Designing an Employee Database Using the SQLite Software and Python

NAME: FRIDAY ODEH OVBIRORO

STUDENT NO: 22034665

Abstract

This report explores the creation and structuring of a SQL database tailored for an employee management system and discusses the following key aspects: data generation process and the data schema, the process of incorporating real-world scenarios, data cleanliness, and ethical considerations and data privacy policies as well as example queries that showcase joins and the employment of various data types in the database.

Introduction

This project focuses on designing a SQL database tailored for employee management. Utilizing Python, we generate data that mirrors real-world scenarios, addressing issues of data cleanliness, ethical considerations, and data privacy. The database is structured around four primary tables: Employee, Project, Training, and Performance. Collectively, these tables contain 1000 entries spanning 15 unique columns, accommodating diverse data formats, including nominal, ordinal, interval, and ratio. Primary and foreign keys are strategically implemented to establish meaningful relationships among the tables, facilitating comprehensive employee management.

Database Generation:

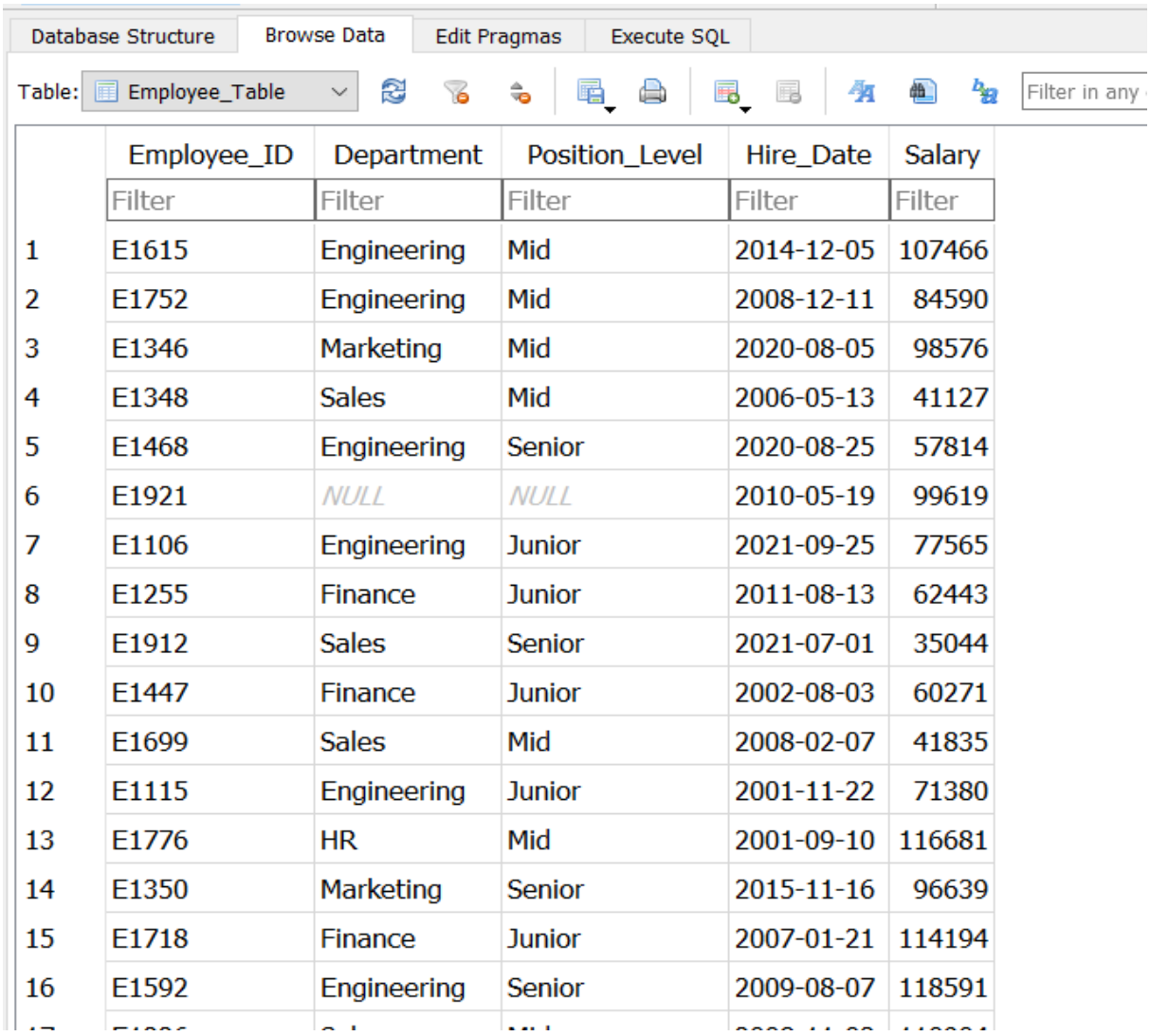
The data for the employee database was randomly generated using Python, creating four main tables: Employee, Performance, Project, and Training, each consisting of 1000 entries distributed across 15 columns. These columns in the database are meticulously designed to accommodate diverse data formats, including nominal, ordinal, interval, and ratio. To introduce real-world complexity, the database incorporates intentional missing values as well as foreign and compound keys. Below is detailed breakdown of each table:

Employee Table: The "Employee" table acts as a comprehensive repository for vital employee-related information within the database. The attributes include:

- **Employee_ID**: A distinctive identifier assigned to each employee, operating as a nominal data type—a label without conveying a specific numerical value.
- **Department**: Specifies the department to which each employee belongs, representing nominal data.
- **Position_Level:** Indicates the hierarchical position or level of each employee, utilizing ordinal data to convey a rank or order.
- **Hire_Date**: Records the date when each employee was hired, functioning as an interval data type, denoting the time span between a specific starting point and the hire date.
- **Salary**: Captures the salary of each employee as a ratio data type, facilitating quantitative analysis.

The code snippet and the image below illustrate the Python-based data generation process and the dataset for the "Employee" table in Sqlite Software.

```
# Importing the necessary Libarary

import pandas as pd
import numpy as np

# Stating the number of observations
n = 1000  # Number of rows

# Employee_ID : Nominal Data Type
employee_number = np.random.choice(range(1001, 2001), n, replace=False)
Employee_ID = [f'E{str(i).zfill(4)}' for i in employee_number]

# Department: Nominal Data Type
Department = ['Sales', 'Marketing', 'Finance', 'HR', 'Engineering']
Department = np.random.choice(Department, n, p=[0.3, 0.2, 0.15, 0.15, 0.2])

# Position_Level: Ordinal Data Type
Position_Level = ['Junior', 'Mid', 'Senior']
Position_Level = np.random.choice(Position_Level, n, p=[0.4, 0.35, 0.25])

# Hire_Date: Interval Data Type
Hire_year = np.random.randint(2000, 2023, n)
Hire_month = np.random.randint(1, 13, n)
Hire_day = np.random.randint(1, 29, n)
Hire_Date = [f'{Hire_year[i]}-{str(Hire_month[i]).zfill(2)}-{str(Hire_day[i]).zfill(2)}' for i in range(n)]

# Salary: Ratio Data Type
Salary = np.random.randint(30000, 120001, n)

### Creating a dataframe for the first table - Employees Table
Employee_Table = pd.DataFrame({
    'Employee_ID': Employee_ID,
    'Department': Department,
    'Position_Level': Position_Level,
    'Hire_Date': Hire_Date,
    'Salary': Salary
})
```



	Employee_ID	Department	Position_Level	Hire_Date	Salary
	Filter	Filter	Filter	Filter	Filter
1	E1615	Engineering	Mid	2014-12-05	107466
2	E1752	Engineering	Mid	2008-12-11	84590
3	E1346	Marketing	Mid	2020-08-05	98576
4	E1348	Sales	Mid	2006-05-13	41127
5	E1468	Engineering	Senior	2020-08-25	57814
6	E1921	*NULL*	*NULL*	2010-05-19	99619
7	E1106	Engineering	Junior	2021-09-25	77565
8	E1255	Finance	Junior	2011-08-13	62443
9	E1912	Sales	Senior	2021-07-01	35044
10	E1447	Finance	Junior	2002-08-03	60271
11	E1699	Sales	Mid	2008-02-07	41835
12	E1115	Engineering	Junior	2001-11-22	71380
13	E1776	HR	Mid	2001-09-10	116681
14	E1350	Marketing	Senior	2015-11-16	96639
15	E1718	Finance	Junior	2007-01-21	114194
16	E1592	Engineering	Senior	2009-08-07	118591

Training Table:

The "Training" table acts as a repository for employee training records, capturing essential information to facilitate effective training management. The attributes include:

- **Training_ID:** A unique identifier for each training record, serving as a nominal data type and functioning as the primary key of this table.
- **Employee_ID:** A distinctive identifier assigned to each employee undergoing training, serving as a nominal data type and linking to the employee table as a foreign key.
- **Training_Type:** Specifies the type of training, providing nominal data that categorizes training programs without implying a specific order.
- **Training_Status:** Indicates the current status of the training, serving as nominal data that categorizes the training progress.
- **Training_Duration:** Represents the duration of the training, utilizing numerical values as a ratio data type.

The code snippet and the image below demonstrates the data generation process using Python and the dataset for the "Training" table in SQLite Software.

```
# Creating the training table

# Training_ID: Nominal Data Type
training_number = np.random.choice(range(3001, 4001), n, replace=False)
Training_ID = [f'T{str(i).zfill(4)}' for i in training_number]

# Training_Type: Nominal Data Type
Training_Type = ['Technical', 'Managerial', 'HR Policies', 'Financial']
Training_Type = np.random.choice(Training_Type, n, p=[0.25, 0.25, 0.25, 0.25])

# Training_Status: Ordinal Data Type
Training_Status = ['Not Started', 'In Progress', 'Completed']
Training_Status = np.random.choice(Training_Status, n, p=[0.2, 0.3, 0.5])

# Training_Duration: Ratio Data Type (in hours)
Training_Duration = np.random.randint(8, 41, n)

### Creating a dataframe for the fourth table - Training Table
Training_Table = pd.DataFrame({
    'Training_ID': Training_ID,
    'Employee_ID': Employee_ID,
    'Training_Type': Training_Type,
    'Training_Status': Training_Status,
    'Training_Duration': Training_Duration
})
```

Table: Training_Table

Filter in any column

	Training_ID	Employee_ID	Training_Type	Training_Status	Training_Duration
	Filter	Filter	Filter	Filter	Filter
1	T3820	E1615	Financial	Completed	30
2	T3958	E1752	Financial	Not Started	19
3	T3682	E1346	Financial	Not Started	34
4	T3919	E1348	*NULL*	*NULL*	18
5	T3379	E1468	Technical	Not Started	26
6	T3757	E1921	HR Policies	Completed	17
7	T3673	E1106	Managerial	In Progress	14
8	T3954	E1255	Managerial	Completed	11
9	T3029	E1912	Technical	Not Started	12
10	T3415	E1447	Managerial	Completed	32
11	T3298	E1699	Managerial	In Progress	32
12	T3545	E1115	Managerial	Completed	11
13	T3213	E1776	Managerial	Completed	17
14	T3978	E1350	Managerial	Not Started	39
15	T3181	E1718	HR Policies	In Progress	20
16	T3686	E1592	HR Policies	Completed	18
17	T3662	E1226	*NULL*	*NULL*	36
18	T3882	E1571	Technical	In Progress	15

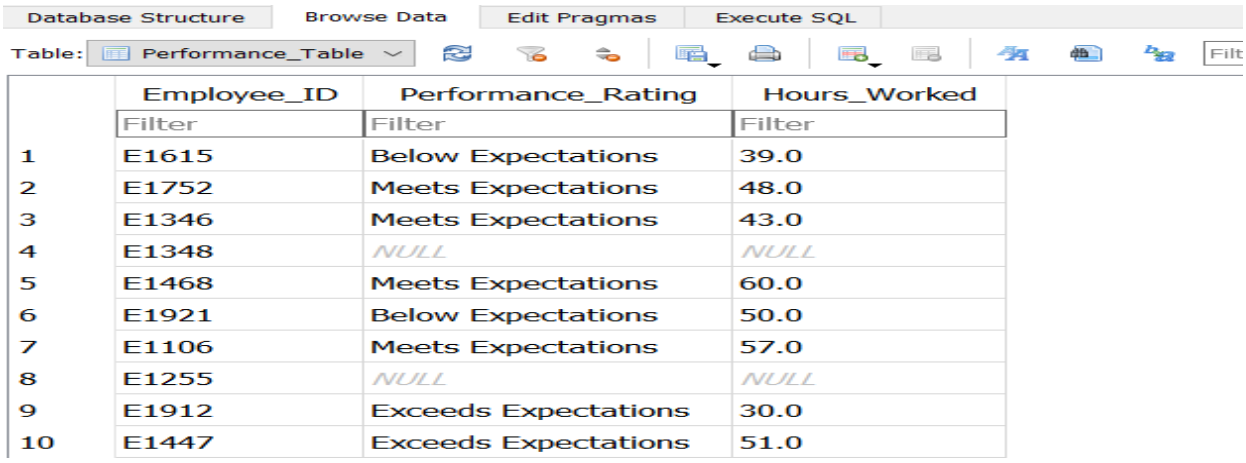
Performance Table:

The "Performance" table serves as a log for recording employee performance-related information, providing insights into work achievements and productivity. The attributes include:

- **Employee_ID:** A distinctive identifier assigned to each employee, serving as a nominal data type and acting as the primary key of this table.
- **Performance_Rating:** Represents the performance rating assigned to each employee, serving as a numerical value with a ratio data type. This attribute allows for a quantitative analysis of employee performance.
- **Hours_Worked:** Indicates the total hours worked by each employee, utilizing numerical values as a ratio data type.

The code snippet and the image below illustrate the data generation process using Python and the dataset for the "Performance" table in SQLite Software.

```
# Creating the performance Table

# Performance_Rating: Ordinal Data Type
Performance_Rating = ['Below Expectations', 'Meets Expectations', 'Exceeds Expectations']
Performance_Rating = np.random.choice(Performance_Rating, n, p=[0.2, 0.6, 0.2])

# Employee_ID : Nominal Data Type
employee_number = np.random.choice(range(1001, 2001), n, replace=False)
Employee_ID = [f'E{str(i).zfill(4)}' for i in employee_number]

# Hours_Worked: Ratio Data Type
Hours_Worked = np.random.randint(30, 61, n)

### Creating a dataframe for the third table - Performance Table
   Performance_Table = pd.DataFrame({
   'Employee_ID': Employee_ID,
   'Performance_Rating': Performance_Rating,
   'Hours_Worked': Hours_Worked
})
```



	Employee_ID	Performance_Rating	Hours_Worked
	Filter	Filter	Filter
1	E1615	Below Expectations	39.0
2	E1752	Meets Expectations	48.0
3	E1346	Meets Expectations	43.0
4	E1348	*NULL*	*NULL*
5	E1468	Meets Expectations	60.0
6	E1921	Below Expectations	50.0
7	E1106	Meets Expectations	57.0
8	E1255	*NULL*	*NULL*
9	E1912	Exceeds Expectations	30.0
10	E1447	Exceeds Expectations	51.0

Project Table:

The "Project" table serves as a comprehensive record of ongoing projects within the organization, encapsulating key details for effective project management. The attributes include:

- **Project_ID:** A unique identifier for each project, serving as a nominal data type and functioning as the primary key of this table.
- **Employee_ID:** A distinctive identifier assigned to each employee involved in a project, acting as a nominal data type and linking to the employee table as a foreign key.
- **Project_Status:** Indicates the current status of the project, providing nominal data that categorizes projects without implying a specific order.
- **Deadline_Date:** Specifies the deadline for the project, measured as an interval from a specific starting point, facilitating effective project timeline management.

The code snippet and the image below illustrate the data generation process using Python and the dataset for the "Project" table in SQLite Software.

```
# Project_ID: Nominal Data Type
project_number = np.random.choice(range(2001, 3001), n, replace=False)
Project_ID = [f'P{str(i).zfill(4)}' for i in project_number]

# Project_Status: Ordinal Data Type
Project_Status = ['Not Started', 'In Progress', 'Completed']
Project_Status = np.random.choice(Project_Status, n, p=[0.25, 0.5, 0.25])

# Employee_ID : Nominal Data Type
employee_number = np.random.choice(range(1001, 2001), n, replace=False)
Employee_ID = [f'E{str(i).zfill(4)}' for i in employee_number]

# Deadline_Date: Interval Data Type
Deadline_year = np.random.randint(2023, 2026, n)
Deadline_month = np.random.randint(1, 13, n)
Deadline_day = np.random.randint(1, 29, n)
Deadline_Date = [f'{Deadline_year[i]}-{str(Deadline_month[i]).zfill(2)}-
{str(Deadline_day[i]).zfill(2)}' for i in range(n)]

### Creating a dataframe for the second table - Projects Table
Project_Table = pd.DataFrame({
    'Project_ID': Project_ID,
    'Employee_ID': Employee_ID,
    'Project_Status': Project_Status,
    'Deadline_Date': Deadline_Date
})
```

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Project_Table

	Project_ID	Employee_ID	Project_Status	Deadline_Date
	Filter	Filter	Filter	Filter
1	P2425	E1615	Not Started	2023-08-03
2	P2246	E1752	Completed	2023-09-03
3	P2734	E1346	Not Started	2023-01-01
4	P2818	E1348	In Progress	2025-07-06
5	P2243	E1468	In Progress	2025-12-23
6	P2451	E1921	In Progress	2025-11-04
7	P2143	E1106	Not Started	2023-09-18
8	P2701	E1255	In Progress	2023-07-21
9	P2374	E1912	In Progress	2024-01-05
10	P2549	E1447	In Progress	2023-12-27
11	P2781	E1699	In Progress	2025-07-27

Integrating Missing Values, Foreign Keys, And Composite Keys To Reflect Real World Scenario

To mirror real-world scenarios accurately, I purposely introduced missing values across all tables in the database. Furthermore, I introduced both primary keys and foreign keys, with the latter serving as pointers to other tables within the database. These keys were properly designated as indices within their respective Python tables and adjusted accordingly in SQLite.

The below visual gives a representation of the code integration, showcasing the incorporation of missing values and the setup of foreign and compound keys in the tables in python which was also modified in SQLite Database

```
# Setting Primary Key for the Employee Table

Employee_Table.set_index('Employee_ID', inplace=True)
print(Employee_Table)

# Setting Primary and Compound Keys for the Project Table
Project_Table.set_index(['Project_ID', 'Employee_ID'], inplace=True)
print(Project_Table)

# Setting Primary  for the Performance Table
Performance_Table.set_index(['Employee_ID'], inplace=True)
print(Performance_Table)

# Setting Primary and Compound Keys for the Training Table
Training_Table.set_index(['Training_ID', 'Employee_ID'], inplace=True)
print(Training_Table)
```

```
# Setting Null Values into the database

# Randomly select 101 indices to set to NaN in the Employee_Table
random_indices = np.random.choice(Employee_Table.index, 101, replace=False)

# Set the corresponding values in the "Department" and "Position_Level" columns to NaN
Employee_Table.loc[random_indices, ['Department', 'Position_Level']] = np.nan

# Randomly select 98 indices to set to NaN in the Project_Table
random_indices2 = np.random.choice(Project_Table.index, 98, replace=False)

# Set the corresponding values in the "Project_Status" column to NaN
Project_Table.loc[random_indices2, 'Project_Status'] = np.nan

# Randomly select 150 indices to set to NaN in the Performance_Table
random_indices3 = np.random.choice(Performance_Table.index, 150, replace=False)

# Set the corresponding values in the "Performance_Rating" and "Hours_Worked" columns to NaN
Performance_Table.loc[random_indices3, ['Performance_Rating', 'Hours_Worked']] = np.nan

# Randomly select 52 indices to set to NaN in the Training_Table
random_indices4 = np.random.choice(Training_Table.index, 52, replace=False)

# Set the corresponding values in the "Training_Type" and "Training_Status" columns to NaN
Training_Table.loc[random_indices4, ['Training_Type', 'Training_Status']] = np.nan
```

Database Schema:

The multi-table database consists of four key tables: Employee, Project, Training, and Performance. Each table employs a strategic use of primary, foreign, and compound keys to establish meaningful relationships between them. To maintain data integrity, specific columns in each table are subject to carefully implemented constraints such as unique and not null constraints.

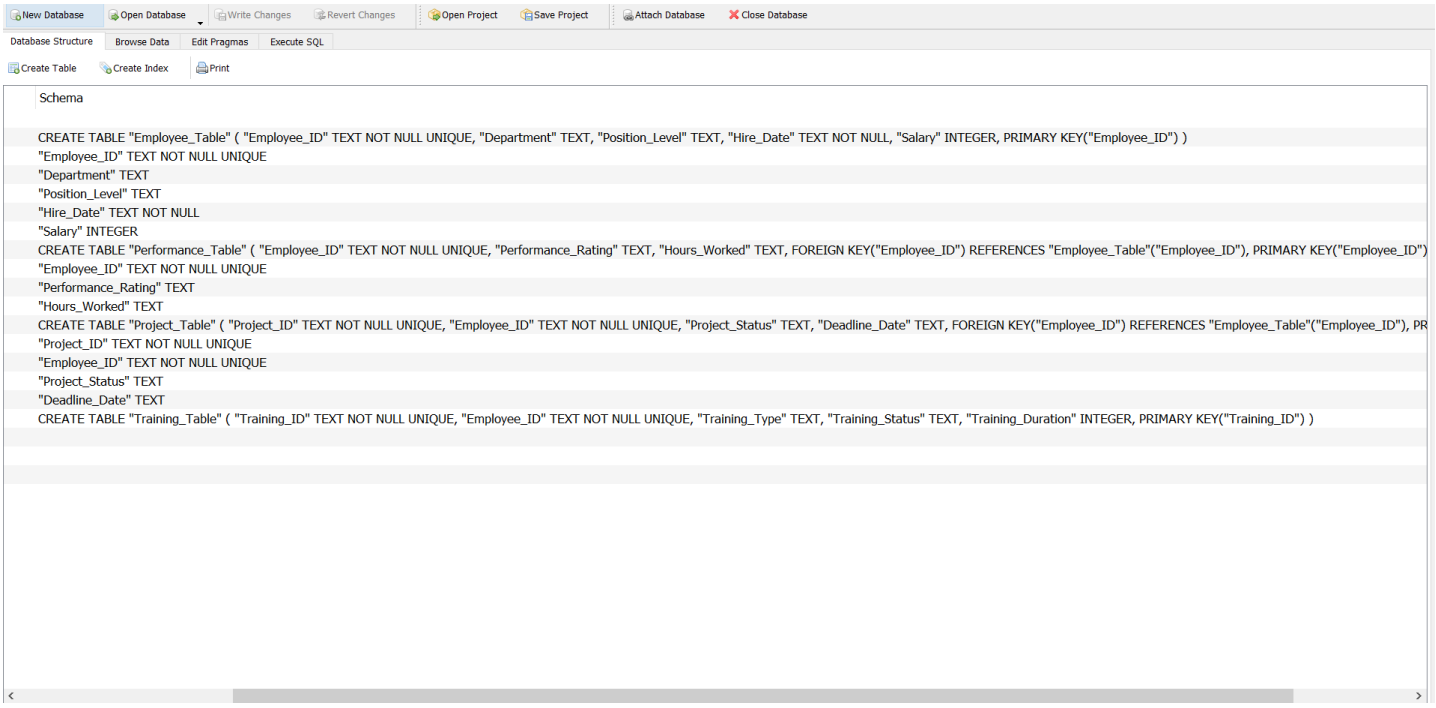
Employee Table: The primary key in the Employee Table is the Employee_ID, serving as a pivotal reference for other tables in the database. I've reinforced this key with unique and not null constraints, ensuring accurate and complete data. The hire date was also constraint on not null

Project Table: In the Project Table, I've established a compound key using the Project_ID. This key serves as the primary identifier for the Project Table, while the Employee_ID column acts as a foreign key referencing the Employee Table. To fortify data integrity, unique and not null constraints are applied to the Project_ID.

Training Table: The Training Table features a compound key combining Training_ID and Employee_ID. Employee_ID acts as a foreign key referencing the Employee Table, while Training_ID assumes the role of the primary key for the Training Table. I've applied unique and not null constraints to both keys, safeguarding the accuracy and completeness of the information.

Performance Table: The Performance Table utilizes the Employee_ID as the primary key, ensuring a unique identifier for each employee's performance data. The Performance Table includes columns for Performance_Rating and Hours_Worked. To maintain data integrity, unique and not null constraints are applied to the Employee_ID.

The screenshot below visually represents the database schema, showcasing foreign and compound keys, along with the constraints applied to each table.



Report Justification and Ethical Discussion

The integration of the "Employee Table," "Project Table," "Training Table," and "Performance Table" in this employee database creates a comprehensive dataset covering employee details, project assignments, training records, and performance metrics. This strategic database design facilitates effective data analysis and reporting within the organizational context. The justification for each table is delineated below:

Employee Table:

- **Justification:** The "Employee Table" acts as the foundational repository for employee details, including Employee_ID, Department, Position Level, Hire Date, and Salary. This table is essential for identifying and managing individual employees across the organization. It serves as a cornerstone for personalized communication, workforce management, and tracking membership-related benefits.

Project Table:

- Justification: The "Project Table" serves as a centralized database for project-related information, capturing Project_ID, Employee_ID, Project Status, and Deadline Date. This table is vital for tracking employee involvement in various projects, project statuses, and associated deadlines. It provides insights into workforce allocation and project management.

Training Table:

- Justification: The "Training_Table" records essential information about employee training, including Training_ID, Employee_ID, Training_Type, Training_Status, and Training_Duration. This table plays a crucial role in monitoring and analyzing employee training initiatives, ensuring compliance, and enhancing skill development. It contributes to informed decisions regarding workforce training and development.

Performance Table:

- Justification: The "Performance Table" captures key metrics related to employee performance, including Employee_ID, Performance_Rating, and Hours_Worked. This table is instrumental in evaluating employee performance over time, facilitating data-driven decisions for performance management and resource allocation.

Ethics and Data Privacy Considerations:

In the development of the four data tables, namely the "Employee_Table," "Project_Table," "Training_Table," and "Performance_Table," a paramount focus has been placed on upholding rigorous ethical standards and data privacy principles, aligning with real-world practices. This commitment is in accordance with data protection regulations such as General Data Protection Regulation (GDPR), ensuring the responsible handling of individual information. Below, we explore the ethical considerations and privacy safeguards integrated into the design and generation of these tables.

Data Anonymization and Privacy Preservation: A foundational principle guiding the data generation process for this workforce database was the deliberate exclusion of sensitive personal identifiers. The data collected in these tables underwent meticulous anonymization, ensuring that elements such as employee names and contact information were omitted. This approach minimizes the risk of privacy breaches and safeguards employee confidentiality.

Use of Pseudonyms and Random Identifiers: Following data privacy best practices, employee identifiers, such as 'Employee_ID' were intentionally designed as pseudonyms or random identifiers. These identifiers are used exclusively for identification and tracking within the dataset and are not directly linked to real individuals. This approach ensures that employee identities remain undisclosed, reinforcing data privacy and security.

It's imperative to note that the intentional omission of certain elements, such as names and contact information, aligns with ethical considerations and data protection principles, prioritizing employee privacy and confidentiality.

Query Example: Illustrating Joins and Data Types:

In the concluding section of this report, I present a snapshot query exemplifying the utilization of joins and showcasing various data types within the employee database. The query addresses the following question:

Question: What is the Employee ID, Department, and Position Level, Hire Dates of employees with completed project statuses?

SQL 1

```
- What are the Booking IDs, Event IDs, Customer Names, Ticket Prices, and Ratings of the top 10 bookings with the highest ticket prices?-

SELECT  B.Booking_ID, F.Event_ID, B.Customer_Name, B.Ticket_Price,F.Rating
FROM Bookings_Table AS B
INNER JOIN Feedback_Table AS F
ON B.Event_ID = F.Event_ID
ORDER BY B.Ticket_Price DESC
LIMIT 10
```

	Booking_ID	Event_ID	Customer_Name	Ticket_Price	Rating
1	B1573	E0357	Henry Oni	200	2
2	B1665	E0635	Kemi Ojo	200	1
3	B1585	E0904	Kemi Fabayo	200	2
4	B1340	E0464	*NULL*	200	4
5	B1813	E0613	Jack Johnson	200	3
6	B1027	E0487	*NULL*	200	2
7	B1145	E0387	*NULL*	199	2
8	B1743	E0757	Jack Oni	199	3
9	B1600	E0073	Seyi Fabayo	199	1
10	B1092	E0778	Jack Daniel	199	2

```
Execution finished without errors.
Result: 10 rows returned in 34ms
At line 3:
SELECT  B.Booking_ID, F.Event_ID, B.Customer_Name, B.Ticket_Price,F.Rating
FROM Bookings_Table AS B
INNER JOIN Feedback_Table AS F
ON B.Event_ID = F.Event_ID
ORDER BY B.Ticket_Price DESC
LIMIT 10
```

This query establishes connections between the Employee Table and the Project Table. It not only provides insights into data relationships through joins but also highlights the diverse data types present within the database as seen below

Conclusion:

This report delves into the development of an employee database, highlighting ethical data practices, diverse data types, and thoughtful table design and database schema. The four core tables—Employee_Table, Project_Table, Training_Table, and Performance_Table—form a comprehensive dataset for streamlined data analysis. The integration of missing values, primary and foreign keys, and ethical considerations demonstrates a dedication to responsible data management. The presented query exemplifies the effective use of joins and the utilization of diverse data types, providing insight into the analytical capabilities of the employee database.