Name: Nikhil Prema chandra rao
Student ID: 02105149

Title: CSV Query Performance Benchmark - Documentation

---

1. Schema Overview

The salary tracker database stores information about employees' personal details, job roles, salaries, departments, and schools. The main focus of this schema is to track salaries and job positions for employees at different schools.

2. Table List

- **Employees**: Stores personal information about each employee.
- **Departments**: Represents various departments within a school.
- **Schools**: Stores details about the schools.
- **JobTitles**: Lists all possible job titles for employees.
- **SalaryDetails**: Stores salary and employment information for employees.

3. Database Tables & Structure

1. Employees Table

This table stores information about the employees.

Column Name	Data Type	Description
EmployeeID	INTEGER	Primary Key, Unique identifier for each employee.
FirstName	TEXT	First name of the employee.
LastName	TEXT	Last name of the employee.
BirthDate	DATE	Birthdate of the employee.
Email	TEXT	Email address of the employee.
PhoneNumber	TEXT	Phone number of the employee.

2. Departments Table

This table stores the details of each department.

Column Name	Data Type	Description
DepartmentID	INTEGER	Primary Key, Unique identifier for each department.
DepartmentName	TEXT	Name of the department (e.g., "Engineering", "HR").
SchoolID	INTEGER	Foreign Key, references Schools(SchoolID).

3. Schools Table

This table stores information about the schools.

Column Name	Data Type	Description
SchoolID	INTEGER	Primary Key, Unique identifier for each school.
SchoolName	TEXT	Name of the school.
SchoolCampus	TEXT	Campus location of the school.
Address	TEXT	Address of the school.

4. JobTitles Table

This table stores the job titles that employees may hold.

Column Name	Data Type	Description
JobTitleID	INTEGER	Primary Key, Unique identifier for each job title.
JobTitle	TEXT	Name of the job title (e.g., "Lecturer", "Manager").

5. SalaryDetails Table

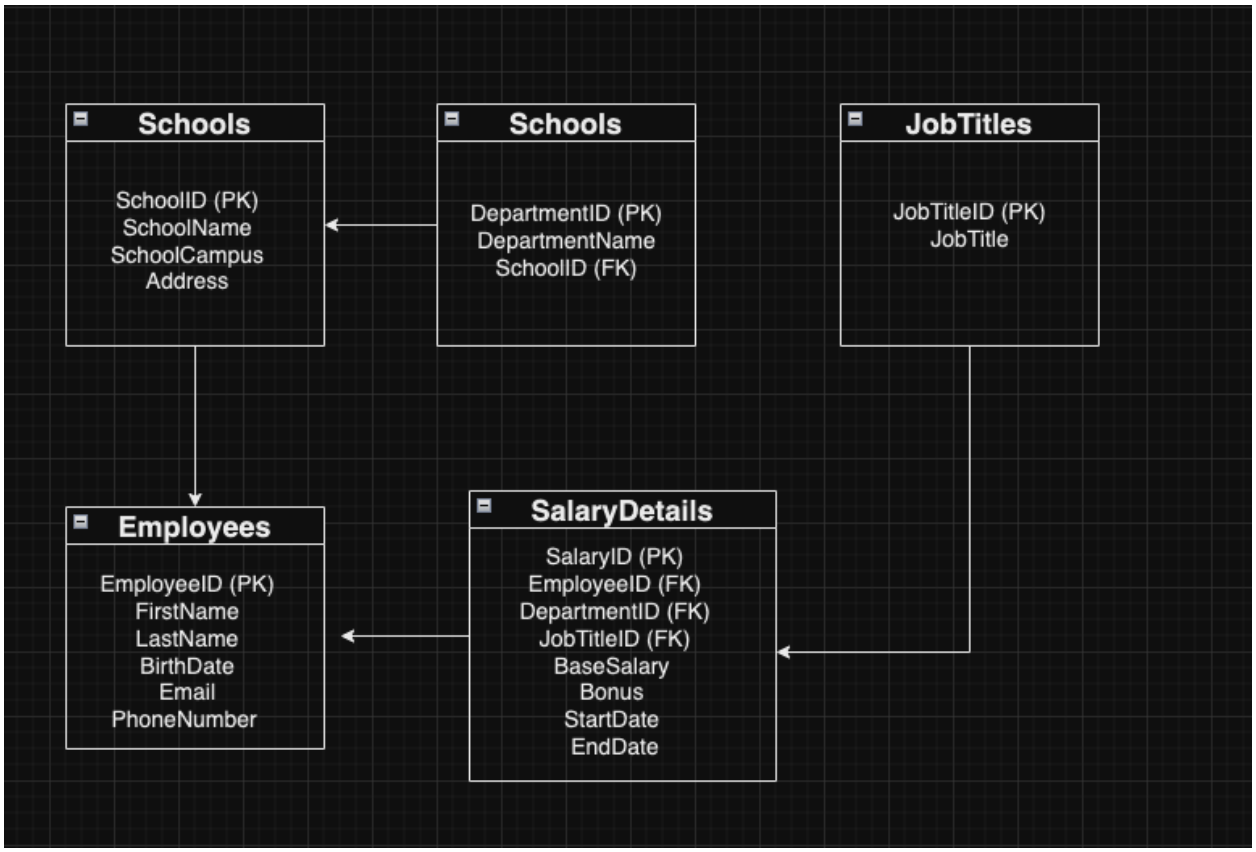
This table stores salary-related information for each employee.

Column Name	Data Type	Description
SalaryID	INTEGER	Primary Key, Unique identifier for each salary record.
EmployeeID	INTEGER	Foreign Key, references Employees(EmployeeID).
DepartmentID	INTEGER	Foreign Key, references Departments(DepartmentID).
JobTitleID	INTEGER	Foreign Key, references JobTitles(JobTitleID).
BaseSalary	INTEGER	The employee's base salary.
Bonus	INTEGER	Bonus amount for the employee.
StartDate	DATE	Employment start date.
EndDate	DATE	Employment end date (nullable).

4. Relationships Between Tables

- **Employees** ↔ **SalaryDetails**: One-to-many relationship, as one employee can have multiple salary records.
- **Departments** ↔ **SalaryDetails**: One-to-many relationship, as one department can have multiple employees with salary records.
- **JobTitles** ↔ **SalaryDetails**: One-to-many relationship, as one job title can be associated with multiple salary records.
- **Schools** ↔ **Departments**: One-to-many relationship, as one school can have multiple departments.

5. Example Schema Diagram



6. Data Integrity Constraints

- **Primary Keys:** Each table has a primary key (PK) to ensure uniqueness.
- **Foreign Keys:** Foreign keys (FK) enforce the relationship between tables and ensure referential integrity. For example:
    - **`EmployeeID`** in the **`SalaryDetails`** table reference**s** **`Employees(EmployeeID).`**
    - **`DepartmentID`** in the **`SalaryDetails`** table references **`Departments(DepartmentID).`**
- **Data Types:** Each column's data type is carefully chosen to match the expected data (e.g., **`INTEGER`** for IDs and salaries, **`TEXT`** for names and addresses)**.**
- **Not Null Constraints:** Important fields like **`EmployeeID`**, **`DepartmentID`,** and **`SalaryID`** should not be null.
- **Unique Constraints:** Fields like **`Email`** in the **`Employees`** table should have a unique constraint to ensure that no two employees have the same email address.

7. Documentation for Future Reference

The schema described above should be documented thoroughly, both in the form of comments in the database (using tools like SQL comments) and in written documentation for end-users and developers. It is important to describe the following:

1. **Tables and Columns**: Detailed explanation of each table and column.
2. **Relationships**: Explanation of the relationships between tables, including referential integrity.
3. **Constraints**: Descriptions of the primary keys, foreign keys, and any unique or not-null constraints.
4. **Indexing**: Any indexes that need to be created to speed up queries (e.g., indexes on **`EmployeeID`** or **`DepartmentID`**).

8. SQL Code for Creating the Schema

Here's an example of how you could define the schema in SQL:

```
CREATE TABLE Schools (
    SchoolID INTEGER PRIMARY KEY,
    SchoolName TEXT NOT NULL,
    SchoolCampus TEXT,
    Address TEXT
);

CREATE TABLE Departments (
    DepartmentID INTEGER PRIMARY KEY,
    DepartmentName TEXT NOT NULL,
    SchoolID INTEGER,
    FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID)
);

CREATE TABLE JobTitles (
    JobTitleID INTEGER PRIMARY KEY,
    JobTitle TEXT NOT NULL
);

CREATE TABLE Employees (
    EmployeeID INTEGER PRIMARY KEY,
    FirstName TEXT NOT NULL,
    LastName TEXT NOT NULL,
    BirthDate DATE NOT NULL,
    Email TEXT UNIQUE NOT NULL,
```

```
    PhoneNumber TEXT
);

CREATE TABLE SalaryDetails (
    SalaryID INTEGER PRIMARY KEY,
    EmployeeID INTEGER,
    DepartmentID INTEGER,
    JobTitleID INTEGER,
    BaseSalary INTEGER,
    Bonus INTEGER,
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),
    FOREIGN KEY (JobTitleID) REFERENCES JobTitles(JobTitleID)
);
```

9. Explanation of the Normalization process

Normalization Process Explanation:

Normalization is a database design technique that involves organizing the attributes (columns) and tables of a database to reduce redundancy and dependency. The goal is to eliminate anomalies that can occur during data insertion, update, and deletion. This process ensures that the database is efficient, consistent, and scalable.

Steps of Normalization:

1. **First Normal Form (1NF):**
   Ensures that each column contains atomic (indivisible) values.
   - Removes repeating groups, meaning that each field contains a single value, and the order of the data does not matter.
2. **Second Normal Form (2NF):**
   - Achieved when the database is in 1NF and all non-key attributes are fully dependent on the primary key.
   - This step eliminates partial dependencies, i.e., no attribute should be dependent on only part of a composite key.
3. **Third Normal Form (3NF):**
   - Achieved when the database is in 2NF, and all attributes are not only fully dependent on the primary key but also on the key only.
   - This removes transitive dependencies, meaning no non-key attribute should depend on another non-key attribute.

4. **Boyce-Codd Normal Form (BCNF):**
   A stricter version of 3NF where every determinant is a candidate key. It addresses situations where 3NF doesn't handle certain types of dependency adequately.

Normalization helps to minimize data redundancy and ensures the database is more maintainable, ensuring that each piece of information is stored only once.

Schema Design Decisions:

When designing the schema for this project, several decisions were made to ensure that the database was both functional and efficient:

1. **Separation of Concerns:**
   - The schema uses different tables to logically separate distinct entities, such as individuals, job titles, departments, and schools. This reduces data redundancy and allows for more efficient data management.
   - For instance, a table for individuals (PersonID, PersonName) and a separate table for departments (DepartmentName) and schools (SchoolName) helps minimize repetitive data storage.
2. **Foreign Keys:**
   - Foreign keys were used to establish relationships between different tables. For example, the *PersonID* in the salary tracker table is a foreign key that links to the person's details in a separate person table.
   - This ensures referential integrity, meaning that data remains consistent across related tables.
3. **Data Integrity and Constraints:**
   - Constraints like *NOT NULL*, *UNIQUE*, and *CHECK* were used to enforce data integrity, ensuring that the data entered into the database meets specific requirements.
   - For example, ensuring that earnings are always positive, or that dates follow the correct format, helps maintain the quality of the data.
4. **Normalization Decisions:**
   The dataset was normalized to 3NF to remove redundancy and ensure that each attribute only depends on the primary key.
   - For instance, storing employment status, earnings, and birthdate in one table avoids repeating these fields for each entry in the dataset, while still allowing queries to link relevant information efficiently.
5. **Indexing:**
   Indexes were likely used on key columns (such as *PersonID*, *SchoolName*, and *DepartmentName*) to speed up queries involving joins or search operations.
   - This improves the performance of queries that rely on these columns for filtering, grouping, or ordering results.
6. **Query Optimization:**

- The schema design was influenced by the need for optimized query performance. For example, normalized tables minimize data duplication, making operations like `JOIN` more efficient.
- This schema supports efficient queries such as retrieving the maximum earnings per person or counting the number of active faculty in a particular department.

In summary, the normalization process and careful schema design decisions were made to ensure the database is efficient, maintainable, and consistent. These decisions also helped optimize the performance of SQL queries by reducing redundancy, ensuring proper relationships between entities, and enforcing data integrity.

10. Conclusion

This design offers a robust and normalized structure for the Salary Tracker database, ensuring that data is stored efficiently and relationships between entities are logically defined. By adhering to best practices in database normalization, the design minimizes redundancy, promotes data integrity, and simplifies future maintenance or scalability efforts.

The use of well-defined relationships and constraints ensures that the database enforces rules for data consistency, such as referential integrity, while still allowing the flexibility needed to adapt to evolving business requirements. Key entities, such as employees, positions, salaries, and adjustments, are clearly delineated, making the system intuitive for both developers and analysts.

Comprehensive documentation accompanying the database will further enhance its usability by providing detailed explanations of each table, field, and relationship. This ensures that new developers, database administrators, and other stakeholders can quickly understand and work with the system, reducing the learning curve and potential for misinterpretation.

Overall, this design serves as a foundation for a reliable and efficient Salary Tracker system, capable of handling current needs while being adaptable for future growth and feature expansion.