Understanding and Addressing Employee Attrition in the USA

Nazmus Saquib
nsaquib2
CSE 460
nsaquib2@buffalo.edu

Sakib Hassan
sakibhas
CSE 460
sakibhas@buffalo.com

Tawhidur Rahman
tawhidur
CSE 460
tawhidur@buffalo.edu

Overview of Phase 1

Problem Statement

Employee attrition in the USA has been on the rise over the past decade, posing significant challenges for organizations. The lack of understanding regarding the underlying factors contributing to attrition makes it difficult for businesses to implement effective retention strategies. Our project aims to analyze the reasons behind employee attrition and provide insights to help organizations mitigate this issue.

Background

The problem of employee attrition is significant as it impacts organizational productivity, morale, and ultimately, profitability. High turnover rates lead to increased recruitment and training costs, disrupt team dynamics, and can harm the company's reputation.

Contribution

By developing a database to analyze employee attrition, we aim to provide actionable insights for organizations to better understand and address this issue. Our contribution lies in offering a systematic approach to identify key factors driving attrition, thereby enabling organizations to implement targeted retention strategies.

Target Users:

1. Database Users

Human resource managers, organizational analysts, and business executives will use the database to gain insights into employee attrition trends, identify risk factors, and develop retention strategies.

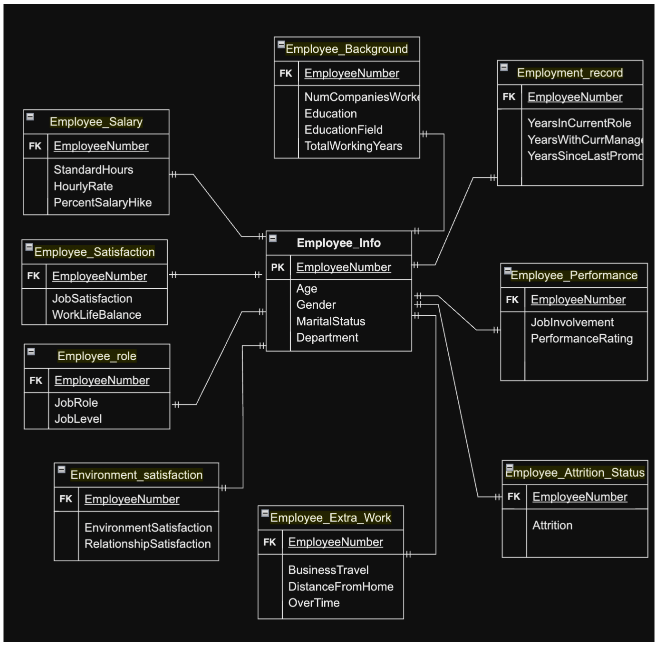
2. Database Administrators

The database administrators will be responsible for maintaining the database, ensuring data integrity, and optimizing query performance. They will also oversee data updates and security measures.

3. Real-Life Scenario

For instance, consider a multinational corporation with a high turnover rate in its call center operations. The HR department aims to understand the reasons behind this attrition and develop interventions to improve employee retention. Our database will serve as a valuable tool for HR managers to analyze employee data and make informed decisions.

E/R Diagram



The E/R diagram depicts the relationships between different tables in our database.

Here the employeenumber is a unique value for all the tables. We used it as a primary key for the employee_info table and as a foreign key for all other tables.

List of Relations and Attributes:

Employee_Info
Attributes: EmployeeNumber (Primary Key), Age, Gender, MaritalStatus, Department

Employee_Background:
Attributes: EmployeeNumber (Foreign Key), NumCompaniesWorked, Education, EducationField, TotalWorkingYears

Employee_Performace:
Attributes: EmployeeNumber (Foreign Key), JobInvolvement, PerformanceRating

Employee_Attrition_Status:
Attributes: EmployeeNumber (Foreign Key), Attrition

Employee_Extra_Work:
Attributes: EmployeeNumber (Foreign Key), BusninessTravel, DistanceFromHome, OverTime

Employee_Satisfaction:
Attributes: EmployeeNumber (Foreign Key), JobSatisfaction, WorkLifeBalance,

Employee_Salary:
Attributes: EmployeeNumber (Foreign Key), StandardHours, HourlyRate, PercentSalaryHike

Employee_role:
Attributes: EmployeeNumber (Foreign Key), JobRole, JobLevel

Employee_record:
Attributes: EmployeeNumber (Foreign Key), YearsInCurrentRole, YearsWithCurrManager, YearsSinceLastPromotion

Environment_satisfaction:
Attributes: EmployeeNumber, (Foreign Key) EnvironmentSatisfaction, RelationshipSatisfaction

<u>Schema:</u>

Employee_Info(EmployeeNumber (Primary Key) varchar, Age int, Gender varchar, MaritalStatus varchar, Department varchar)

Employee_Background(EmployeeNumber (Foreign Key) varchar, NumCompaniesWorked int, Education varchar, EducationField varchar, TotalWorkingYears int)

Employee_Performace(EmployeeNumber (Foreign Key) varchar, JobInvolvement varchar, PerformanceRating int)

Employee_Attrition_Status(EmployeeNumber (Foreign Key) varchar, Attrition varchar)

Employee_Extra_Work(EmployeeNumber (Foreign Key) varchar, BusninessTravel int, DistanceFromHome int, OverTime int)

Employee_Satisfaction(EmployeeNumber (Foreign Key) varchar, JobSatisfaction int, WorkLifeBalance int)

Employee_Salary(EmployeeNumber (Foreign Key) varchar, StandardHours int, HourlyRate int, PercentSalaryHike int)

Employee_role(EmployeeNumber (Foreign Key) varchar, JobRole varchar, JobLevel int)

Employee_record(EmployeeNumber (Foreign Key) varchar, YearsInCurrentRole int, YearsWithCurrManager int, YearsSinceLastPromotion int)

Environment_satisfaction(EmployeeNumber (Foreign Key) varchar, EnvironmentSatisfaction int, RelationshipSatisfaction int)

BCNF Transformation: We ensured that all relations are in Boyce-Codd Normal Form (BCNF) by checking any functional dependencies or redundancies. This involved decomposing tables or modifying their structure to eliminate anomalies and maintain data integrity.

Data Acquisition and Preparation

We acquired a large dataset containing relevant information about employee attributes, such as demographics, job satisfaction, performance ratings, and work-life balance. This dataset is obtained from real-world sources such as HR databases or simulated using data generation programs. We ensured that the dataset aligns with our database schema and contains enough diversity to support various types of SQL queries.

Data Acquisition: Obtained a large dataset containing information on employee attributes such as age, gender, education level, job role, performance ratings, etc. This dataset reflected real-world scenarios and was sufficiently diverse to capture different factors influencing employee attrition.

Data Preparation: Ensured that the acquired dataset was in a suitable format for loading into our database. This involved writing scripts to transform the data into a compatible form, cleaning the data to remove any inconsistencies or errors, and structuring the data to fit our database schema.

Data Loading: Loaded the dataset into our database, ensuring that each table corresponds to a specific entity or attribute of interest. Verified that the data is accurately represented in the database and that relationships between tables are properly established.

Data Validation: Performed validation checks to ensure the integrity and quality of the data. This involved running queries to verify data consistency, identifying any anomalies or discrepancies, and resolving any issues that arise.

Creating Links: Ensured that the dataset contained enough interconnections between rows of different tables to support a variety of advanced SQL queries.

Ensuring Boyce-Codd Normal Form (BCNF) Compliance

To ensure that our database design adheres to BCNF, we analyzed the functional dependencies within each relation and decomposed tables as necessary to eliminate any redundancy or dependency violations.

Dependency Analysis: We identified the functional dependencies within each relation in our database schema.

BCNF Decomposition: No relation is found to violate BCNF, so we did not have to decompose the relation into smaller relations to eliminate redundancy and dependency violations.

In our initial schema, there is no multi-valued attribute, no partial dependency, and no transitive dependency, where EmployeeNumber is the superkey. So, all the relations are in BCNF.

Why the database is in BCNF:

Primary Key Uniqueness: In BCNF, each table must have a primary key, and all non-key attributes must be non-transitively dependent (i.e., directly dependent) on the primary key. In our diagram, EmployeeNumber is indicated as the primary key (PK) for all tables. This suggests that each table's attributes are functionally dependent on EmployeeNumber, which is essential for BCNF compliance.

Functional Dependencies: The relationships (arrows) between tables primarily reflect foreign key (FK) constraints where EmployeeNumber in one table links to EmployeeNumber in another, signifying that all attribute dependencies on EmployeeNumber are direct and not through some other attribute. This direct dependency aligns with BCNF's requirement that every determinant in a table is a candidate key.

Normalization Check:
Employee_Info: Attributes like Age, Gender, MaritalStatus, and Department directly depend on EmployeeNumber. There are no attributes that functionally determine EmployeeNumber, nor are there transitive dependencies.

Employee_Performance and others: Similarly, in tables like Employee_Performance, Employee_Salary, Employee_Satisfaction, and more, each non-key attribute is functionally dependent only on EmployeeNumber. This eliminates the possibility of transitive dependencies where a non-key attribute determines another non-key attribute.

No Redundancy or Update Anomalies: Each piece of information is stored only once. For instance, employee age is stored only in the Employee_Info table and not repeated elsewhere, which prevents update anomalies and maintains data integrity.

The ER diagram, therefore, exhibits a well-structured relational schema in which all non-key attributes in every table are directly dependent on the primary keys, with no transitive dependencies. This is the core requirement for a database to be in BCNF. Such a design helps in maintaining data consistency and integrity across the database, which is crucial for effective data management and retrieval.

Phase 2

1. Handling Larger Dataset and Indexing Concepts

Challenges Encountered

One challenge we faced when we dealt with our large dataset was slower query performance due to the volume of data being processed. Additionally, loading large datasets into the database took considerable time and required optimization to ensure efficient data ingestion.

Solution through indexing

To address performance issues, we adopted indexing concepts to optimize query execution. Indexes were created on columns frequently used in WHERE clauses or JOIN conditions to speed up data retrieval.

We identified key columns in our dataset that were frequently used in queries, such as EmployeeNumber and Age. Indexes were created on these columns to improve query performance. An example of our indexing query is below.

Query Query History

```
Update employee_background set education = '2' where employeenumber = '34';
```

Data Output Messages Notifications

```
UPDATE 0

Query returned successfully in 163 msec.
```

Here we can see that the query took 163 mili sec. Then we applied indexing on the EmployeeNumber column of the employee_background table.

Query Query History

```
CREATE INDEX idx_emp_bg
ON employee_background (Employeenumber);
```

Data Output Messages Notifications

```
CREATE INDEX

Query returned successfully in 286 msec.
```

Below is the result after indexing.

Query Query History

```
Update employee_background set education = '2' where employeenumber = '34';

CREATE INDEX idx_emp_bg
ON employee_background (Employeenumber);
```

Data Output Messages Notifications

```
UPDATE 0

Query returned successfully in 44 msec.
```

We can see that the same query that took 163 mili sec before now takes only 44 mili sec after indexing. It means the query was optimized by 70%.

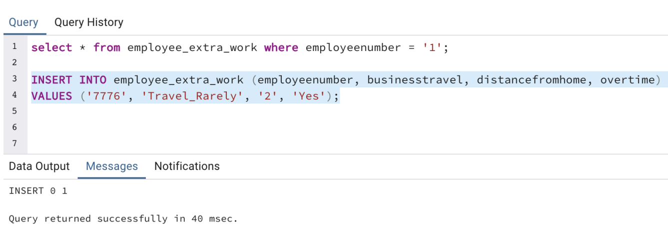
Regular monitoring of query execution times helped us evaluate the effectiveness of indexing strategies. Adjustments were made as needed to further optimize performance. By adopting these indexing concepts, we were able to enhance the efficiency of our database queries and mitigate the challenges posed by handling larger datasets.

2. Testing database with 10 queries

Inserting Data (2 queries):

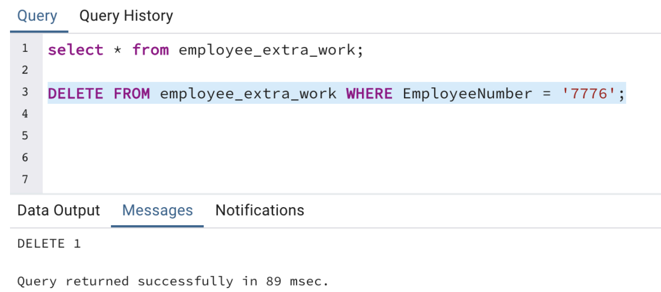
2.1. Inserted a new employee record into the employee_info table.

2.2. Inserted a record into the employee_extra_work table for a specific employee.

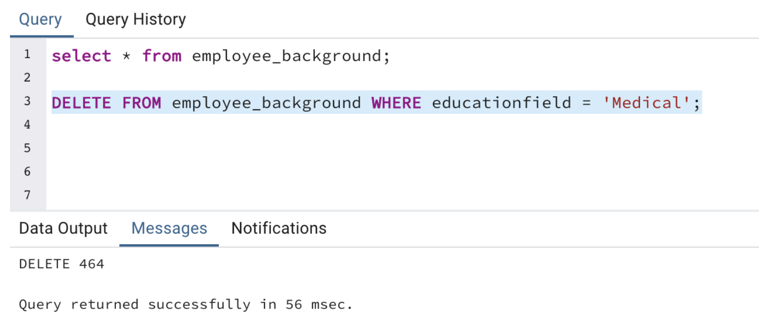


<u>Delete (2 queries):</u>

2.3. Deleted an employee record from the employee_extra_work table.

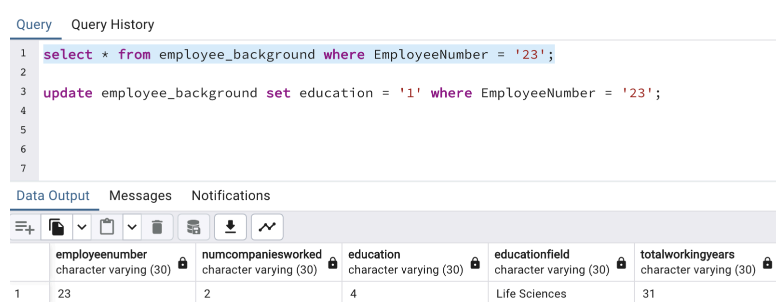


2.4. Deleted employee records whose medical field are medical from the employee_background table.



<u>Update (2 queries):</u>

2.5. Updated education to “1” for a specific employee in the employee_background table.



Query Query History

```sql
select * from employee_background where EmployeeNumber = '23';

update employee_background set education = '1' where EmployeeNumbe
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 44 msec.
```

2.6. Updated employee_role to “IT executive” for a specific employee in the employee_role table.

Query Query History

```sql
select * from employee_role where EmployeeNumber = '445';

update employee_role set jobrole = 'IT Executive' where EmployeeNumber = '445
```

Data Output Messages Notifications

	employeenumber character varying (30)	jobrole character varying (30)	joblevel character varying (30)
1	445	Sales Executive	2

Query Query History

```sql
select * from employee_role where EmployeeNumber = '445';

update employee_role set jobrole = 'IT executive' where EmployeeNumber
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 83 msec.
```

Select (4 queries):

2.7. Joined Employee_info and Employee_Performance table on EmployeeNumber and selected age, department, performancerating, and employeenumber for all the employees.

Query Query History

```sql
SELECT e.EmployeeNumber, e.Age, e.Department, ep.PerformanceRating
FROM Employee_info e
JOIN Employee_Performance ep ON e.EmployeeNumber = ep.EmployeeNumber;
```

Data Output Messages Notifications

	employeenumber character varying (30)	age character varying (30)	department character varying (30)	performancerating character varying (30)
1	1	41	Sales	3
2	2	49	Research & Development	4
3	4	37	Research & Development	3
4	5	33	Research & Development	3
5	7	27	Research & Development	3
6	8	32	Research & Development	3

2.8. Calculated the average age for each department (group by) in the Employee_info table, converting the Age column to a numeric type before averaging to handle potential string representations of age values.

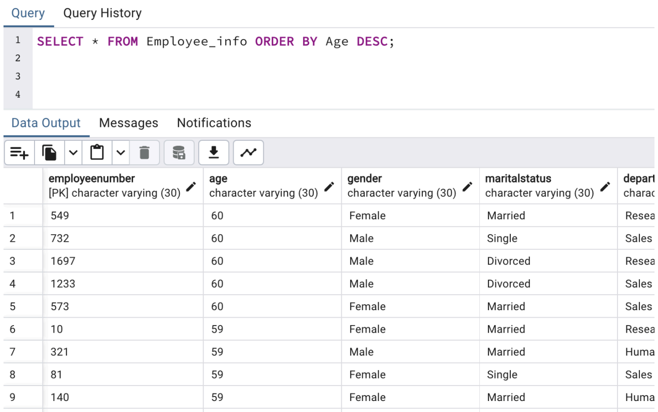
Query Query History

```sql
SELECT Department, AVG(Age::numeric) AS AverageAge
FROM Employee_info
GROUP BY Department;
```

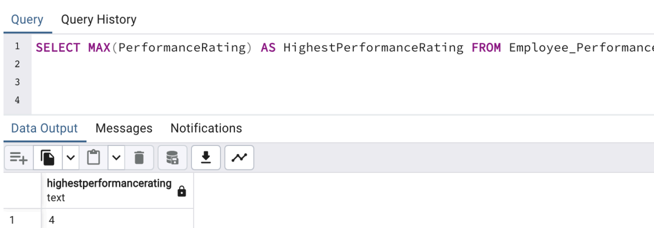
Data Output Messages Notifications

	department character varying (30)	averageage numeric
1	Marketing	27.0000000000000000
2	Human Resources	37.8095238095238095
3	Research & Development	37.0426638917793965
4	Sales	36.5426008968609865

2.9. Retrieved all columns (*) from the Employee_info table and ordered the results in descending (DESC) order based on the Age column.



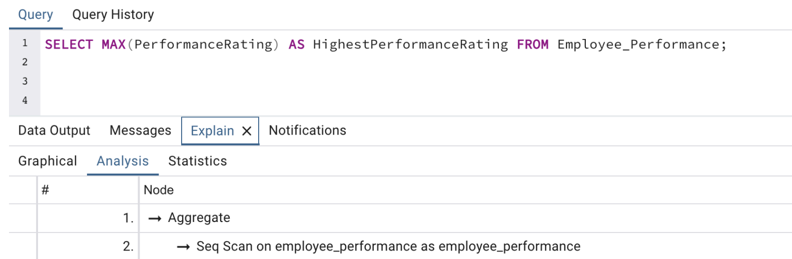
2.10. Selected the maximum value of the PerformanceRating column from the Employee_Performance table and renamed the result as HighestPerformanceRating. It calculates the highest performance rating recorded in the table.



3. Query execution analysis

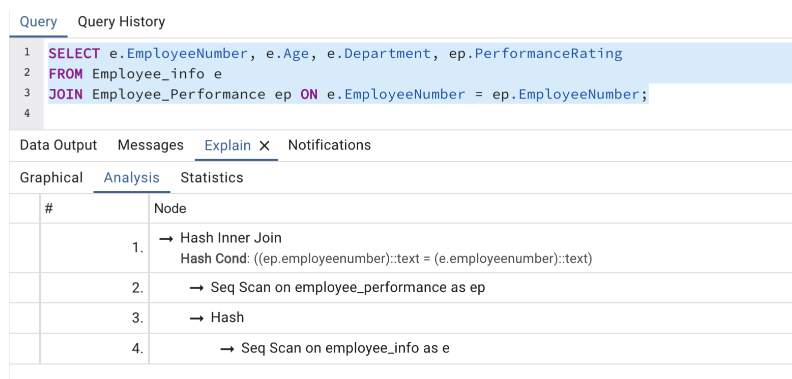
We analyzed the explanation generated by PostgreSQL for each query to identify potential bottlenecks and areas for optimization.

Problematic Query 1:



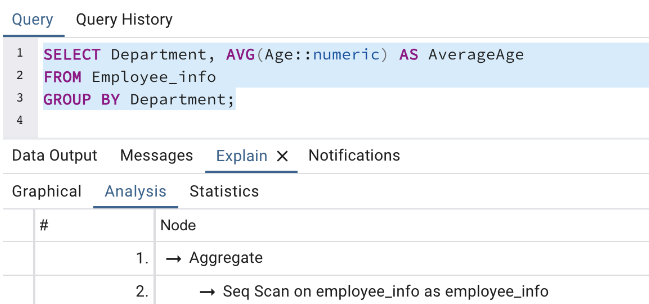
Using the explain tool we found out that one way to optimize this query is to create an index on the PerformanceRating column in the Employee_Performance table. This can improve the performance of the query by allowing PostgreSQL to quickly find the maximum value without scanning the entire table.

Problematic query 2:



One way to optimize the query is to use Integer Keys and ensure that EmployeeNumber in both tables is of the same integer type for faster join performance. Moreover, we can create indexes on EmployeeNumber in both Employee_info and Employee_Performance tables to speed up the join operation. Thirdly, we can avoid converting EmployeeNumber to text for the join condition. We need to ensure that both columns are of the same data type. Lastly, we can also consider using a WHERE clause to filter the data before joining to reduce the number of rows processed.

Problematic Query 3:



With the help of “explain” tool and referring to the book, we came to know that we can optimize this query by ensuring that the ‘Age’ column is stored as a numerical data type (INTEGER, FLOAT, etc.) instead of character varying to avoid the need for casting. Furthermore, we could create an index on the Department column to speed up the grouping process.

Conclusion

By analyzing the execution plans and proposing optimizations, we can address potential performance issues in the identified queries and improve overall database efficiency.

Contributions

33% Each member of the team.