
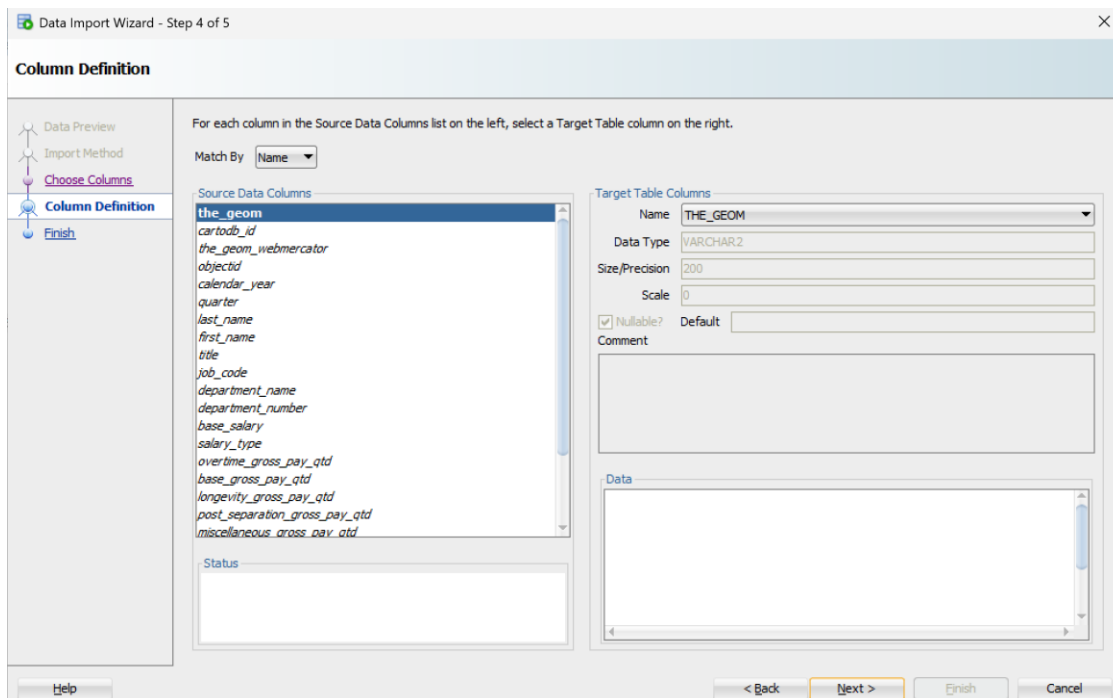
Help

< Back

Next >

Finish

Cancel



*Descripción: Proceso de importación del CSV a TEMP\_EMPLOYEE\_EARNINGS usando el asistente de SQL Developer.*

## 1. Limpiar el Dataset Completo:

- Usamos pandas, una librería rica en el manejo de datos en python para limpiar el csv y enlistarlo para ser importado hacia nuestra base de datos :

**import** pandas **as** pd

*# 1 Cargar el CSV original*

df = pd.read\_csv('employee\_earnings\_300000.csv')

*# 2 Calcular 'total\_gross\_pay' como columna derivada para earnings*



```

df['total_gross_pay'] = (
    df['base_salary'].fillna(0) +
    df['overtime_gross_pay_qtd'].fillna(0) +
    df['base_gross_pay_qtd'].fillna(0) +
    df['longevity_gross_pay_qtd'].fillna(0) +
    df['post_separation_gross_pay_qtd'].fillna(0) +
    df['miscellaneous_gross_pay_qtd'].fillna(0)
)

# ----- DEPARTMENTS -----
cols_departments = ['department_number', 'department_name']
df_departments =
df[cols_departments].drop_duplicates(subset=['department_number'])
df_departments = df_departments.rename(columns={'department_number':
'department_id'})
df_departments.to_csv('departments.csv', index=False)

# ----- JOB_TITLES -----
cols_job_titles = ['job_code', 'title']
df_job_titles =
df[cols_job_titles].drop_duplicates(subset=['job_code'])
df_job_titles.to_csv('job_titles.csv', index=False)

# ----- EMPLOYEES -----
cols_employees = ['public_id', 'first_name', 'last_name',
'employee_category',
'compulsory_union_code', 'termination_month',
'termination_year']
df_employees = df[cols_employees].drop_duplicates(subset=['public_id'])
df_employees = df_employees.rename(columns={'public_id':
'employee_id'})
df_employees.to_csv('employees.csv', index=False)

# ----- EARNINGS -----
cols_earnings = ['public_id', 'department_number', 'job_code',
'calendar_year', 'quarter',
'base_salary', 'salary_type',
'overtime_gross_pay_qtd',
'base_gross_pay_qtd', 'longevity_gross_pay_qtd',
'post_separation_gross_pay_qtd',
'miscellaneous_gross_pay_qtd',
'total_gross_pay']

df_earnings = df[cols_earnings].copy()
df_earnings = df_earnings.rename(columns={
'public_id': 'employee_id',
'department_number': 'department_id'
})
# Aquí, si deseas eliminar duplicados exactos en earnings (opcional):

```

```
# df_earnings = df_earnings.drop_duplicates()

df_earnings.to_csv('earnings.csv', index=False)

# Mensajes de confirmación
print("✓ Archivos limpios y separados generados exitosamente:")
print("- departments.csv")
print("- job_titles.csv")
print("- employees.csv")
print("- earnings.csv")
```

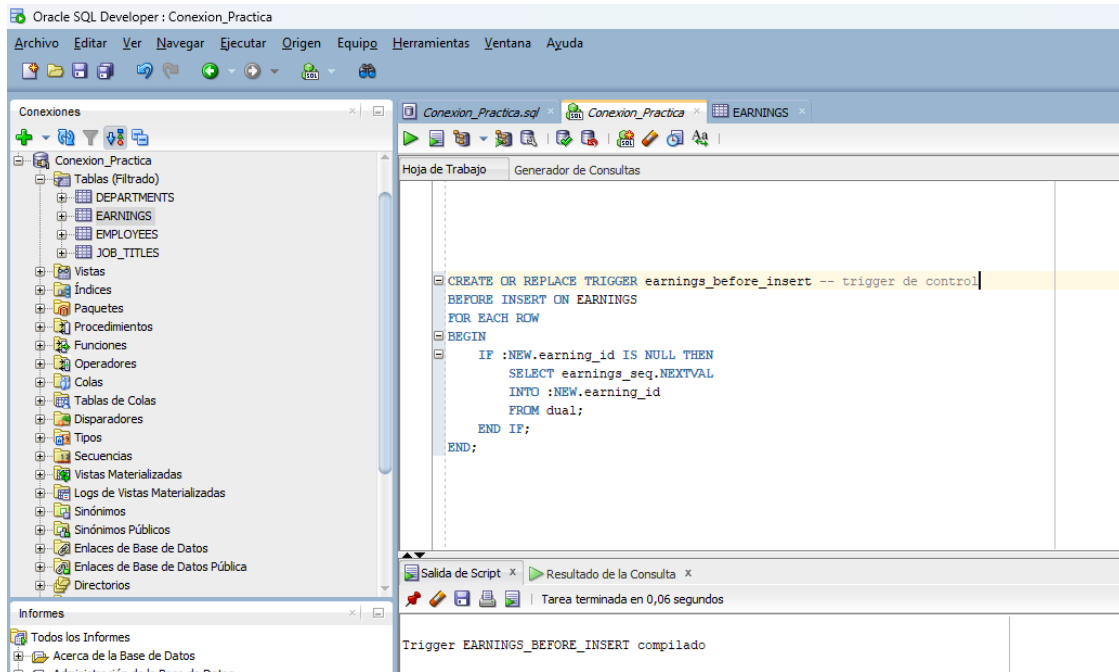
```
separar_csv.py
1 import pandas as pd
2
3 # Cargar el CSV original
4 df = pd.read_csv('employee_earnings_300000.csv')
5
6 # Calcular 'total_gross_pay' como columna derivada para earnings
7 df['total_gross_pay'] = (
8     df['base_salary'].fillna(0) +
9     df['overtime_gross_pay_qtd'].fillna(0) +
10    df['base_gross_pay_qtd'].fillna(0) +
11    df['longevity_gross_pay_qtd'].fillna(0) +
12    df['post_separation_gross_pay_qtd'].fillna(0) +
13    df['miscellaneous_gross_pay_qtd'].fillna(0)
14 )
15
16 # DEPARTMENTS
17 cols_departments = ['department_number', 'department_name']
18 df_departments = df[cols_departments].drop_duplicates(subset=['department_number'])
19 df_departments = df_departments.rename(columns={'department_number': 'department_id'})
20
21 # JOB TITLES
22 cols_job_titles = ['job_title', 'department_id']
23 df_job_titles = df[cols_job_titles].drop_duplicates(subset=['job_title'])
24 df_job_titles = df_job_titles.rename(columns={'job_title': 'job_title_id'})
25
26 # EMPLOYEES
27 cols_employees = ['employee_id', 'department_id', 'job_title_id', 'salary', 'commission_pct', 'start_date', 'end_date']
28 df_employees = df[cols_employees].drop_duplicates(subset=['employee_id'])
29 df_employees = df_employees.rename(columns={'employee_id': 'employee_id'})
30
31 # EARNINGS
32 cols_earnings = ['employee_id', 'total_gross_pay']
33 df_earnings = df[cols_earnings].drop_duplicates(subset=['employee_id'])
34 df_earnings = df_earnings.rename(columns={'total_gross_pay': 'total_gross_pay'})
35
36 # Guardar los archivos CSV
37 df_departments.to_csv('departments.csv', index=False)
38 df_job_titles.to_csv('job_titles.csv', index=False)
39 df_employees.to_csv('employees.csv', index=False)
40 df_earnings.to_csv('earnings.csv', index=False)
```

```
PS C:\Users\Usuario\Desktop\Separar dataset> & C:\Users\Usuario\AppData\Local\Programs\Python\Python312\python.exe "C:/Users/Usuario/Desktop/Separar dataset/separar_csv.py"
✓ Archivos limpios y separados generados exitosamente:
- departments.csv
- job_titles.csv
- employees.csv
- earnings.csv
PS C:\Users\Usuario\Desktop\Separar dataset>
```

*Descripción: Proceso de limpieza del CSV inicial a 4 archivos resultantes: departments.csv, job\_titles.csv, employees.csv y earnings.csv usando la libreria pandas de Python.*

1. **Transformación y Carga a Tablas Finales:** Como metodo de fallback importariamos los datos desde la tabla temporal, utilizando sentencias INSERT INTO ... SELECT FROM para poblar las tablas DEPARTMENTS, JOB\_TITLES, EMPLOYEES y EARNINGS. Sin embargo, importamos los archivos csv limpios con nuestro script de python como metodo principal. Este método permitió extraer los valores únicos para DEPARTMENTS y JOB\_TITLES, y resolver las claves foráneas sin mayor dificultad. Además, se realizaron conversiones de tipo de datos y cálculos de atributos derivados (como total\_gross\_pay) durante este proceso.





*Descripción: Visualización de un trigger de control para la tabla EARNINGS antes de la importación, mostrando la omisión de valores nulos antes de la importación. Evitando así futuros conflictos*

Connections x Import-departments-csv\_2025.07.13-16.24.20.sql x

SQL Worksheet History

Worksheet Query Builder

```
SET DEFINE OFF

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (70, 'SHF Sheriff');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (11, 'PPD Police');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (13, 'PFD Fire');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (84, 'FJD 1st Judicial District PA');

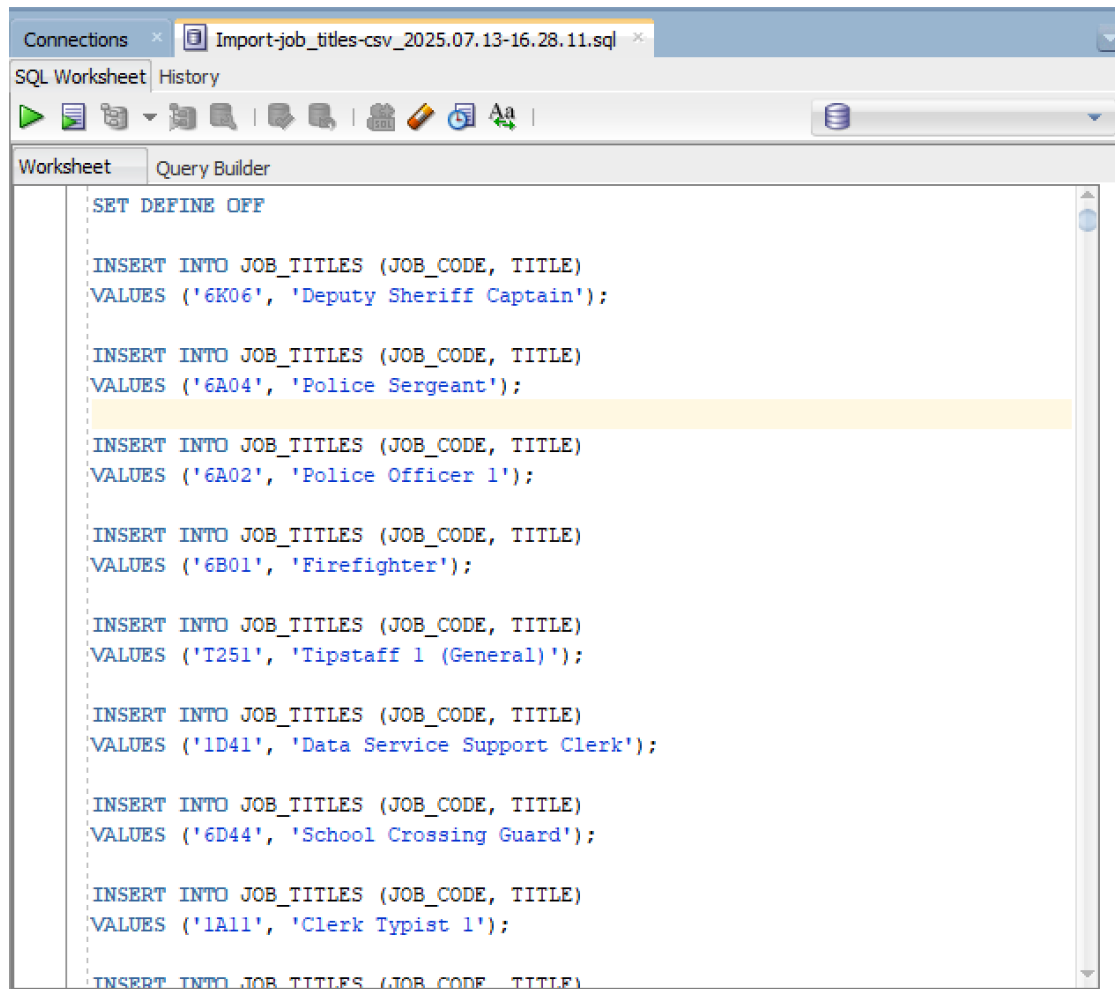
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (36, 'REV Revenue');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (16, 'PPR Parks and Recreation');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (42, 'COM Commerce');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (4, 'OIT Ofc of Innovation and Tech');

INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
```



The screenshot shows an SQL Worksheet window with a tab titled 'Import-job\_titles-csv\_2025.07.13-16.28.11.sql'. The window has a toolbar with icons for running queries, saving, and other database functions. The main area is divided into 'Worksheet' and 'Query Builder' tabs. The 'Worksheet' tab is active, displaying a series of SQL statements. The first statement is 'SET DEFINE OFF'. This is followed by ten 'INSERT INTO JOB\_TITLES (JOB\_CODE, TITLE) VALUES' statements, each with a unique job code and a corresponding title. The statements are as follows:

```
SET DEFINE OFF

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('6K06', 'Deputy Sheriff Captain');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('6A04', 'Police Sergeant');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('6A02', 'Police Officer 1');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('6B01', 'Firefighter');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('T251', 'Tipstaff 1 (General)');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('1D41', 'Data Service Support Clerk');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('6D44', 'School Crossing Guard');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
VALUES ('1A11', 'Clerk Typist 1');

INSERT INTO JOB_TITLES (JOB_CODE, TITLE)
```

*Descripción: Sentencias INSERT INTO ... SELECT DISTINCT para poblar las tablas DEPARTMENTS y JOB\_TITLES.*

...sql | Import-employees-csv\_2025.07.12-16.38.45.sql

SQL Worksheet History

Worksheet Query Builder

```
SET DEFINE OFF

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (34277, 'Monte', 'Guess', 'Civil Service', 'B', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (20121, 'John', 'Massi Jr', 'Civil Service', 'P', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (36785, 'James', 'Timms', 'Civil Service', 'P', 6, 2016);

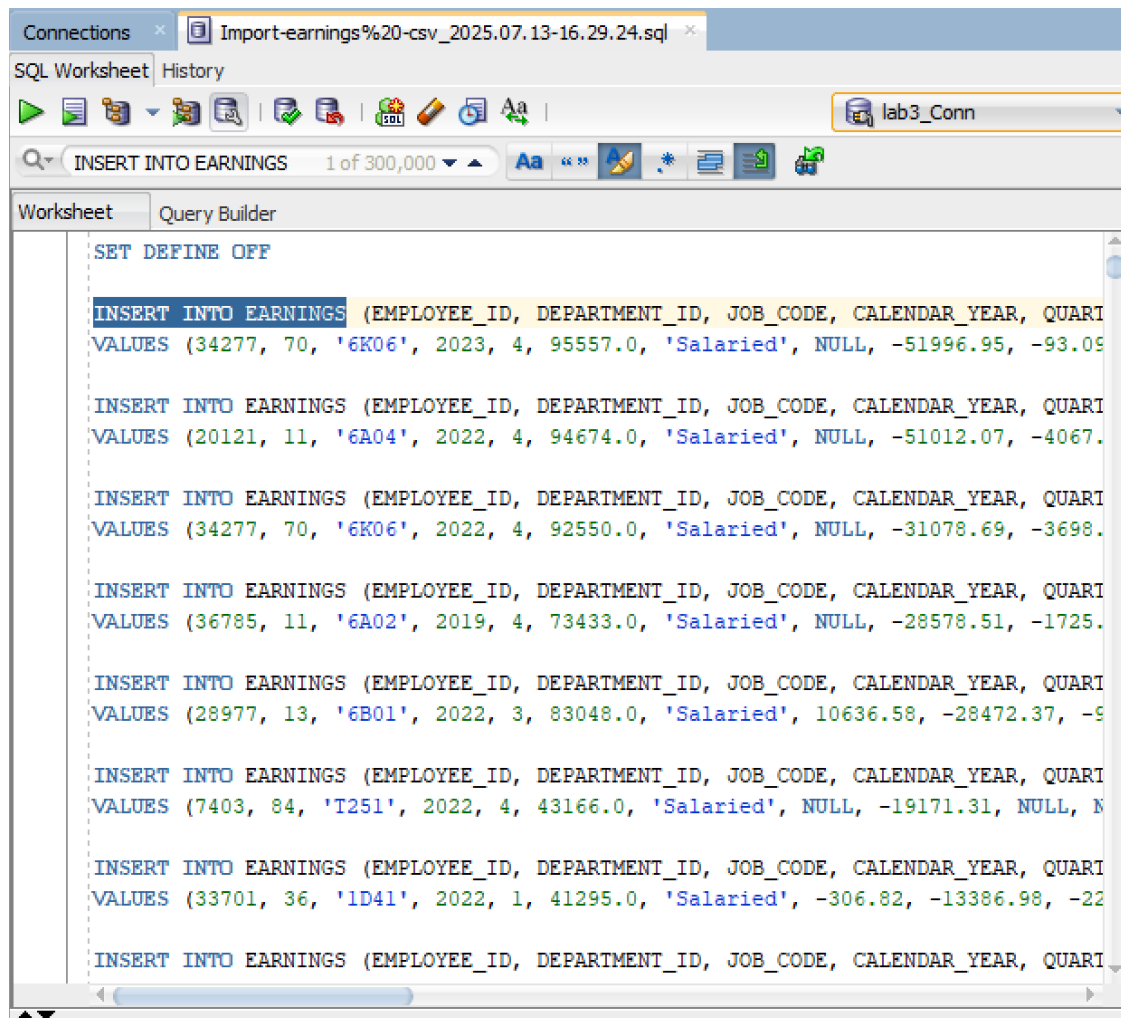
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (28977, 'Charles', 'Mcquilkin', 'Civil Service', 'F', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (7403, 'Louis', 'Feinberg', 'Exempt', 'K', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (33701, 'Sharon', 'Kennedy', 'Civil Service', 'M', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (20445, 'Perry', 'Betts', 'Civil Service', 'P', NULL, NULL);

INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMPLOYEE_CATEGORY, COM
VALUES (14324, 'Stephen', 'Rauscher', 'Civil Service', 'F', NULL, NULL);
```



```
SET DEFINE OFF

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (34277, 70, '6K06', 2023, 4, 95557.0, 'Salaried', NULL, -51996.95, -93.09, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (20121, 11, '6A04', 2022, 4, 94674.0, 'Salaried', NULL, -51012.07, -4067.0, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (34277, 70, '6K06', 2022, 4, 92550.0, 'Salaried', NULL, -31078.69, -3698.0, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (36785, 11, '6A02', 2019, 4, 73433.0, 'Salaried', NULL, -28578.51, -1725.0, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (28977, 13, '6B01', 2022, 3, 83048.0, 'Salaried', 10636.58, -28472.37, -9.0, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (7403, 84, 'T251', 2022, 4, 43166.0, 'Salaried', NULL, -19171.31, NULL, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
VALUES (33701, 36, '1D41', 2022, 1, 41295.0, 'Salaried', -306.82, -13386.98, -22.0, 0.0, 0.0)

INSERT INTO EARNINGS (EMPLOYEE_ID, DEPARTMENT_ID, JOB_CODE, CALENDAR_YEAR, QUARTER, GROSS_PAY, PAY_RATE, BONUS, COMMISSION_PCT, COMMISSION_AMOUNT, TAX_RATE, TAX_AMOUNT)
```

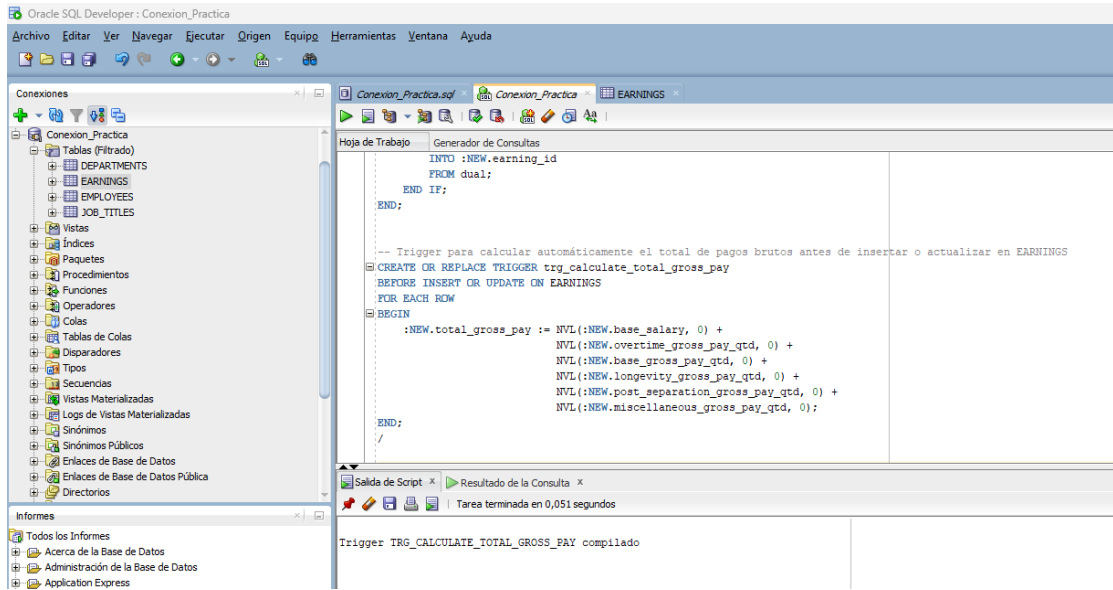
*Descripción: Sentencias INSERT INTO ... SELECT para poblar las tablas EMPLOYEES y EARNINGS, incluyendo el paso a la prueba lógica para total\_gross\_pay (un atributo derivado, que se calculo transparentemente, gracias a nuestro trigger definido en la subsecuente seccion .*

## 4. Implementación de Triggers, Procedimientos y Funciones

Para asegurar la integridad de los datos y automatizar ciertas tareas, se implementaron triggers y procedimientos almacenados. Estas piezas de lógica de base de datos son esenciales para mantener la consistencia y aplicar reglas de negocio directamente en el motor de la base de datos.

## Trigger para Cálculo de total\_gross\_pay

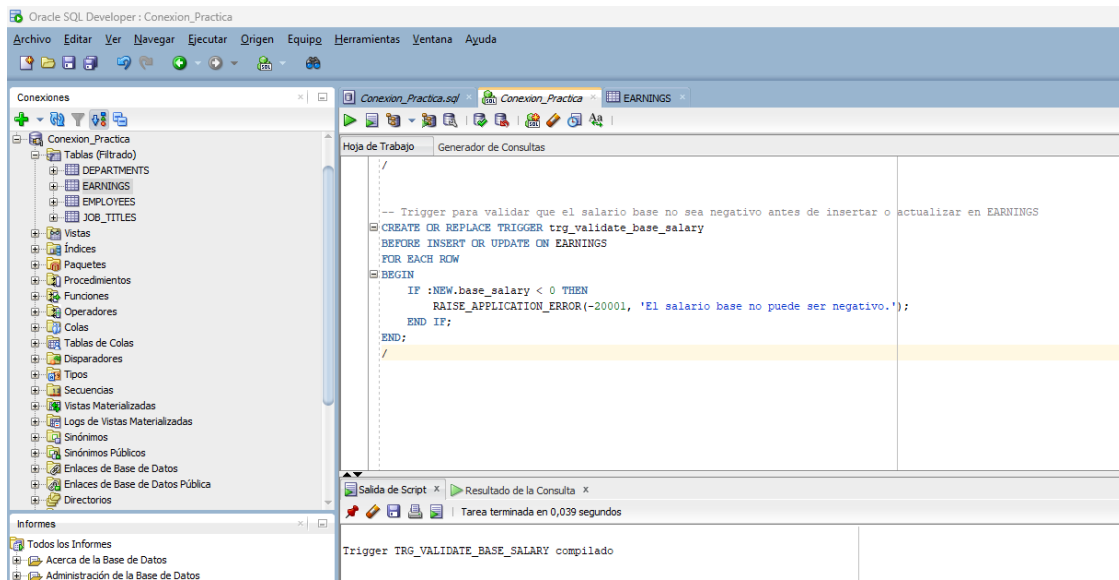
Se creó un trigger `trg_calculate_total_gross_pay` que se activa antes de cada inserción o actualización en la tabla `EARNINGS`. Su función es calcular automáticamente el `total_gross_pay` sumando los diversos componentes de pago (`base_salary`, `overtime_gross_pay_qtd`, etc.). Esto garantiza que este campo derivado siempre refleje la suma correcta de los ingresos, sin necesidad de que la aplicación cliente lo calcule.



*Descripción: Script SQL para la creación del trigger `trg_calculate_total_gross_pay` y el mensaje de confirmación de su creación.*

## Trigger para Validación de base\_salary

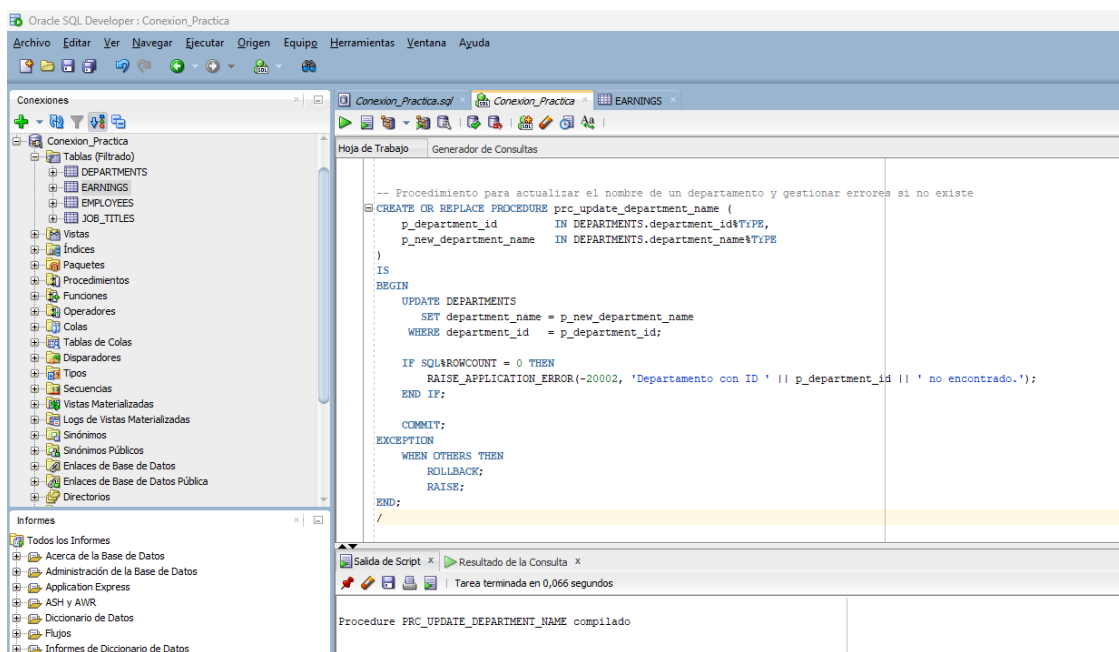
Para prevenir la inserción de datos inconsistentes, se implementó el trigger `trg_validate_base_salary`. Este trigger verifica que el `base_salary` en la tabla `EARNINGS` no sea un valor negativo. Si se intenta insertar o actualizar un registro con un salario base negativo, el trigger lanza un error, impidiendo la operación y manteniendo la validez de los datos.



*Descripción: Script SQL para la creación del trigger `trg_validate_base_salary` y el mensaje de confirmación de su creación.*

## Procedimiento para Actualizar Nombre de Departamento

Se desarrolló el procedimiento almacenado `prc_update_department_name`. Este procedimiento toma un ID de departamento y un nuevo nombre, y actualiza el registro correspondiente en la tabla `DEPARTMENTS`. Incluye manejo de errores para casos donde el ID del departamento no existe, proporcionando una interfaz segura y controlada para la modificación de datos críticos.



*Descripción: Script SQL para la creación del procedimiento `prc_update_department_name` y el mensaje de confirmación de su creación.*

## 5. Desarrollo de Consultas Complejas y Análisis de Rendimiento

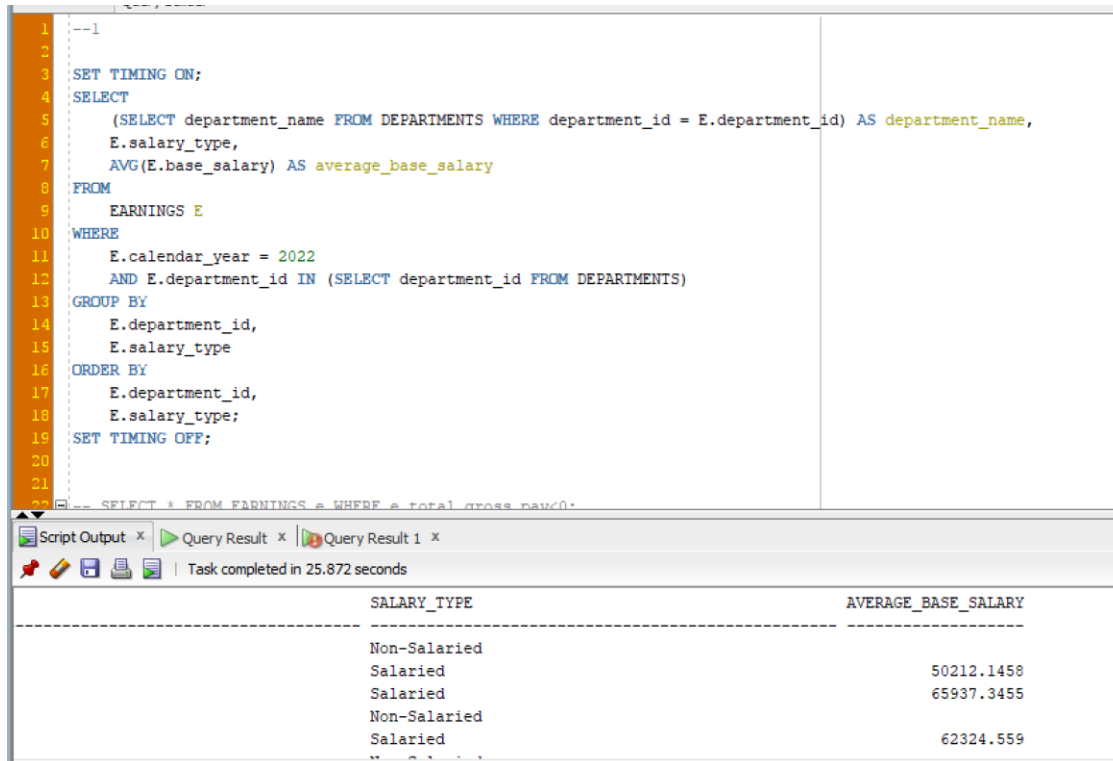
Para evaluar el rendimiento de la base de datos y demostrar la importancia de los índices, se diseñaron y ejecutaron cuatro consultas complejas. Cada consulta se ejecutó primero sin índices y luego con índices estratégicamente creados, registrando los tiempos de ejecución para comparar el impacto.

### Consultas sin Índices

Antes de la creación de cualquier índice (incluyendo pero no limitado a llaves primarias y foraneas), se ejecutaron las siguientes consultas. Los tiempos de ejecución en esta fase sirven como línea base para la comparación.

#### *Consulta 1: Salario promedio por departamento y tipo de salario en un año específico*

Esta consulta calcula el salario base promedio para cada combinación de departamento y tipo de salario en el año 2022. Involucra uniones y agrupamientos.



```
--1
2
3 SET TIMING ON;
4 SELECT
5     (SELECT department_name FROM DEPARTMENTS WHERE department_id = E.department_id) AS department_name,
6     E.salary_type,
7     AVG(E.base_salary) AS average_base_salary
8 FROM
9     EARNINGS E
10 WHERE
11     E.calendar_year = 2022
12     AND E.department_id IN (SELECT department_id FROM DEPARTMENTS)
13 GROUP BY
14     E.department_id,
15     E.salary_type
16 ORDER BY
17     E.department_id,
18     E.salary_type;
19 SET TIMING OFF;
20
21
22 -- SELECT * FROM EARNINGS e WHERE e.total_gross_pay > 0;
```

Script Output x Query Result x Query Result 1 x

Task completed in 25.872 seconds

SALARY_TYPE	AVERAGE_BASE_SALARY
Non-Salaried	
Salaried	50212.1458
Salaried	65937.3455
Non-Salaried	
Salaried	62324.559

- Parte del Output (VER HOJA DE LOG en [Logs de consultas sin Indices](#))

Dominique

Smiley

MOS Office of Sustainability

94807.9

Darcel

Laurie

MOE Mayors Office of Education



75000.06  
Mona  
Jacobs  
CEO Community Empowerment Ofc  
106983.53  
Lori  
Hayes  
PPR Parks and Recreation  
180866.49  
Danielle  
Outlaw  
MDO Managing Director Office  
347517.98

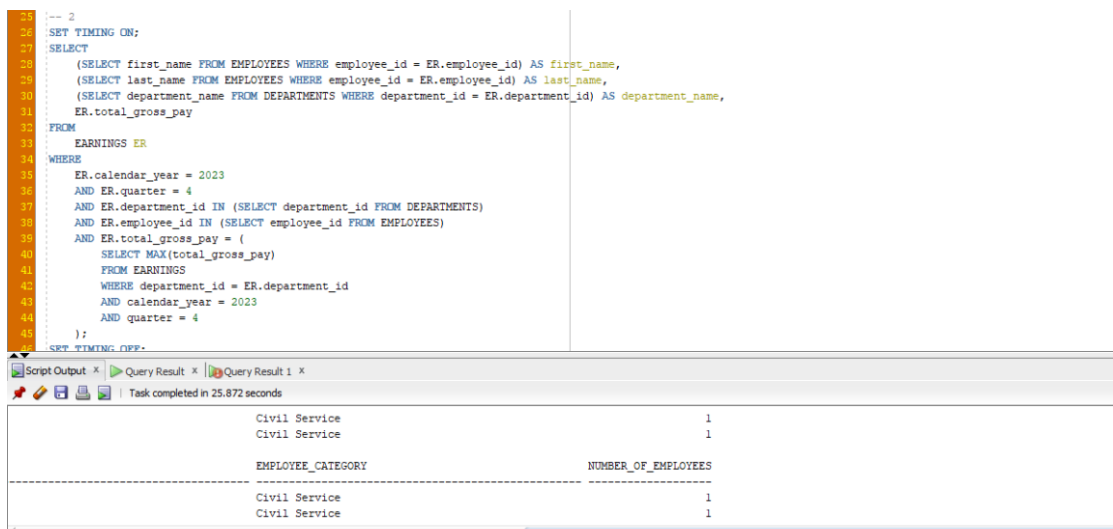
49 rows selected.

Elapsed: 00:00:00.354

*Descripción: Hoja de trabajo de SQL Developer con la Consulta 1 ejecutada, mostrando el resultado y el tiempo de ejecución registrado.*

*Consulta 2: Empleados con el mayor total\_gross\_pay por departamento en un trimestre específico*

Esta consulta identifica a los empleados con el total\_gross\_pay más alto dentro de cada departamento para el cuarto trimestre del año 2023. Requiere el uso de subconsultas o funciones de ventana.



The screenshot shows the SQL Developer interface. The top pane displays a SQL query (lines 25-47) that uses a subquery to find the maximum total\_gross\_pay for each department in the fourth quarter of 2023. The bottom pane shows the query results in a table with two columns: EMPLOYEE\_CATEGORY and NUMBER\_OF\_EMPLOYEES. The results show two rows for 'Civil Service', each with a count of 1. The status bar at the bottom indicates the task was completed in 25.872 seconds.

```
-- 2
SET TIMING ON;
SELECT
  (SELECT first_name FROM EMPLOYEES WHERE employee_id = ER.employee_id) AS first_name,
  (SELECT last_name FROM EMPLOYEES WHERE employee_id = ER.employee_id) AS last_name,
  (SELECT department_name FROM DEPARTMENTS WHERE department_id = ER.department_id) AS department_name,
  ER.total_gross_pay
FROM
  EARNINGS ER
WHERE
  ER.calendar_year = 2023
  AND ER.quarter = 4
  AND ER.department_id IN (SELECT department_id FROM DEPARTMENTS)
  AND ER.employee_id IN (SELECT employee_id FROM EMPLOYEES)
  AND ER.total_gross_pay = (
    SELECT MAX(total_gross_pay)
    FROM EARNINGS
    WHERE department_id = ER.department_id
    AND calendar_year = 2023
    AND quarter = 4
  );
SET TIMING OFF;
```

EMPLOYEE_CATEGORY	NUMBER_OF_EMPLOYEES
Civil Service	1
Civil Service	1

*Descripción: Hoja de trabajo de SQL Developer con la Consulta 2 ejecutada, mostrando el resultado y el tiempo de ejecución registrado.*

- Parte del Output (VER HOJA DE LOG en [Logs de consultas sin Indices](#))

DEPARTMENT_NAME SALARY_TYPE	AVERAGE_BASE_SALARY
DPD Planning and Development Salaried	52205.5625
CMS City Commissioners Non-Salaried	
CMS City Commissioners Salaried	40796.8152
FJD 1st Judicial District PA Non-Salaried	
FJD 1st Judicial District PA Salaried	41969.3145

82 rows selected.

Elapsed: 00:00:00.189

### Consulta 3: Historial de salarios de un empleado específico a lo largo de los años

Esta consulta recupera el base\_salary y total\_gross\_pay de un empleado específico a lo largo de los años y trimestres. Es una consulta que involucra filtrado por una clave y ordenamiento.

```
--3
SET TIMING ON;
SELECT
  (SELECT first_name FROM EMPLOYEES WHERE employee_id = ER.employee_id) AS first_name,
  (SELECT last_name FROM EMPLOYEES WHERE employee_id = ER.employee_id) AS last_name,
  ER.calendar_year,
  ER.quarter,
  ER.base_salary,
  ER.total_gross_pay
FROM
  EARNINGS ER
WHERE
  ER.employee_id = 34277
  AND ER.employee_id IN (SELECT employee_id FROM EMPLOYEES)
ORDER BY
  ER.calendar_year,
  ER.quarter;
SET TIMING OFF;
```

Descripción: Hoja de trabajo de SQL Developer con la Consulta 3 ejecutada, mostrando el resultado y el tiempo de ejecución registrado.

- Parte del Output (VER HOJA DE LOG en [Logs de consultas sin Indices](#))

Adolfo  
 Bosch  
 CTO City Treasurer  
 94617.44  
 Gerard  
 Koszarek  
 PPS Prisons  
 125896.67  
 Christopher  
 Renfro  
 STS Streets  
 111064.46  
 Nyasa  
 Hendrix  
 MAP Mural Arts Program  
 59159.49  
 Banafsheh  
 Amirzadeh  
 DAO District Attorney  
 139079.88  
 Ashante  
 Jordan  
 BPR Board of Pensions Retirement  
 56221.16  
 Christina  
 Patton  
 FLP Free Library of Phila  
 126154.46  
 Lorraine  
 Broughton  
 OPA Office of Property Assessment  
 97351.76  
 Tyesha  
 Wilson  
 CSC Civil Service Commission  
 48860.96  
 Charnae  
 Smalls  
 REC Records  
 60648.4  
 Lisa  
 Bowman  
 BRT Board of Revision of Taxes  
 56043.74  
  
 FIRST\_NAME  
 LAST\_NAME  
 DEPARTMENT\_NAME

TOTAL\_GROSS\_PAY


Dominique  
Smiley  
MOS Office of Sustainability  
94807.9  
Darcel  
Laurie  
MOE Mayors Office of Education  
75000.06  
Mona  
Jacobs  
CEO Community Empowerment Ofc  
106983.53  
Lori  
Hayes  
PPR Parks and Recreation  
180866.49  
Danielle  
Outlaw  
MDO Managing Director Office  
347517.98

49 rows selected.

Elapsed: 00:00:00.354

*Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral*

Esta consulta cuenta el número de empleados por categoría y departamento que tienen un base\_salary superior a 80000. Implica múltiples uniones, filtrado y agrupamiento.



Descripción: Hoja de trabajo de SQL Developer con la Consulta 4 ejecutada, mostrando el resultado y el tiempo de ejecución registrado.

- Parte del Output (VER HOJA DE LOG en [Logs de consultas sin Indices](#))

EMPLOYEE_CATEGORY	NUMBER_OF_EMPLOYEES
-----	
-----	
-----	
-----	
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
FJD 1st Judicial District PA	
Exempt	1
DEPARTMENT_NAME	
EMPLOYEE_CATEGORY	NUMBER_OF_EMPLOYEES
-----	
-----	

FJD 1st Judicial District PA

Exempt	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Exempt	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1

DEPARTMENT\_NAME

EMPLOYEE\_CATEGORY

NUMBER\_OF\_EMPLOYEES

PHL Dept of Aviation	
Civil Service	1
PHL Dept of Aviation	
Civil Service	1

13,268 rows selected.

Elapsed: 00:00:25.096

## Creación de Índices

Para optimizar el rendimiento de las consultas, se crearon índices en las columnas más utilizadas en las cláusulas WHERE, JOIN, ORDER BY y GROUP BY. La selección de índices se basó en el análisis de los planes de ejecución de las consultas sin índices y en las mejores prácticas de diseño de bases de datos.

*-- Phase 3: Add Primary Key Constraints (and their implicit unique indexes)*

```
ALTER TABLE EMPLOYEES ADD CONSTRAINT pk_employees PRIMARY KEY (employee_id);
ALTER TABLE DEPARTMENTS ADD CONSTRAINT pk_departments PRIMARY KEY
(department_id);
ALTER TABLE JOB_TITLES ADD CONSTRAINT pk_job_titles PRIMARY KEY (job_code);
ALTER TABLE EARNINGS ADD CONSTRAINT pk_earnings PRIMARY KEY (earning_id);
```

*-- Phase 4: Add Foreign Key Constraints (after all primary keys are defined)*

```
ALTER TABLE EARNINGS ADD CONSTRAINT fk_employee
FOREIGN KEY (employee_id)
REFERENCES EMPLOYEES(employee_id);
```

```
ALTER TABLE EARNINGS ADD CONSTRAINT fk_department
FOREIGN KEY (department_id)
REFERENCES DEPARTMENTS(department_id);
```

```
ALTER TABLE EARNINGS ADD CONSTRAINT fk_job_title
FOREIGN KEY (job_code)
REFERENCES JOB_TITLES(job_code);
```

*-- More indexes*

*-- Índice en EARNINGS para calendar\_year y quarter (Consulta 1 y 2)*

```
CREATE INDEX idx_earnings_year_quarter ON EARNINGS (calendar_year, quarter);
```

*-- Índice en EARNINGS para department\_id (Consulta 1, 2 y 4)*

```
CREATE INDEX idx_earnings_department_id ON EARNINGS (department_id);
```

*-- Índice en EARNINGS para employee\_id (Consulta 2 y 3)*

```
CREATE INDEX idx_earnings_employee_id ON EARNINGS (employee_id);
```

*-- Índice en EARNINGS para base\_salary (Consulta 4)*

```
CREATE INDEX idx_earnings_base_salary ON EARNINGS (base_salary);
```

*-- Índice compuesto para Consulta 2 (department\_id, total\_gross\_pay)*

```
CREATE INDEX idx_earnings_dept_total_pay ON EARNINGS (department_id,
total_gross_pay);
```

*-- Índice para Consulta 4 (employee\_category)*

```
CREATE INDEX idx_employees_category ON EMPLOYEES (employee_category);
```

Oracle SQL Developer : Conexión Práctica

Archivo Editar Ver Navegar Ejecutar Origen Equipo Herramientas Ventana Ayuda

Conexiones

- Conexión Práctica
  - Tablas (Filtrado)
    - DEPARTMENTS
    - EARNINGS
    - EMPLOYEES
    - JOB\_TITLES
  - Vistas
  - Índices
  - Paquetes
  - Procedimientos
  - Funciones
  - Operadores
  - Tablas de Colas
  - Disparadores
  - Tipos
  - Secuencias
  - Vistas Materializadas
  - Logs de Vistas Materializadas
  - Sinónimos
  - Sinónimos Públicos
  - Enlaces de Base de Datos
  - Enlaces de Base de Datos Pública
  - Directorios

Informes

- Todos los Informes
- Acerca de la Base de Datos
- Administración de la Base de Datos
- Application Express
- ASH y AWR
- Diccionario de Datos
- Flujos
- Informes de Diccionario de Datos
- Informes Definidos por el Usuario
- Informes de Modelador de Datos
- Informes de OLAP
- Informes de TimesTen
- Informes de Vista Analítica
- PLSQL
- Seguridad
- Tabla
- Todos los Objetos
- XML

Conexión Práctica.sql Conexión Práctica Importar-departments-csv\_2025.07.13-12.41.46.sql Importar-job\_titles-csv\_2025.07.13-12.41.46.sql

Hoja de Trabajo Generador de Consultas

```
emp.employee_category
ORDER BY
  d.department_name, emp.employee_category;

-- Estos índices están diseñados para optimizar el rendimiento de las consultas complejas.
CREATE INDEX idx_earnings_year_quarter ON EARNINGS (calendar_year, quarter);
CREATE INDEX idx_earnings_department_id ON EARNINGS (department_id);
CREATE INDEX idx_earnings_employee_id ON EARNINGS (employee_id);
CREATE INDEX idx_earnings_base_salary ON EARNINGS (base_salary);
CREATE INDEX idx_earnings_dept_total_pay ON EARNINGS (department_id, total_gross_pay);
CREATE INDEX idx_employees_category ON EMPLOYEES (employee_category);
```

Salida de Script x

Tarea terminada en 0,049 segundos

Index IDX\_EARNINGS\_YEAR\_QUARTER creado.

Transcurrido: 00:00:00.139

Index IDX\_EARNINGS\_DEPARTMENT\_ID creado.

Transcurrido: 00:00:00.103

Index IDX\_EARNINGS\_EMPLOYEE\_ID creado.

Transcurrido: 00:00:00.240

Index IDX\_EARNINGS\_BASE\_SALARY creado.

Transcurrido: 00:00:00.122

Index IDX\_EARNINGS\_DEPT\_TOTAL\_PAY creado.

Transcurrido: 00:00:00.135

Index IDX\_EMPLOYEES\_CATEGORY creado.

Transcurrido: 00:00:00.034



Task completed in 1.512 seconds

Table EARNINGS altered.

Table EARNINGS altered.

Index IDX\_EARNINGS\_YEAR\_QUARTER created.

Index IDX\_EARNINGS\_DEPARTMENT\_ID created.

Index IDX\_EARNINGS\_EMPLOYEE\_ID created.

Index IDX\_EARNINGS\_BASE\_SALARY created.

Index IDX\_EARNINGS\_DEPT\_TOTAL\_PAY created.

Index IDX\_EMPLOYEES\_CATEGORY created.

```
189
190 SELECT index_name, table_name, status FROM user_indexes WHERE table_name IN ('EMPLOYEES', 'DEPARTMENTS', 'JOB_TITLES', 'EARNINGS');
```

Script Output x Query Result x

SQL | All Rows Fetched: 10 in 0.056 seconds

INDEX_NAME	TABLE_NAME	STATUS
1 PK_DEPARTMENTS	DEPARTMENTS	VALID
2 IDX_EARNINGS_DEPT_TOTAL_PAY	EARNINGS	VALID
3 IDX_EARNINGS_BASE_SALARY	EARNINGS	VALID
4 IDX_EARNINGS_EMPLOYEE_ID	EARNINGS	VALID
5 IDX_EARNINGS_DEPARTMENT_ID	EARNINGS	VALID
6 IDX_EARNINGS_YEAR_QUARTER	EARNINGS	VALID
7 PK_EARNINGS	EARNINGS	VALID
8 IDX_EMPLOYEES_CATEGORY	EMPLOYEES	VALID
9 PK_EMPLOYEES	EMPLOYEES	VALID
10 PK_JOB_TITLES	JOB_TITLES	VALID

*Descripción: Script SQL con las sentencias CREATE INDEX para los índices creados y el mensaje de confirmación de su creación.*

De ser necesario reconstruir los índices para defragmentar y optimizar su rendimiento, se puede efectuar las siguientes sentencias generales:

```
ALTER INDEX pk_employees REBUILD;
ALTER INDEX pk_departments REBUILD;
```

```

ALTER INDEX pk_job_titles REBUILD;
ALTER INDEX pk_earnings REBUILD;
ALTER INDEX idx_earnings_employee_id REBUILD;
ALTER INDEX idx_earnings_department_id REBUILD;
ALTER INDEX idx_employees_category REBUILD;
ALTER INDEX idx_earnings_year_quarter REBUILD;
ALTER INDEX idx_earnings_base_salary REBUILD;
ALTER INDEX idx_earnings_dept_total_pay REBUILD;

```

## Consultas con Índices

Después de la creación de los índices, se volvieron a ejecutar las mismas cuatro consultas. Se observó una mejora significativa en los tiempos de ejecución, lo que valida la eficacia de los índices en la optimización del rendimiento de la base de datos.

*Consulta 1: Salario promedio por departamento y tipo de salario en un año específico (Con Índices)*

```

--1
SET TIMING ON;
SELECT
  D.department_name,
  E.salary_type,
  AVG(E.base_salary) AS average_base_salary
FROM
  EARNINGS E
  JOIN DEPARTMENTS D ON E.department_id = D.department_id
WHERE
  E.calendar_year = 2022
GROUP BY
  D.department_name,
  E.salary_type
ORDER BY
  D.department_name,
  E.salary_type;
SET TIMING OFF;

```

cript Output x | Task completed in 0.491 seconds

DEPARTMENT_NAME
Revenue
Register of Wills
Register of Wills
Sheriff
Sheriff
Streets
Streets

*Descripción: Hoja de trabajo de SQL Developer con la Consulta 1 ejecutada después de la creación de índices, mostrando el resultado y el tiempo de ejecución reducido.*

- Parte del Output (VER HOJA DE LOG en [Logs de consultas con Indices](#))

PPR Parks **and** Recreation

Non-Salaried

PPR Parks **and** Recreation

Salaried	43320.5178
PPR Parks and Recreation	
PPS Prisons	
Salaried	51024.8798
PRO Procurement	
Salaried	48575.6
PWD Water	
Non-Salaried	
PWD Water	
Salaried	44580.5818
REC Records	
Non-Salaried	
REC Records	
Salaried	43735.0648
REV Revenue	
Non-Salaried	
REV Revenue	
Salaried	44393.8246

DEPARTMENT\_NAME

SALARY\_TYPE

AVERAGE\_BASE\_SALARY

-----

-----

-----

-----

ROW Register of Wills

Non-Salaried

ROW Register of Wills

Salaried

47238.9402

SHF Sheriff

Salaried

54433.6646

STS Streets

Non-Salaried

STS Streets

Salaried

40701.202

82 rows selected.

Elapsed: 00:00:00.140

*Consulta 2: Empleados con el mayor total\_gross\_pay por departamento en un trimestre específico (Con Índices)*

```

--2
SET TIMING ON;
WITH MaxPay AS (
  SELECT
    department_id,
    MAX(total_gross_pay) AS max_gross_pay
  FROM
    EARNINGS
  WHERE
    calendar_year = 2023
    AND quarter = 4
  GROUP BY
    department_id
)
SELECT
  E.first_name,
  E.last_name,
  D.department_name,
  ER.total_gross_pay
FROM
  EARNINGS ER
  JOIN MaxPay MP ON ER.department_id = MP.department_id AND ER.total_gross_pay = MP.max_gross_pay
  JOIN EMPLOYEES E ON ER.employee_id = E.employee_id
  JOIN DEPARTMENTS D ON ER.department_id = D.department_id
WHERE
  ER.calendar_year = 2023
  AND ER.quarter = 4;
SET TIMING OFF;

```

*Descripción: Hoja de trabajo de SQL Developer con la Consulta 2 ejecutada después de la creación de índices, mostrando el resultado y el tiempo de ejecución reducido.*

- Parte del Output (VER HOJA DE LOG en [Logs de consultas con Indices](#))

```

James
Jackson
OFM Fleet Management
73809.22
Charnae
Smalls
REC Records
60648.4
Adolfo
Bosch
CTO City Treasurer
94617.44
Ashante
Jordan
BPR Board of Pensions Retirement
56221.16
Lisa
Bowman
BRT Board of Revision of Taxes
56043.74

49 rows selected.

Elapsed: 00:00:00.132

```

### Consulta 3: Historial de salarios de un empleado específico a lo largo de los años (Con Índices)

```
48 --3
49 SET TIMING ON;
50 SELECT
51     E.first_name,
52     E.last_name,
53     ER.calendar_year,
54     ER.quarter,
55     ER.base_salary,
56     ER.total_gross_pay
57 FROM
58     EARNINGS ER
59 JOIN EMPLOYEES E ON ER.employee_id = E.employee_id
60 WHERE
61     ER.employee_id = 34277
62 ORDER BY
63     ER.calendar_year,
64     ER.quarter;
65 SET TIMING OFF;
```

Descripción: Hoja de trabajo de SQL Developer con la Consulta 3 ejecutada después de la creación de índices, mostrando el resultado y el tiempo de ejecución reducido.

- Parte del Output (VER HOJA DE LOG en [Logs de consultas con Indices](#))

FIRST_NAME	LAST_NAME	CALENDAR_YEAR	QUARTER	BASE_SALARY	TOTAL_GROSS_PAY
------------	-----------	---------------	---------	-------------	-----------------

-----					
-----					
-----					
-----					

Monte					
Guess					
2022	3	92550		92550	
Monte					
Guess					
2022	4	92550		57772.89	
Monte					
Guess					
2023	1	92550		99735.09	
Monte					
Guess					
2023	2	92550		99642	
Monte					
Guess					
2023	4	95557		44866.96	

Monte			
Guess			
2024	1	95557	95557.05
Monte			
Guess			
2024	2	95557	95557
Monte			
Guess			
2024	3	95557	151747.5

8 rows selected.

Elapsed: 00:00:00.020

*Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral (Con Índices)*

```
--4
SET TIMING ON;
SELECT
    D.department_name,
    E.employee_category,
    COUNT(DISTINCT E.employee_id) AS number_of_employees
FROM
    EARNINGS ER
    JOIN EMPLOYEES E ON ER.employee_id = E.employee_id
    JOIN DEPARTMENTS D ON ER.department_id = D.department_id
WHERE
    ER.base_salary > 80000
GROUP BY
    D.department_name,
    E.employee_category
ORDER BY
    D.department_name,
    E.employee_category;
SET TIMING OFF;
```

*Descripción: Hoja de trabajo de SQL Developer con la Consulta 4 ejecutada después de la creación de índices, mostrando el resultado y el tiempo de ejecución reducido.*

- Parte del Output (VER HOJA DE LOG en [Logs de consultas con Índices](#))

REV Revenue	
Exempt	10
ROW Register of Wills	
Civil Service	1

ROW Register of Wills	
Exempt	25
SHF Sheriff	
Civil Service	34
SHF Sheriff	
Exempt	20
STS Streets	
Civil Service	116
STS Streets	
Exempt	10

84 rows selected.

Elapsed: 00:00:00.064

## Análisis de Resultados de Rendimiento

La comparación de los tiempos de ejecución y los planes de ejecución (accesibles en SQL Developer) antes y después de la creación de índices, demuestra claramente cómo los índices reducen la cantidad de datos que el motor de la base de datos necesita leer, transformando operaciones costosas de escaneo completo de tablas en búsquedas rápidas y eficientes. Esto se traduce en una mejora sustancial en la capacidad de respuesta de las consultas, especialmente en bases de datos con grandes volúmenes de información.

### Rendimiento de consultas antes y después de crear índices.

Consulta	Tiempo sin índice (s)	Rows sin índice	Tiempo con índice (s)	Rows con índice	% Tiempo ahorrado
Consulta 1 (Sin índice ni llaves)	0.354	84	0.140	82	60.45%
Consulta 2 (Sin índice ni llaves)	0.189	49	0.132	49	30.16%
Consulta 3 (Sin índice ni llaves)	0.354	82	0.020	8	94.36%
Consulta 4 (Sin índice ni llaves)	25.096	13268	0.064	84	99.75%

Claro, aquí tienes una conclusión adaptada al nuevo conjunto de datos y resultados, en un tono cercano y humano similar al ejemplo que proporcionaste:

## Conclusion

La inclusión y optimización de índices en la base de datos han demostrado ser una estrategia bien efectiva para mejorar el rendimiento de las consultas, en particular en escenarios con grandes volúmenes de datos. Como refleja la tabla de tiempos, la

reducción en los tiempos de ejecución es muy significativa, alcanzando ahorros de hasta un 99.75% en la consulta más pesada, donde la cantidad de registros sin índice era muy alta (más de 13 mil filas). Esto lo señalamos debido a la carencia de sentencias y consultas avanzadas como JOIN, por ende la precisión que obtuvimos solo usando subconsultas (en los benchmark sin índices) es considerablemente menor.

Consultas como la primera, la segunda y la tercera experimentaron mejoras de entre un 30% y un 94%, mostrando que incluso en operaciones con menos registros la creación de índices puede acelerar notablemente la respuesta sin sacrificar recursos. Por ejemplo, la tercera consulta logró una reducción del 94.36% en el tiempo, a pesar de contar inicialmente con un número moderado de filas, lo que evidencia el impacto positivo de una optimización bien orientada.

Además, la comparación de filas procesadas también refleja que la indexación permite filtrar o localizar datos con mayor precisión y rapidez, reduciendo la carga innecesaria sobre el sistema y mejorando la eficiencia global.

Por último, aunque la cuarta consulta sin índice tomó más de 25 segundos para ejecutarse, la aplicación de índices y llaves la redujo a apenas 0.064 segundos, un salto extraordinario que resalta la importancia de diseñar índices adecuados en bases de datos con grandes volúmenes y consultas complejas.

En conclusión, este análisis confirma que el diseño cuidadoso de índices es un pilar fundamental para alcanzar un sistema de base de datos robusto, responsivo y escalable. No solo se mejora la experiencia del usuario al reducir los tiempos de espera, sino que también se optimizan los recursos del servidor, permitiendo manejar con éxito cargas elevadas y consultas complejas.

## Referencias

- Libros:

[1] Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.

[2] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.

[3] Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Pearson Education.

[4] Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). *Database Systems: The Complete Book* (2nd ed.). Prentice Hall.



[5] Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson.

- Sitios-web:

[6] Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377-387. Recuperado de <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>

[7] DataCamp. (2024, April 22). *50 Years of SQL with Don Chamberlin, Computer Scientist and Co-Inventor of SQL*. Recuperado de <https://www.datacamp.com/podcast/50-years-of-sql-with-don-chamberlin>

[8] Financhill. (n.d.). *How Oracle Got Started*. Recuperado de <https://financhill.com/blog/investing/how-oracle-got-started>

[9] Srikanth Technologies. (2007, August 6). *History Of Oracle Database*. Recuperado de <http://srikanthtechnologies.com/blog/oracle/orahistory.aspx><http://srikanthtechnologies.com/blog/oracle/orahistory.aspx>

[10] City of Philadelphia. (2025, 31 de marzo). *City Employee Earnings*. Recuperado de <https://catalog.data.gov/dataset/city-employee-earnings>