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

## Integrantes

* Marco Abarca R.
* Alex Jadan
* Luis Sarango

## Índice

* [Introducción](#introducción)
* [1. Análisis del Dataset y Diseño del Modelo Relacional](#X9c725994ed85898bbb7395ea960aa83bf05b78f)
* [2. Creación de Modelos y Scripts SQL](#Xb930027d226bda81b240a18a58f28dba409da10)
  + [Modelo Conceptual](#modelo-conceptual)
  + [Modelo Lógico](#modelo-lógico)
  + [Modelo Físico](#modelo-físico)
* [3. Importación de Datos CSV](#Xc1445ea910267011a1a04264208efb9d9f54f06)
* [4. Implementación de Triggers, Procedimientos y Funciones](#Xae04feac7e31f6408226c85be262b137776bf6b)
  + [Trigger para Cálculo de total\_gross\_pay](#trigger-para-cálculo-de-totalgrosspay)
  + [Trigger para Validación de base\_salary](#trigger-para-validación-de-basesalary)
  + [Procedimiento para Actualizar Nombre de Departamento](#X561a9ab056618af4fd2e646a04496618bf5cd16)
* [5. Desarrollo de Consultas Complejas y Análisis de Rendimiento](#X5784f6d6264069e8b5ab3f58fd8cf6ca49f8f11)
  + [Consultas sin Índices](#consultas-sin-índices)
    - [Consulta 1: Salario promedio por departamento y tipo de salario en un año específico](#X2dfcccc6b84e455b030ecada611861ce280aa71)
    - [Consulta 2: Empleados con el mayor total\_gross\_pay por departamento en un trimestre específico](#X22a4ddf3abe7f39f7110d736467c6bb482835c7)
    - [Consulta 3: Historial de salarios de un empleado específico a lo largo de los años](#X0b1ba19f5caffd6556b2bd66c2cfec80e2c4307)
    - [Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral](#Xaea3a05cd8af6a52bea920329d5daff6035fb77)
  + [Creación de Índices](#creación-de-índices)
  + [Consultas con Índices](#consultas-con-índices)
    - [Consulta 1: Salario promedio por departamento y tipo de salario en un año específico (Con Índices)](#X6ab24f89766fba28678348d6538a9ca45c38318)
    - [Consulta 2: Empleados con el mayor total\_gross\_pay por departamento en un trimestre específico (Con Índices)](#Xb0460f221406b57bcb1346ecf9a1b74a4ec9353)
    - [Consulta 3: Historial de salarios de un empleado específico a lo largo de los años (Con Índices)](#X5074c40d9ebde33021548fbfce92f65e2d20704)
    - [Consulta 4: Conteo de empleados por categoría y departamento con salario base superior a un umbral (Con Índices)](#X2c3f318eb0bf7e4156da513778d5e08f2ffe7a4)
  + [Análisis de Resultados de Rendimiento](#análisis-de-resultados-de-rendimiento)
* [Conclusión](#conclusión)
* [Referencias Biográficas y Contexto Histórico](#Xd866bc28cd2cf29891ba53fae04c3f354b95a13)
  + [Edgar F. Codd: El Padre del Modelo Relacional](#Xb0bb9a7c3ae195cda75d2e27c61b04d263df8f6)
  + [Los Creadores de SQL: Chamberlin y Boyce](#los-creadores-de-sql-chamberlin-y-boyce)
  + [Los Fundadores de Oracle: Ellison, Miner y Oates](#Xd7adb34bdd144f876e04a1fc63b87d6fc19c73c)
  + [Referencias](#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.

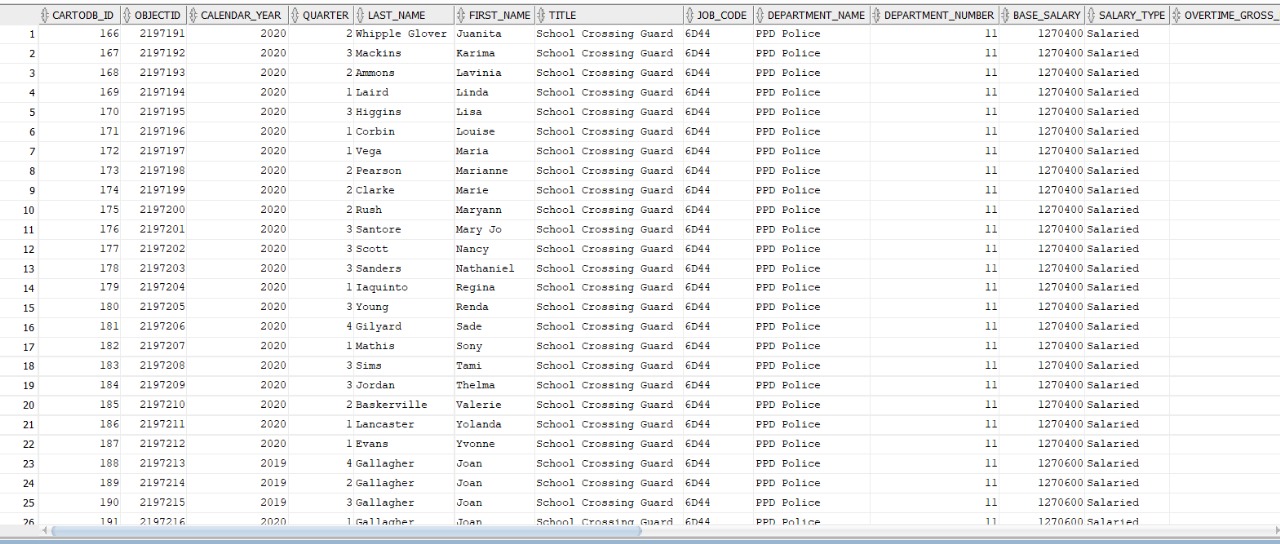
## 1. Análisis del Dataset y Diseño del Modelo Relacional

El primer paso crucial fue comprender la estructura del archivo que escojimos: 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.

  
*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 PRIMARY KEY,  
 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 PRIMARY KEY,  
 department\_name VARCHAR2(200)  
);  
  
CREATE TABLE JOB\_TITLES (  
 job\_code VARCHAR2(20) PRIMARY KEY,  
 title VARCHAR2(200)  
);  
  
CREATE TABLE EARNINGS (  
 earning\_id NUMBER PRIMARY KEY,  
 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),  
 CONSTRAINT fk\_employee  
 FOREIGN KEY (employee\_id)  
 REFERENCES EMPLOYEES(employee\_id),  
 CONSTRAINT fk\_department  
 FOREIGN KEY (department\_id)  
 REFERENCES DEPARTMENTS(department\_id),  
 CONSTRAINT fk\_job\_title  
 FOREIGN KEY (job\_code)  
 REFERENCES JOB\_TITLES(job\_code)  
);  
  
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*

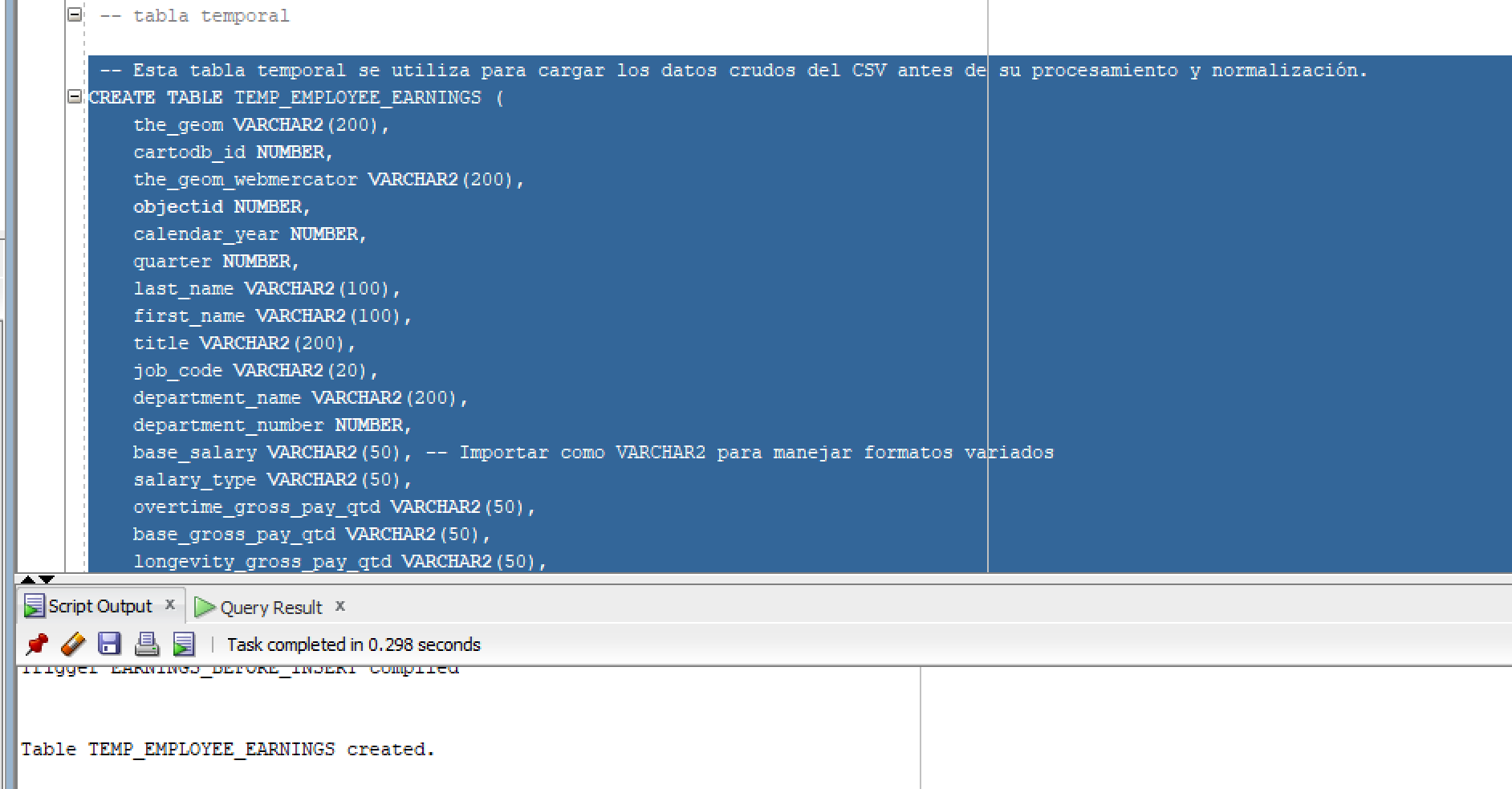
*Descripción: Creacion de la Tabla EARNINGS*

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

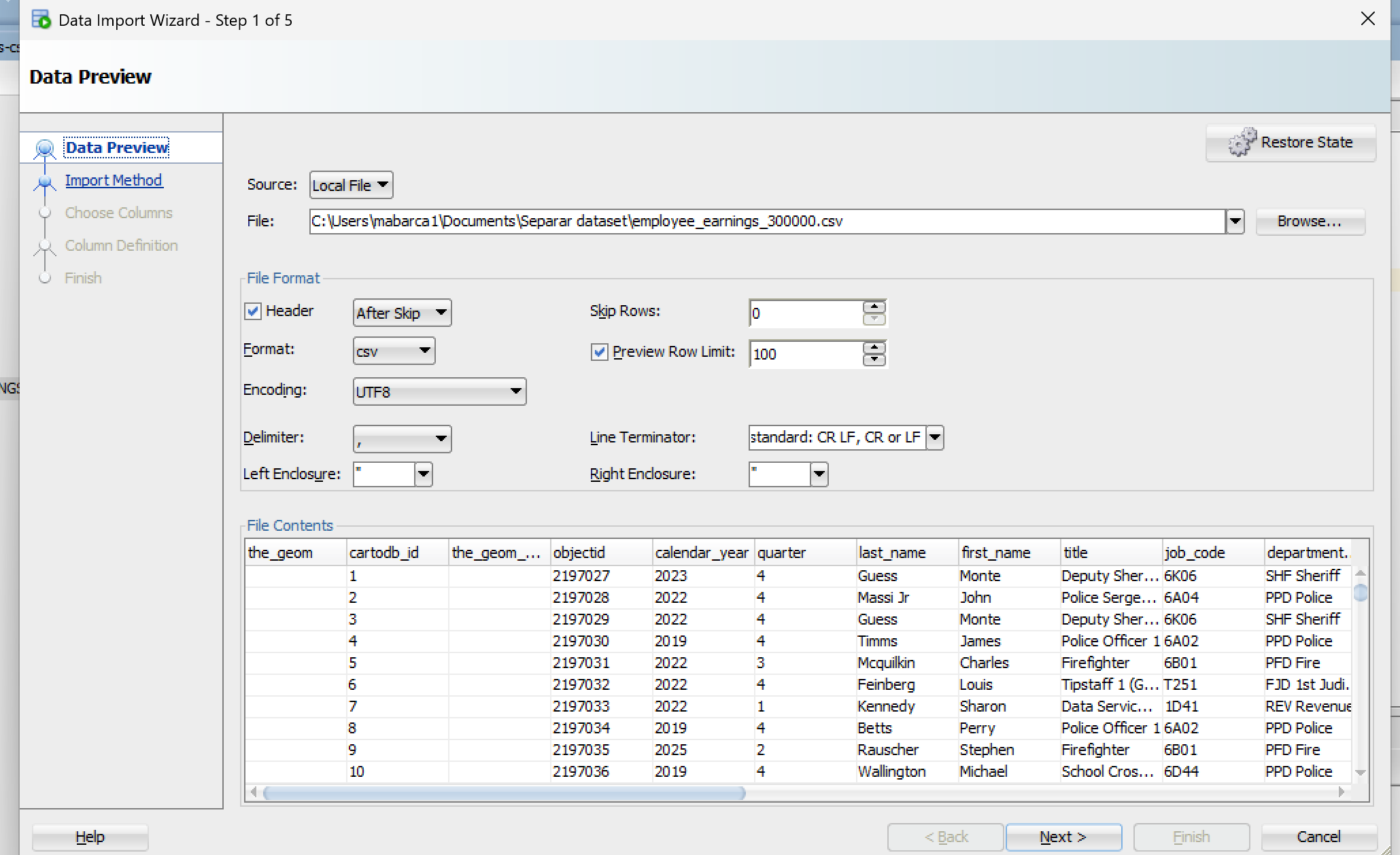
## 3. Importación de Datos CSV

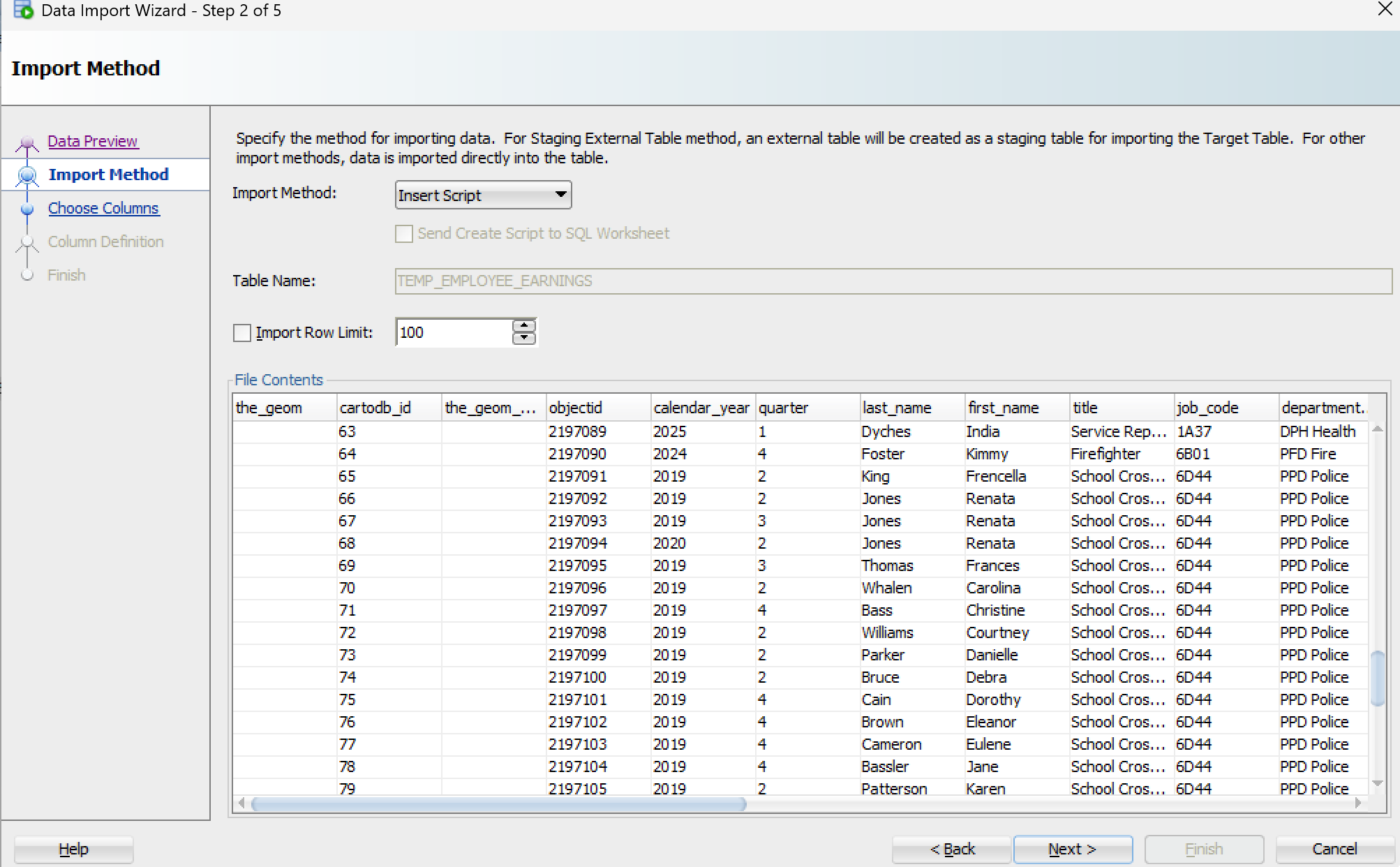
La importación de 300,000 registros desde el CSV a la estructura relacional diseñada fue un paso crítico. Se optó por un enfoque en dos fases para manejar las dependencias de claves foráneas y asegurar la correcta normalización de los datos:

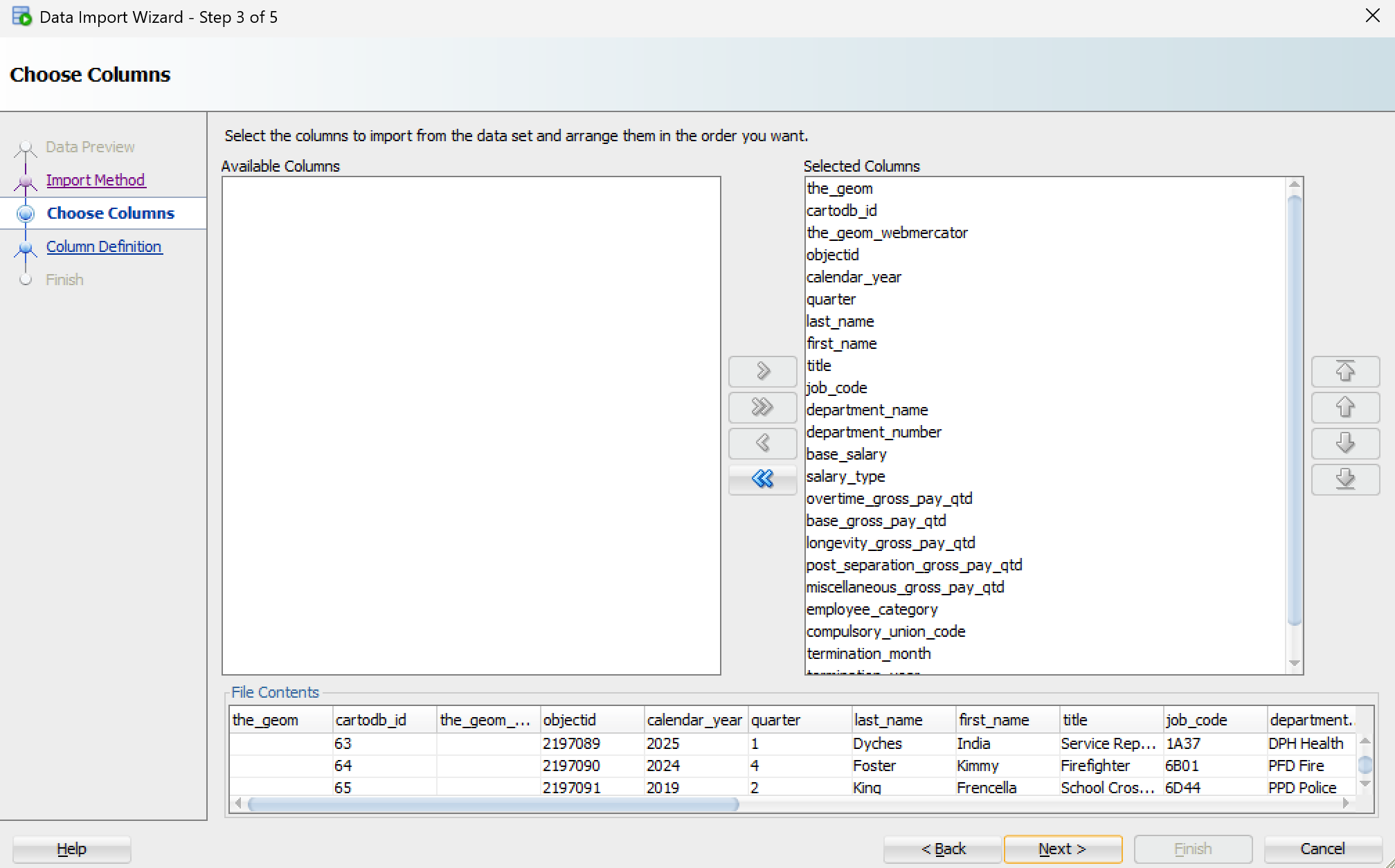
1. **Carga a una Tabla Temporal de Respaldo (Como metodo de fallback):** Primero, el CSV completo se importó a una tabla temporal (TEMP\_EMPLOYEE\_EARNINGS) con todas las columnas definidas como VARCHAR2. Esto minimizó los problemas de tipo de datos durante la carga inicial.

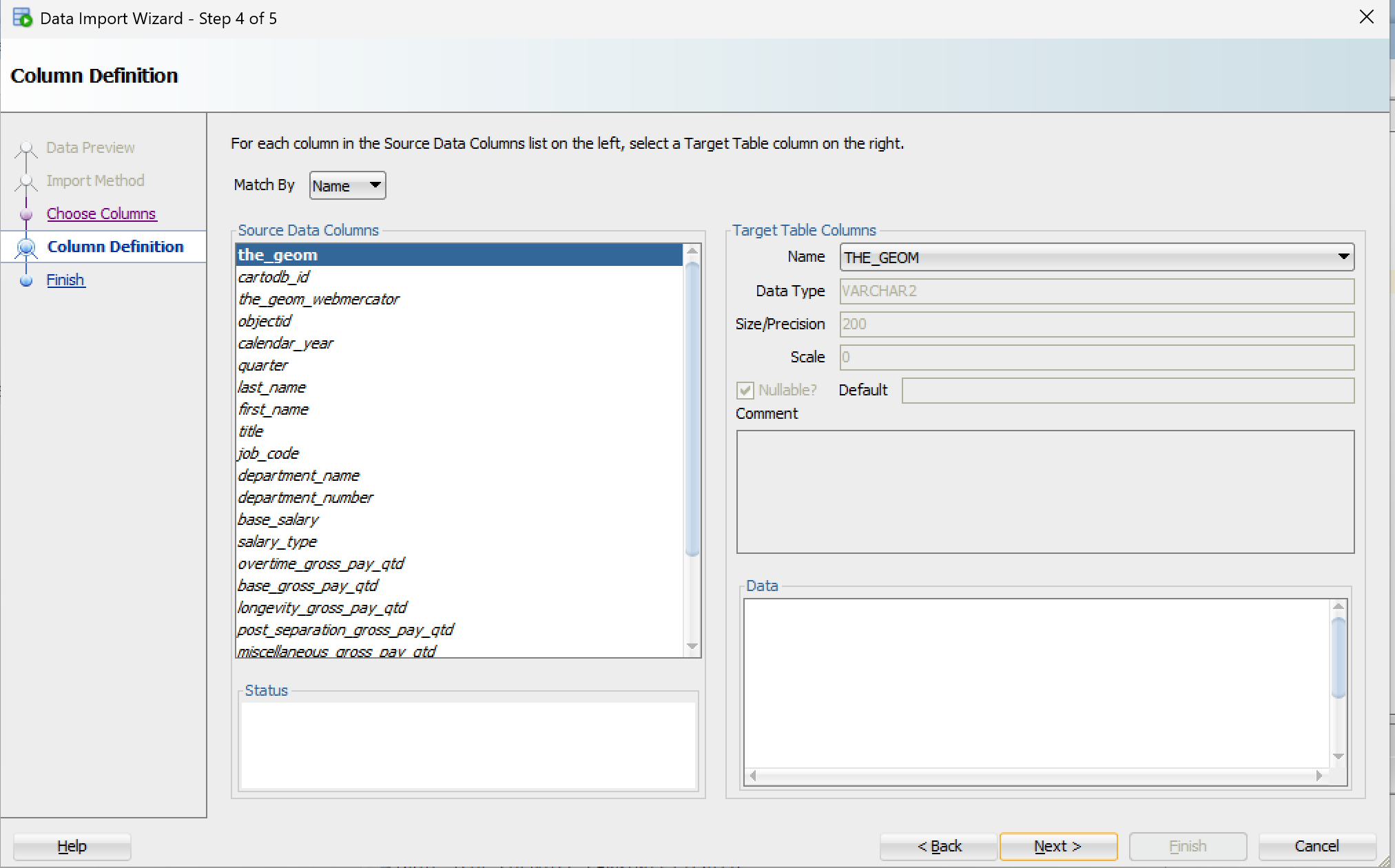


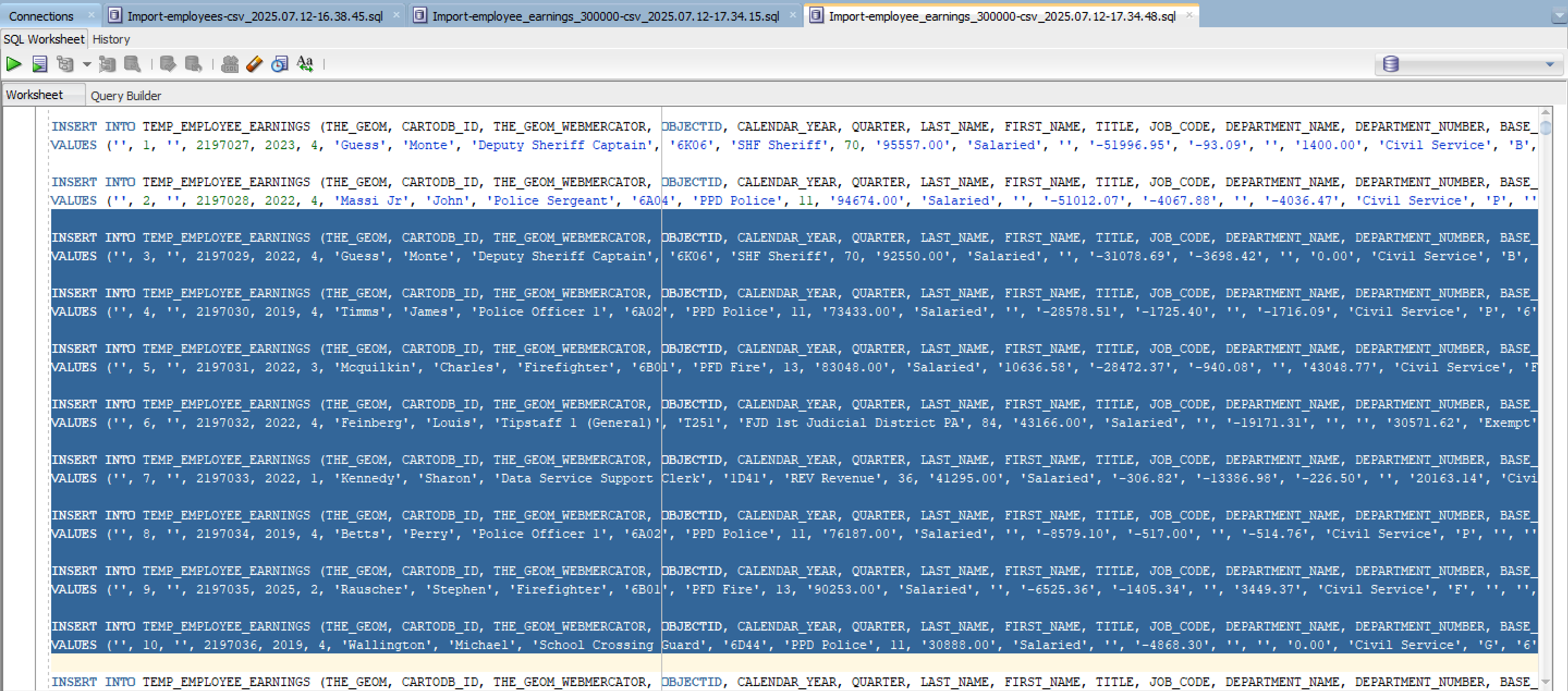
*Descripción: Script SQL para la creación de la tabla TEMP\_EMPLOYEE\_EARNINGS.*











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

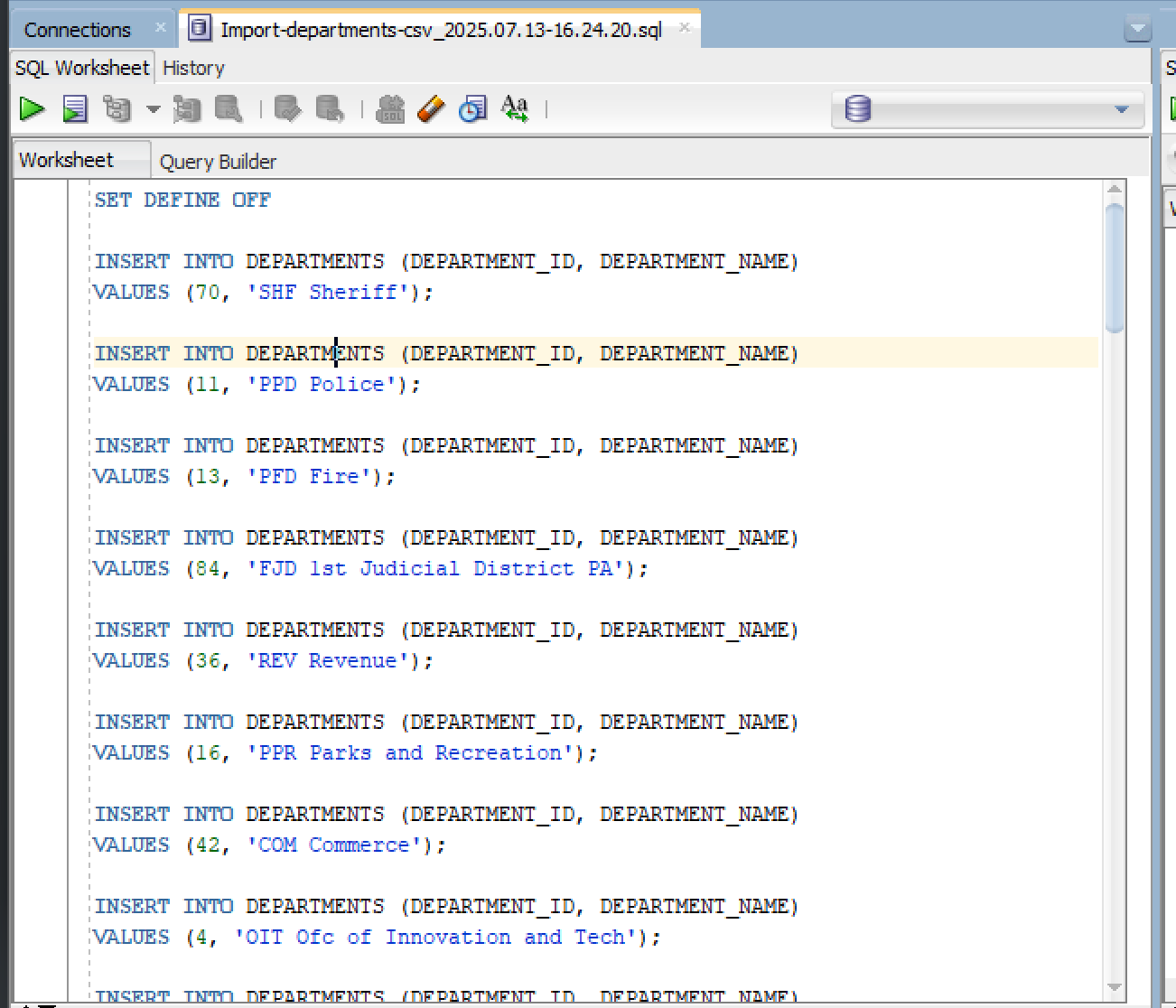
1. **Limpiar el Dataset Completo:**
   * Usamos pandas, una libreria rica en el manejo de datos en python para limpiar el csv y enlistarlo para ser importado hacia nuestra base de datos :

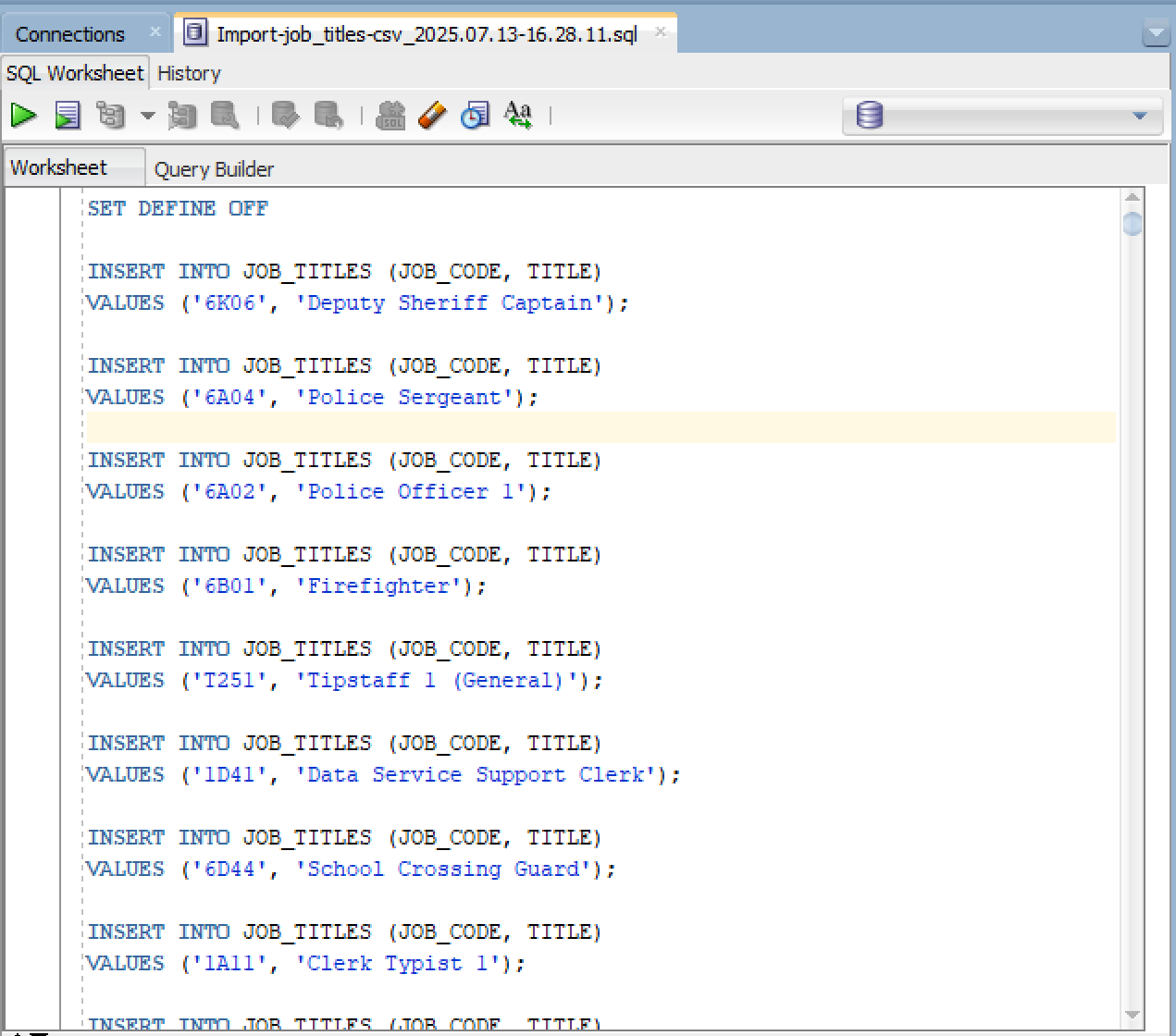
* import pandas as pd  
    
  # 1️⃣ Cargar el CSV original  
  df = pd.read\_csv('employee\_earnings\_300000.csv')  
    
  # 2️⃣ Calcular 'total\_gross\_pay' como columna derivada para earnings  
  df['total\_gross\_pay'] = (  
   df['base\_salary'].fillna(0) +  
   df['overtime\_gross\_pay\_qtd'].fillna(0) +  
   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")

*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.*

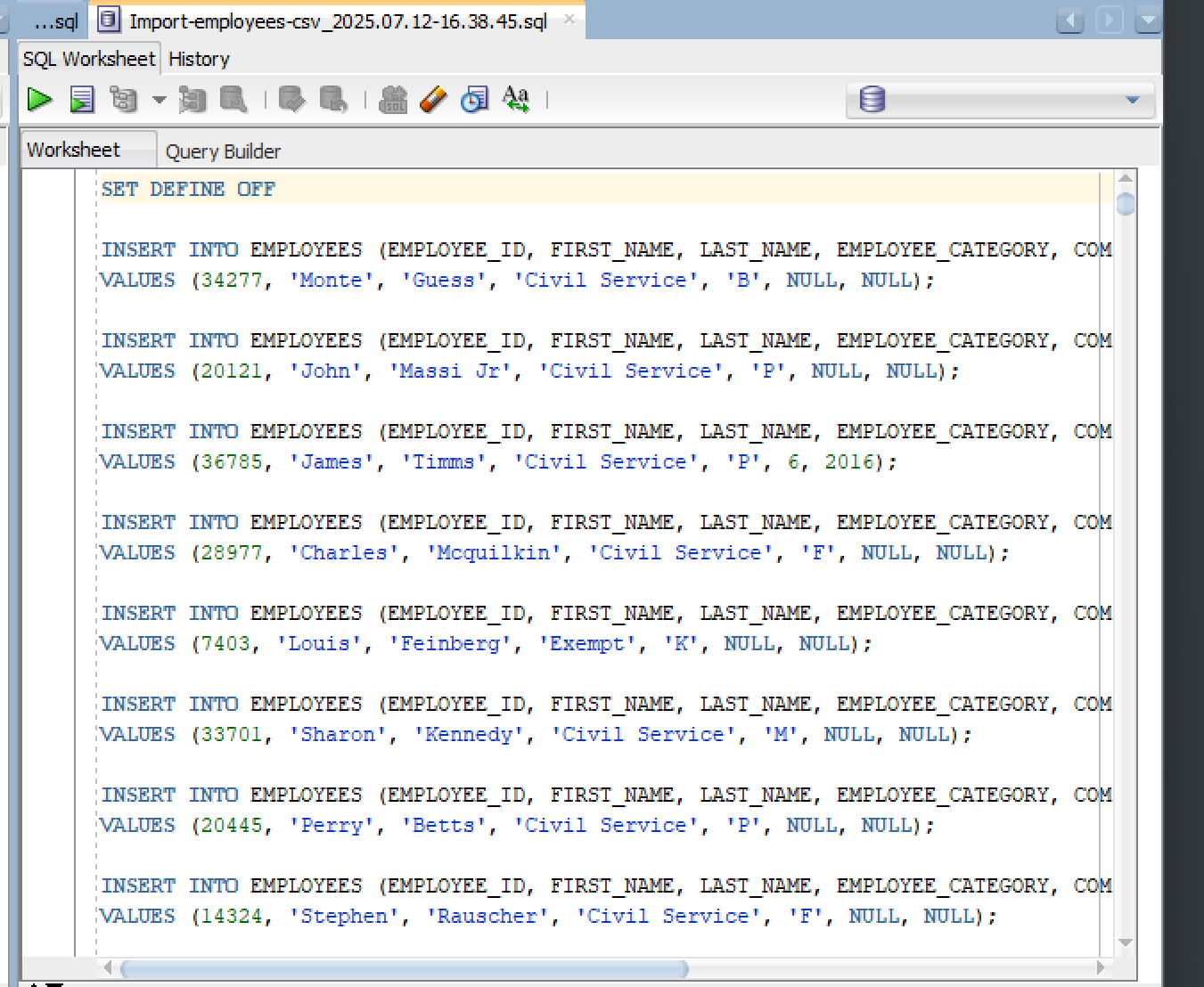
1. **Transformación y Carga a Tablas Finales:** Como metodo de fallback importariamos los datos desde la tabla temporal, utilizando sentencias INSERT INTO ... SELECT FROM para poblar las tablas DEPARTMENTS, JOB\_TITLES, EMPLOYEES y EARNINGS. Sin embargo, importamos los archivos csv limpios con nuestro 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.

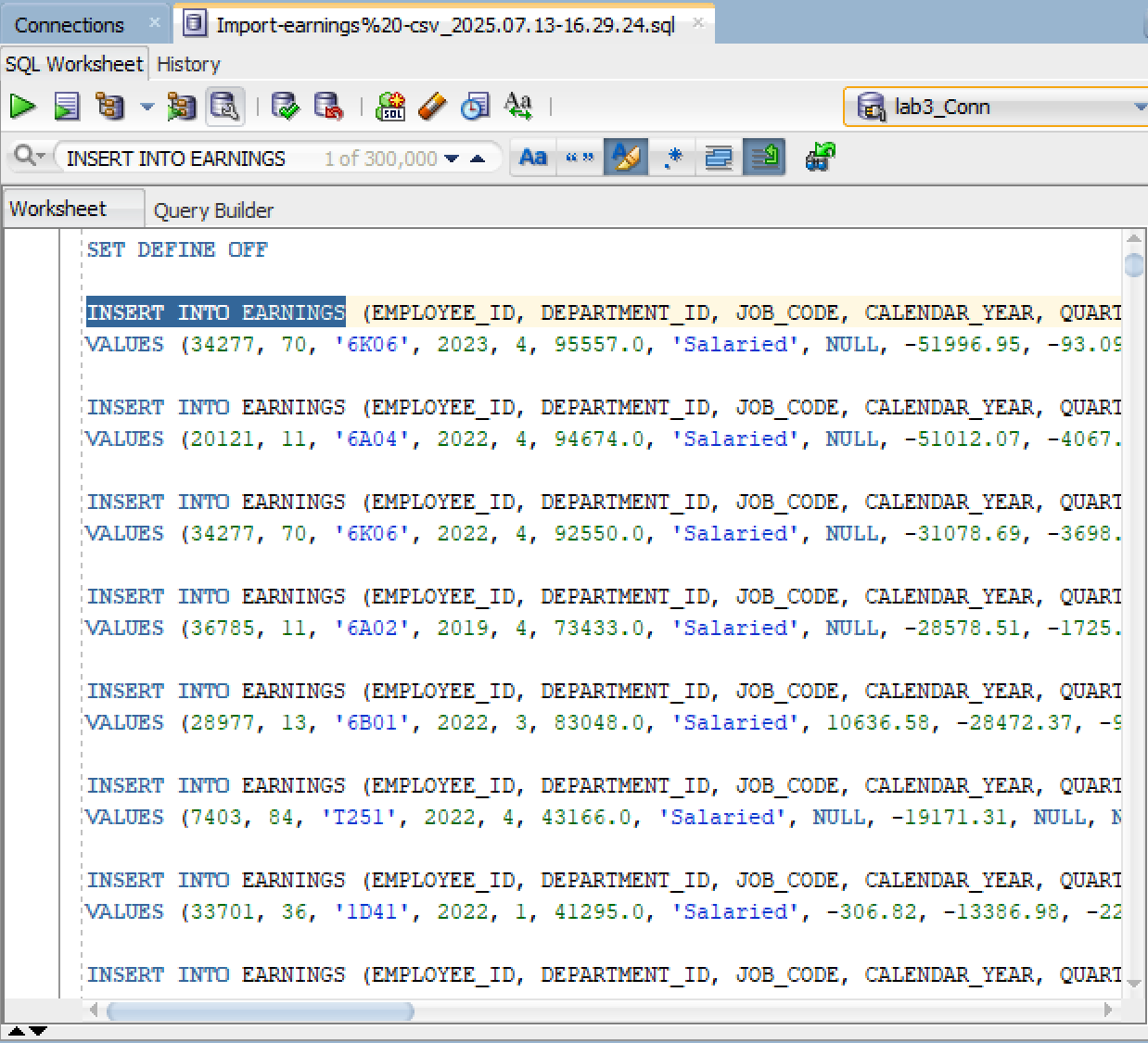
*Descripción: Visualizacion de un trigger de control para la tabla EARNINGS antes de la importación, mostrando la omision de valores nulos antes de la importacion. Evitando asi futuros conflictos*





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





*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, 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.

*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.

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

#### 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.

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

#### 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.*

### 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.

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

### 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)

*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.*

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

*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.*

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

*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.*

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

*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.*

### 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) | Tiempo con índice (s) | % Tiempo ahorrado |
| --- | --- | --- | --- |
| Salario promedio por departamento y tipo de salario en 2022 | 0.038 | 0.028 | 26.32% |
| Empleados con mayor total\_gross\_pay por departamento en 2023 Q4 | 0.087 | 0.015 | 82.76% |
| Historial de salarios de un empleado específico | 0.012 | 0.005 | 58.33% |
| Conteo de empleados por categoría y departamento con salario base superior a 80,000 | 0.149 | 0.25 | -67.79% |

## Conclusión

La implementación de la base de datos en Oracle Xe 11 G, siguiendo un proceso muy metódico que abarca desde el análisis detallado del conjunto de datos hasta la optimización del rendimiento, pone de manifiesto la gran relevancia de un diseño cuidadoso y la aplicación adecuada de técnicas avanzadas para mejorar la eficiencia. Como se observa en la tabla de tiempos de consulta, la creación de índices estratégicos impacta de manera significativa en la velocidad de respuesta de las consultas, logrando reducciones de tiempo considerables en la mayoría de los casos.

Por ejemplo, consultas como la del salario promedio por departamento y tipo de salario en 2022 y la de empleados con mayor total\_gross\_pay por departamento en 2023 Q4 experimentaron mejoras en el tiempo de ejecución del 26.32% y 82.76%, respectivamente, lo que representa un ahorro notable en recursos y tiempo. Asimismo, el historial de salarios de un empleado específico mostró una mejora del 58.33%, evidenciando que la indexación puede ser muy beneficiosa incluso en consultas con pocos registros.

Sin embargo, no todos los casos resultaron en mejoras; la consulta que realiza el conteo de empleados por categoría y departamento con salario base superior a 80,000 presentó un aumento del 67.79% en el tiempo de ejecución al aplicar el índice. Esto subraya que la creación de índices debe ser cuidadosamente analizada y aplicada según el contexto y la naturaleza de las consultas, ya que un índice mal diseñado o innecesario puede incluso degradar el rendimiento.

En resumen, la normalización de datos, la implementación de lógica de negocio mediante triggers y procedimientos, y la creación estratégica de índices conforman los pilares esenciales para desarrollar sistemas de bases de datos eficientes, robustos y fáciles de mantener. Este enfoque integral no solo garantiza la integridad y consistencia de la información, sino que también optimiza la capacidad de respuesta ante consultas, aspecto fundamental en entornos con grandes volúmenes de datos.

Este documento, junto con el de decisiones de diseño, constituye una guía valiosa para entender el proceso y las razones detrás de cada elección técnica, facilitando futuras ampliaciones o ajustes de este sistema y promoviendo buenas prácticas en el manejo y optimización de bases de datos diseñadas para Oracle Xe 11 G.

## 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.aspxhttp://srikanthtechnologies.com/blog/oracle/orahistory.aspx](http://srikanthtechnologies.com/blog/oracle/orahistory.aspx))