SenioCare

YADVENDER SINGH, University at Buffalo, United States of America
SHAWNA SAHA, University at Buffalo, United States of America
DIPTANGSHU DE, University at Buffalo, United States of America

ACM Reference Format:
Yadvender Singh, Shawna Saha, and Diptangshu De. 2023. SenioCare. 1, 1 (August 2023), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PROBLEM STATEMENT

Every day, we have a range of daily tasks to accomplish, such as banking, ordering takeout, paying bills, booking cabs, and so on. Because of digitalization, these services are now at the tip of our fingers and just a click away from the comfort of our own homes. Senior citizens benefit the most from digitalization, yet frequently their lack of technological know-how hinders them from enjoying such services from home. With the aid of SenioCare, older adults may call in and obtain these services.

Intension : Through digitalization, routine tasks like banking, ordering takeout, paying bills, booking cabs, etc. can all be done with the touch of a finger. Senior citizens gain most from digitalization, however, they sometimes are unable to access these services at home due to a lack of technological expertise. Providing them with access to such services is the motivation behind this project.

Solution : Older individuals can dial-in SenioCare's helpline number to use these services.

2 TARGET USER

The User: Senior citizens and employees of SenioCare.
The Administer: Employees of SenioCare who have admin rights.
Real-life scenario description: When a senior citizen dials the hotline number, an employee responds and files a request for the sort of assistance needed. Citizen A, for example, dials the hotline number to order a cab. If this is a first-time call, employee E answers it and inputs citizen A's information. Employee E then prepares a request that includes the type of service as well as any additional requirements.

Authors' addresses: Yadvender Singh, yadvende@buffalo.edu, University at Buffalo, , Buffalo, New York, United States of America, ; Shawna Saha, shawnasa@buffalo.edu, University at Buffalo, , Buffalo, New York, United States of America, ; Diptangshu De, diptangs@buffalo.edu, University at Buffalo, , Buffalo, New York, United States of America,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
Manuscript submitted to ACM

3 DATABASE IMPLEMENTATION

3.1 Data Schema:

Person Table

Attribute Name	Type	Reference	Description
PersonID	Integer	Primary Key	ID of the Person Record On Deletion: Foreign Key Constraint added to maintain Referential Integrity
FirstName	Varchar, Not Null		First Name of the Person
LastName	Varchar		Last Name of the Person
DateOfBirth	Date		Date of Birth of Person
Email	Varchar		Email of the Person
PhoneNumber	Varchar, Not Null		Phone Number of the Person
SSN	Varchar		SSN of the Person
Address	Varchar, Not Null		Address of the Person

Fig. 1. Person Table

Client Table

Attribute Name	Type	Reference	Description
ClientID	Integer	Primary Key	ID of Client Record On Deletion: Foreign Key Constraint added to maintain Referential Integrity
PersonID	Integer, Not Null	Foreign Key referencing Person Table	Foreign Key to a Person Record with Constraint.

Fig. 2. Client Table

Employee Table

Attribute Name	Type	Reference	Null/Not Null
EmpID	Integer	Primary Key	ID of Employee Record On Deletion: Foreign Key Constraint added to maintain Referential Integrity
PersonID	Integer, Not Null	Foreign Key referencing Person Table	Foreign Key to a Person Record with Constraint.
SessionActive	Boolean		Current Login Activity Indicator
RequestAssigned	Boolean		Current Request Assigned Indicator
Role	Varchar, Not Null		Role of Employee like Admin, Volunteer etc.

Fig. 3. Employee Table

EmployeePay Table

Attribute Name	Type	Reference	Description
PayID	Integer	Primary Key	ID of Employee Pay record
EmpID	Integer, Not Null	Foreign Key referencing Employee Table	Foreign Key to Employee Record with constraint
TotalWorkingHours	Integer		Total Hours worked by Employee
HourlyPay	Integer		Hourly wage
TotalPay	Integer		Total Payment calculated for the Employee

Fig. 4. EmployeePay Table

After Milestone 1, Sub-category field was eliminated from the Request Table since the field was redundant and did not provide additional value.

Request Table

Attribute Name	Type	Reference	Description
ReqID	Integer	Primary Key	ID of Request Record
ClientID	Integer, Not Null	Foreign Key referencing Client Table	Foreign Key to Client Record with Constraint
EmpID	Integer, Not Null	Foreign Key referencing Employee Table	Foreign Key to Employee Record with Constraint
StartTime	DateTime		Start time of request
EndTime	DateTime		Completion time of request
Category	Integer	Foreign Key referencing Category Table	Foreign Key to Category Table
Comments	Varchar		Special Request/comments of client
Status	Varchar		Current Status of Request like InProgress, Completed

Fig. 5. Request Table

Category Table

Attribute Name	Type	Reference	Description
CateoryID	Integer	Primary Key	ID of Category Record On Deletion: Foreign Key Constraint added to maintain Referential Integrity
Type	Varchar, Not Null		Name of the Category Record

Fig. 6. Category Table

4 BOYCE-CODD NORMAL FORM (BCNF) ANALYSIS

4.0.1 ***Person Table:*** Functional Dependency is of form:
PersonID -> FirstName, LastName,DateOfBirth, Email, PhoneNumber, SSN, Address.
Since PersonID is a Super Key, this table is in BCNF Form.

4.0.2 ***Client Table:*** Since the relation has only two attributes namely – ClientID and PersonID, this table is in BCNF.

4.0.3 ***Employee Table:*** Functional Dependency is of form:
EmployeeID -> PersonID, SessionActive, RequestAssigned, Role.
PersonID -> EmployeeID, SessionActive, RequestAssigned, Role.
Since PersonID and EmployeeID are Super Keys, this table is in BCNF Form.

4.0.4 ***EmployeePay Table:*** Functional Dependency is of form:
PayID -> EmpID, TotalWorkingHours, HourlyPay, TotalPay.
Since PayID is a Super Key, this table is in BCNF Form.

4.0.5 ***Request Table:*** Functional Dependency is of form:
ReqID -> ClientID, EmpID, StartTime, EndTime, CategoryID, Comments, Status.
Since ReqID is a Super Key, this table is in BCNF Form.

4.0.6 ***Category Table:*** Since the relation has only two attributes namely – CategoryID and Type, this table is in BCNF.

5 DATA SOURCE

5.0.1 ***Person Table:*** FirstName, LastName and Address were taken from GitHub resource link - https://github.com/dwillis/other-people/blob/master/names.csv
For rest of the columns data was generated using script.

5.0.2 ***Client Table:*** Data was generated using script.

5.0.3 ***Employee Table:*** Data was generated using script.

5.0.4 ***EmployeePay Table:*** Data was generated using script.

5.0.5 ***Request Table:*** Data was generated using script.

5.0.6 ***Category Table:*** Data was generated using script.

6 ENTITY RELATIONSHIP DIAGRAM

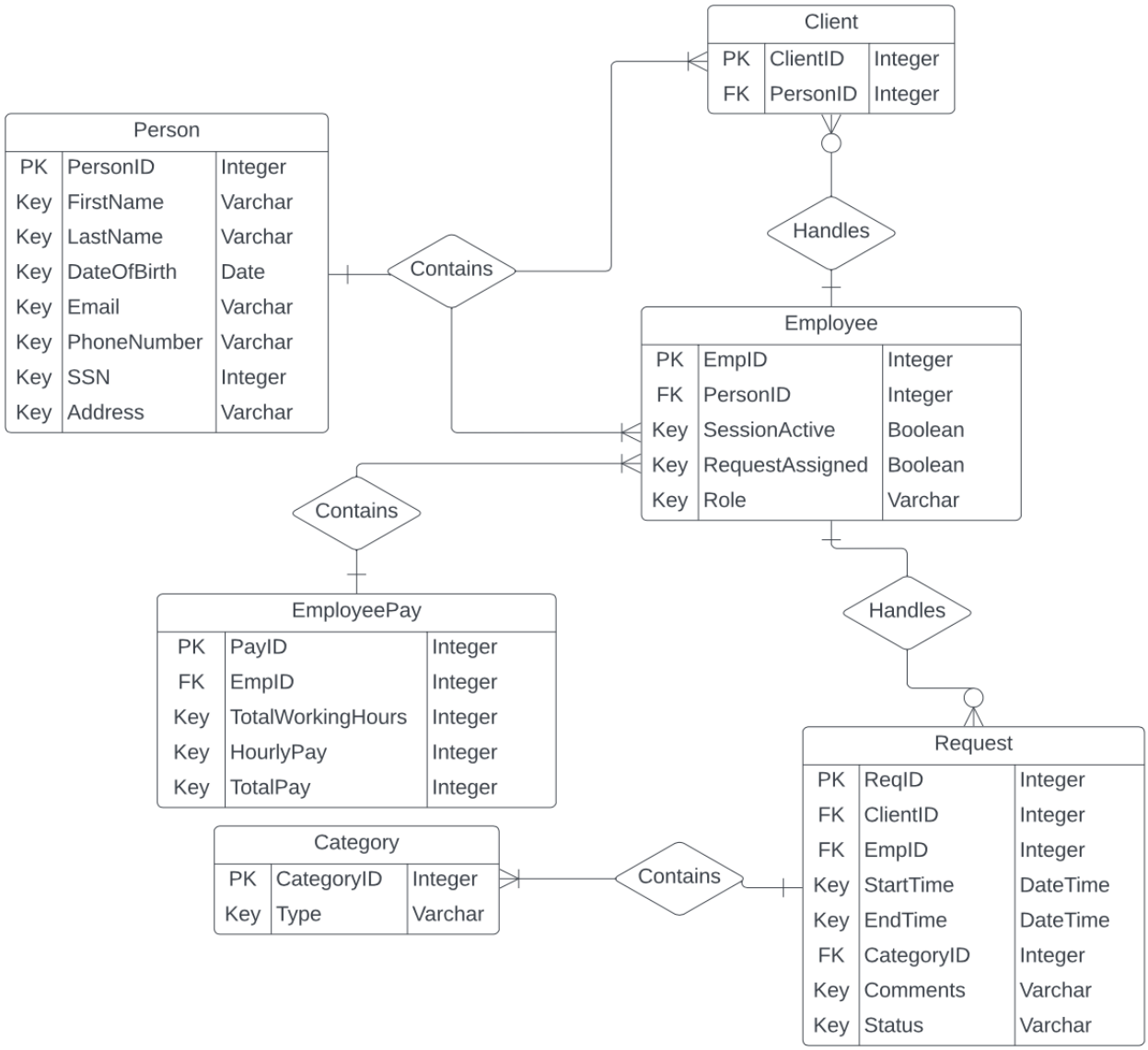

Fig. 7. Entity Relationship Diagram

7 LIST OF QUERIES

7.1 Insert a record

```
INSERT INTO
"Person"("PersonId","FirstName", "LastName", "Address", "DateofBirth", "Email", "PhoneNumber", "SSN")
VALUES ('7870','John', 'Doe', '331, Main St, Buffalo, NY-1421', '11-30-1980', 'johndoe@email.com', '2343454567', '342-67-6789');
```

7.2 Update an existing record

```
UPDATE "Employee" SET "PersonId"='7870', "Role"='Food' WHERE "EmpId" = '50';
```

7.3 Delete an existing record

```
DELETE FROM "Person" WHERE "PersonId"='250';
```

7.4 Function to populate Request table

```
do $$
DECLARE
noOfRecords INT:=1000;
startTime timestamp:= now();
typeArray text[]:='"Health","Banking","Transport","Food"';
modVal INT:=0;
empid INT :=0;
endTime timestamp:= now();
BEGIN
FOR i IN 1..noOfRecords LOOP
modVal=mod(i,4);
empid = floor(random()*100+3);
if modVal=0 THEN
modVal=4;
end if;
startTime=date (timestamp '2022-01-01' + random() * (timestamp '2022-11-30' - timestamp '2022-01-01')) + time '10:00:00'
+ random() * INTERVAL '8 hours';
endTime = startTime + time '01:00:00' + random() * INTERVAL '1 hours';
insert into "Request"("ReqId","ClientId","EmpId","StartTime","EndTime","Category","Status") values(i,i*2,empid,startTime,endTime,
modVal,'Resolved');
end loop;
END
$$;
```

7.5 Fetch Number of Requests pending from each categories

```
SELECT c."Type" AS CategoryName, count(req."ReqId") AS RequestCount
FROM "Request" req
JOIN "Category" c ON c."CategoryId"=req."Category"
WHERE req."Status"='Unresolved'
GROUP BY c."Type";
```

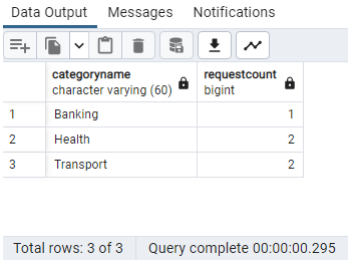

Fig. 8. Select Query

7.6 Fetch the names of Employees with Active Session but not assigned any request

```
SELECT pr."FirstName", pr."LastName", e."Role"
FROM "Employee" e
JOIN "Person" pr ON pr."PersonId" = e."PersonId"
WHERE e."SessionActive" = true AND e."RequestAssigned" = false;
```

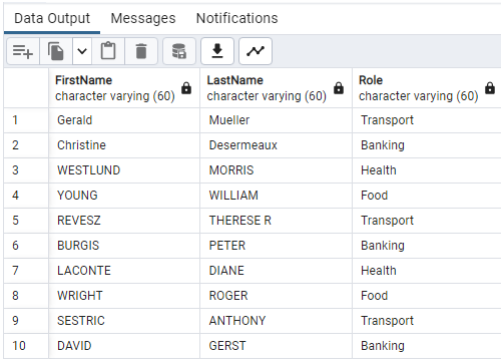

Fig. 9. Select Query

7.7 Fetch Pending Requests with Client Name and Employee Name

```
SELECT CONCAT(pr2."FirstName",' ',pr2."LastName") AS ClientName, CONCAT(pr1."FirstName", ' ', pr1."LastName") AS
EmployeeName, req."StartTime"
FROM "Client" cl
JOIN "Request" req ON req."ClientId" = cl."ClientId"
```

```
JOIN "Employee" ee ON ee."EmpId" = req."EmpId"
JOIN "Person" pr1 ON ee."PersonId" = pr1."PersonId"
JOIN "Person" pr2 ON cl."PersonId" = pr2."PersonId"
WHERE req."Status" = 'Unresolved';
```

Data Output Messages Notifications

	clientname text	employeename text	StartTime timestamp without time zone
1	Edgardo Ramos	Eric Tirschwell	2022-11-29 14:59:40.169679
2	Lee Garrett	Jane Goldberger	2022-11-29 13:47:40.34846
3	SQUIRE DAVID	Shannon Mortimer	2022-11-29 15:38:23.445739
4	BERGIN ROBERT	Cynthia Thompson	2022-11-29 13:52:00.669688
5	COHEN ALAN	Lawrence Brown	2022-11-29 13:36:19.479364

Fig. 10. Select Query

7.8 Fetch number of employees with top 5 Hourly wage and age between 30 to 50

```
SELECT ep."HourlyPay", count(ep."EmpId") FROM "EmployeePay" ep
JOIN "Employee" ee ON ee."EmpId"=ep."EmpId"
JOIN "Person" pr ON pr."PersonId" = ee."PersonId"
WHERE Extract(Year from Age(now(), pr."DateofBirth")) < 50
AND Extract(Year from Age(now(), pr."DateofBirth")) > 30
GROUP BY ep."HourlyPay"
ORDER BY 1 DESC FETCH FIRST 5 ROWS ONLY;
```

	HourlyPay integer	count bigint
1	44	17
2	43	13
3	42	13
4	41	11
5	40	8

Fig. 11. Select Query

7.9 Employee with Best average Turn Around time for request processing

```
WITH bestAvg(EmpId, avgTime) AS
(SELECT req."EmpId", avg(req."EndTime"-req."StartTime")
FROM "Request" req
GROUP BY req."EmpId"
ORDER BY 2 FETCH FIRST ROW ONLY)
SELECT pr."FirstName",pr."LastName", bestAvg.avgTime
FROM bestAvg
JOIN "Employee" ee ON ee."EmpId" = bestAvg.EmpId
JOIN "Person" pr ON pr."PersonId" = ee."PersonId";
```

	FirstName character varying (60)	LastName character varying (60)	avgtime interval
1	Alyssa	Levine	01:15:08.935477

Fig. 12. Select Query

7.10 Find all the employee whose salary is more than the average salary of all employees

```
WITH avgSal(averagesalary) AS
(SELECT avg("TotalPay")
FROM "EmployeePay")
SELECT "EmpId","TotalPay"
FROM "EmployeePay", avgSal
WHERE "EmployeePay"."TotalPay" > avgSal.averagesalary;
```

Data Output Messages Notifications

	EmpId integer	TotalPay integer
1	2	13056
2	3	19136
3	5	19656
4	12	14256
5	14	10962
6	15	9752
7	16	18354
8	17	12740
9	18	16887
10	20	13026
11	21	14198
12	26	15295
13	27	15950
14	29	10648
15	32	9612

Total rows: 419 of 419 Query complete 00:00:00.064

Fig. 13. Select Query

7.11 Employees with Maximum salary from each category

```
WITH maxSal(maxSalary, category) AS (select max(epIN."TotalPay"), eIN."Role"
FROM "Employee" eIN
JOIN "EmployeePay" epIN ON epIN."EmpId" = eIN."EmpId"
GROUP BY eIN."Role")
SELECT pr."FirstName",pr."LastName", e."Role" FROM "Employee" e
JOIN "EmployeePay" ep ON ep."EmpId" = e."EmpId"
JOIN "Person" pr ON pr."PersonId" = e."PersonId", maxSal
WHERE ep."TotalPay" = maxSal.maxSalary AND maxSal.category = e."Role";
```

Data Output | Messages | Notifications

	FirstName character varying (60)	LastName character varying (60)	Role character varying (60)
1	ZIFF	DAVID	Transport
2	DENNIS	JAMES	Banking
3	TSUK	ROBERT	Health
4	BURCH	ROBERT	Food

Fig. 14. Select Query

8 QUERY ANALYSIS

8.1 Fetch Active Employees

8.1.1 ***Query:*** SELECT * FROM Employee" WHERE "SessionActive" = True;

8.1.2 ***Analysis:*** Each time a new request comes in, the system searches for an employee who is active and assigns the request to that employee. Though a simple query this was taking 127 microseconds to run.

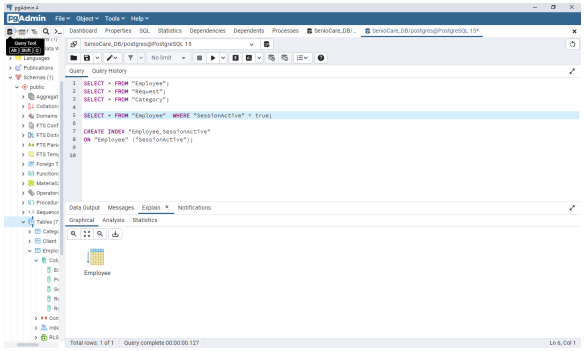

Fig. 15. Select Query before Indexing

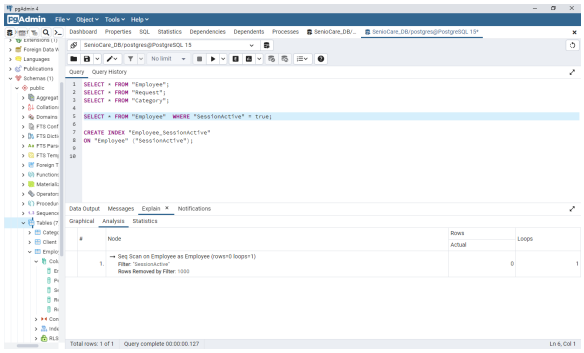

Fig. 16. Select Query before Indexing

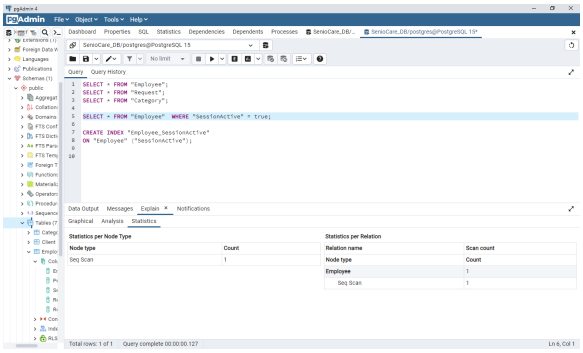

Fig. 17. Select Query before Indexing

In order to optimize the run time an index Employee_SessionActive was created on Employee table.

```
CREATE INDEX "Employee_SessionActive" ON "Employee"("SessionActive");
```

The run time came down from 127 microseconds to 54 microseconds.

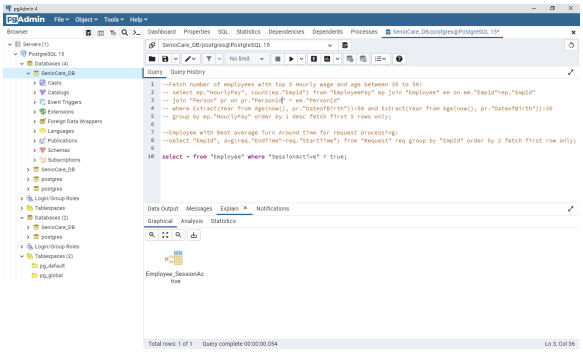

Fig. 18. Select Query after Indexing

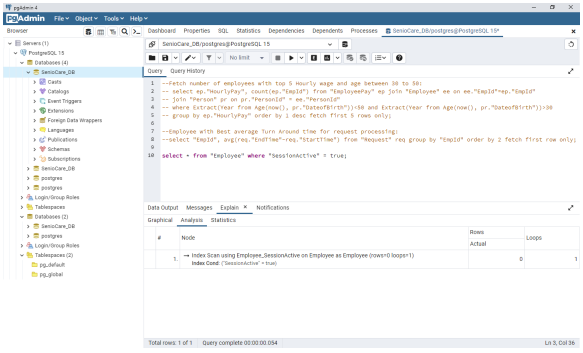

Fig. 19. Select Query after Indexing

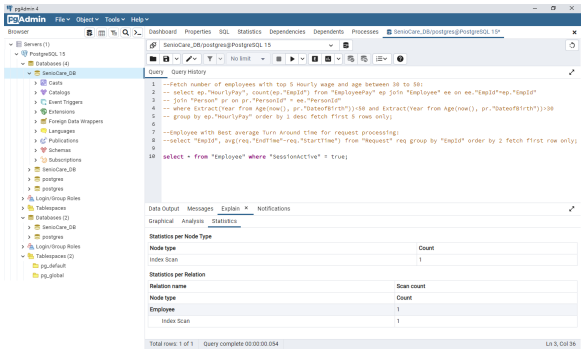

Fig. 20. Select Query after Indexing

8.2 Employee with Best average Turn Around time for request processing

8.2.1 ***Query:*** WITH bestAvg(EmpId, avgTime) AS
(SELECT req."EmpId", avg(req."EndTime"-req."StartTime")
FROM "Request" req
GROUP BY req."EmpId"
ORDER BY 2 FETCH FIRST ROW ONLY)
SELECT pr."FirstName",pr."LastName", bestAvg.avgTime
FROM bestAvg
JOIN "Employee" ee ON ee."EmpId" = bestAvg.EmpId
JOIN "Person" pr ON pr."PersonId" = ee."PersonId";

8.2.2 ***Analysis***. Employee having the best average turn around time was initially taking 507 microseconds to run.

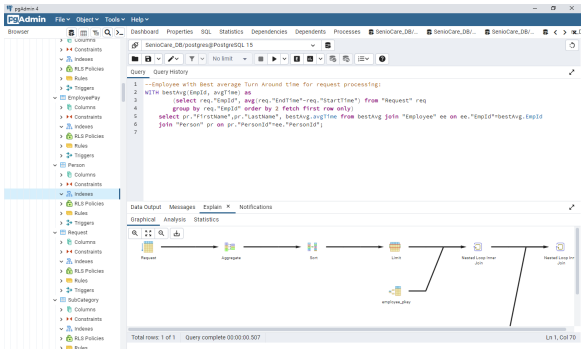

Fig. 21. Select Query before Indexing

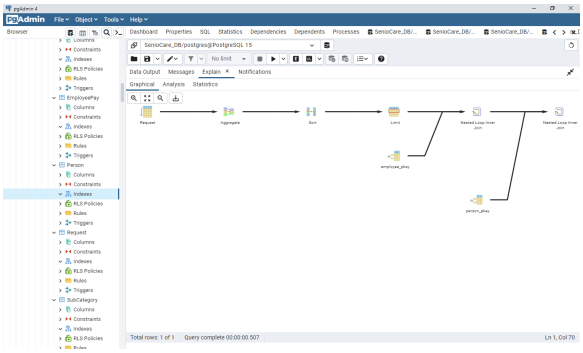

Fig. 22. Analysis Graph before Indexing

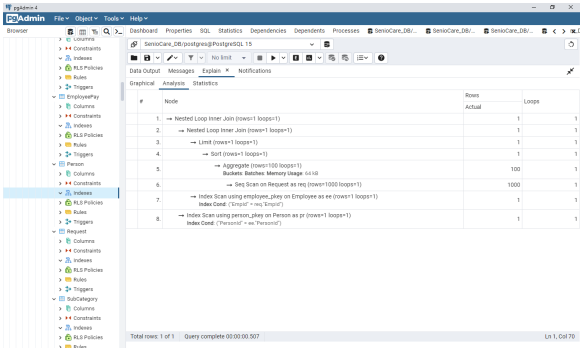

Fig. 23. Analysis before Indexing

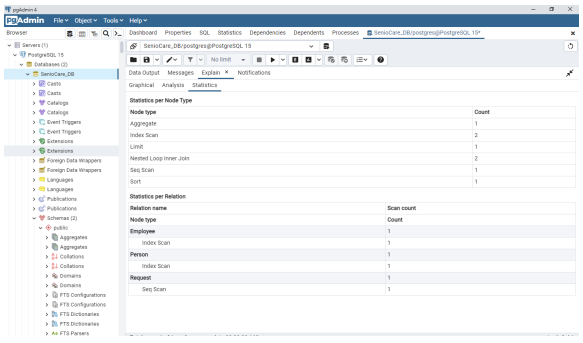

Fig. 24. Statistics before Indexing

In order to optimize the run time an index Employee_EmpId was created on Employee table and Person_PersonID was created on Person table.

CREATE INDEX "Person_PersonId" ON "Person"("PersonId");

CREATE INDEX "EmployeePay_EmpId" ON "EmployeePay"("EmpId"); The run time was significantly reduced from 507 microseconds to 61 microseconds.

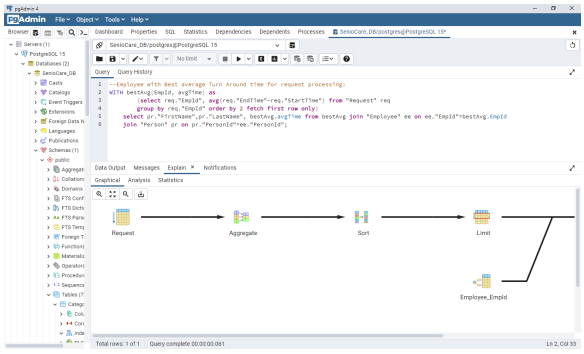

Fig. 25. Select Query after Indexing

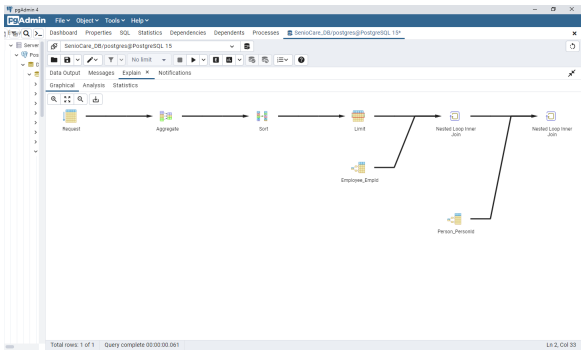

Fig. 26. Analysis Graph after Indexing

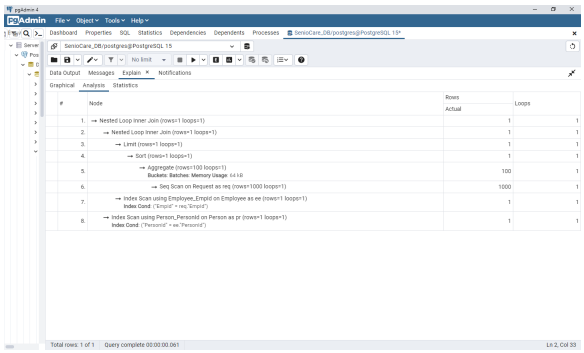

Fig. 27. Analysis after Indexing

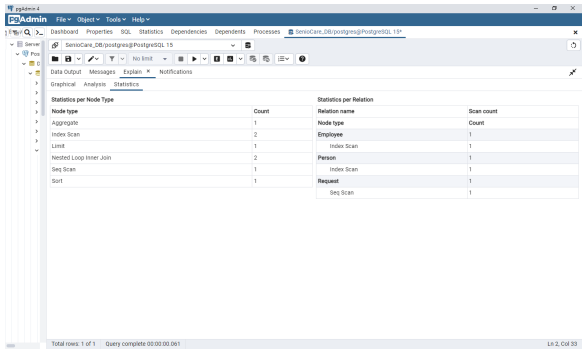

Fig. 28. Statistics after Indexing

8.3 Fetch number of employees with top 5 Hourly wage and age between 30 to 50

8.3.1 ***Query:*** SELECT ep."HourlyPay", count(ep."EmpId") FROM "EmployeePay" ep
JOIN "Employee" ee ON ee."EmpId"=ep."EmpId"
JOIN "Person" pr ON pr."PersonId" = ee."PersonId"
WHERE Extract(Year from Age(now(), pr."DateofBirth")) < 50
AND Extract(Year from Age(now(), pr."DateofBirth")) > 30
GROUP BY ep."HourlyPay"
ORDER BY 1 DESC FETCH FIRST 5 ROWS ONLY;

8.3.2 ***Analysis.*** Initially this query was taking 148 microseconds to run.

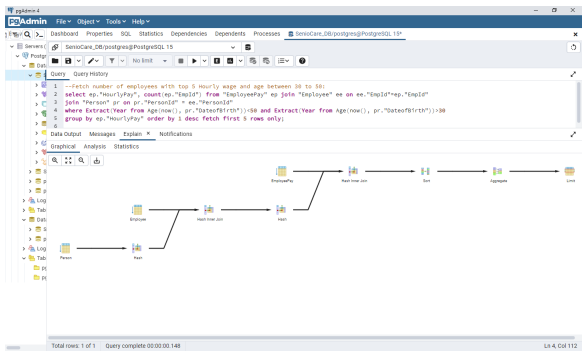

Fig. 29. Select Query before Indexing

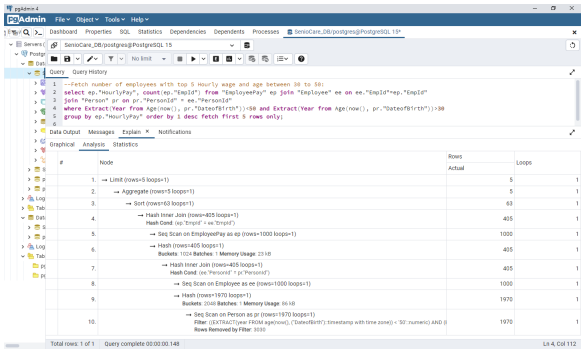

Fig. 30. Select Query before Indexing

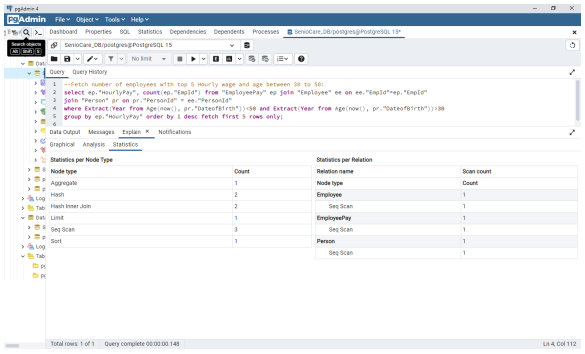

Fig. 31. Select Query before Indexing

After creating the previously stated indexes the run time was significantly reduced from 507 microseconds to 57 microseconds.

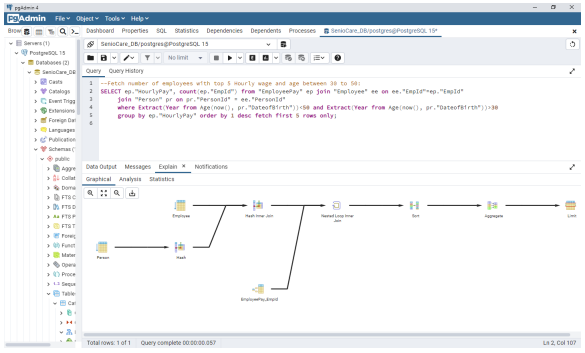

Fig. 32. Select Query after Indexing

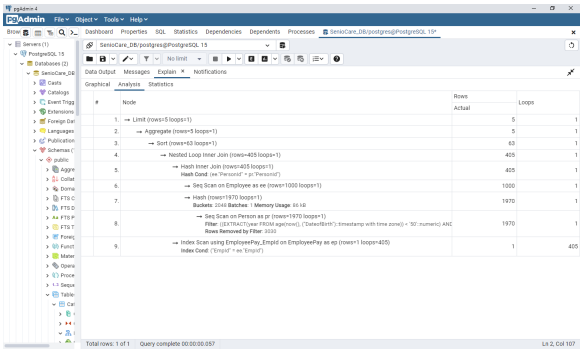

Fig. 33. Select Query after Indexing

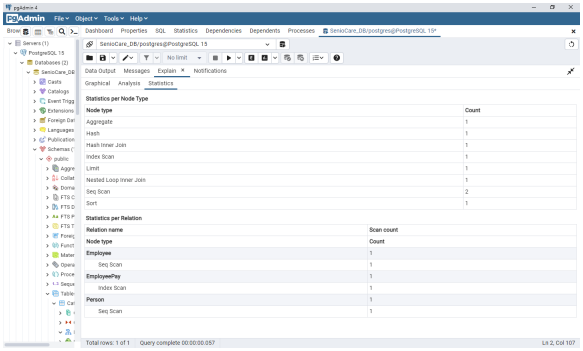

Fig. 34. Select Query after Indexing

8.4 Find all the employee whose salary is more than the average salary of all employees

8.4.1 ***Query:*** WITH avgSal(averagesalary) AS
(SELECT avg("TotalPay")
FROM "EmployeePay")
SELECT "EmpId","TotalPay"
FROM "EmployeePay", avgSal
WHERE "EmployeePay"."TotalPay" > avgSal.averagesalary;

8.4.2 ***Analysis.*** Initially this query was taking 72 microseconds to run.

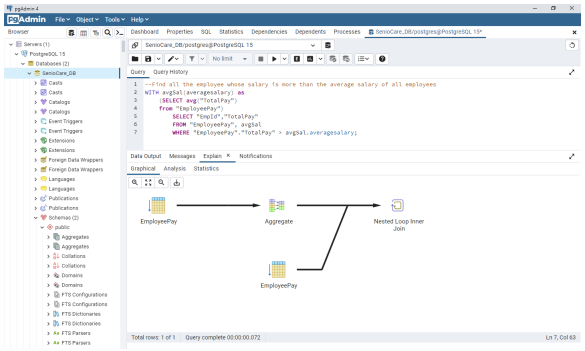

Fig. 35. Select Query before Indexing

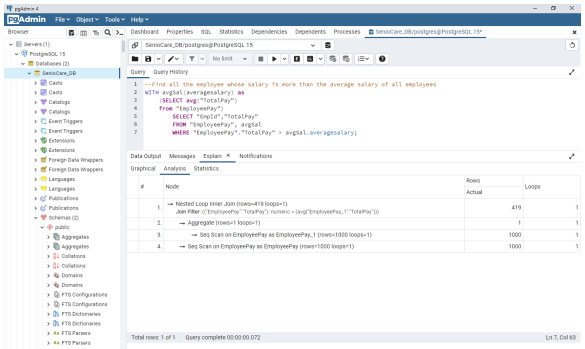

Fig. 36. Select Query before Indexing

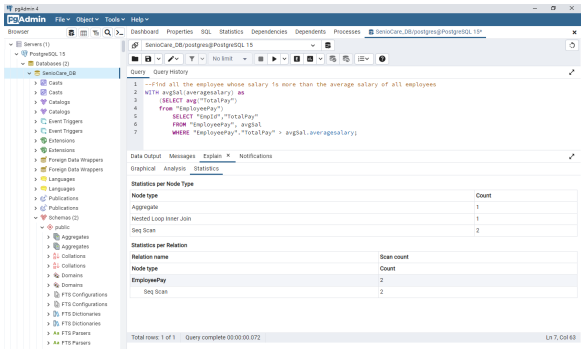

Fig. 37. Select Query before Indexing

After creating the previously stated indexes the run time was significantly reduced from 507 microseconds to 58 microseconds.

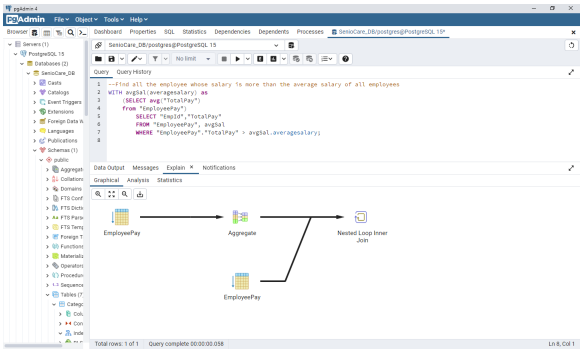

Fig. 38. Select Query after Indexing

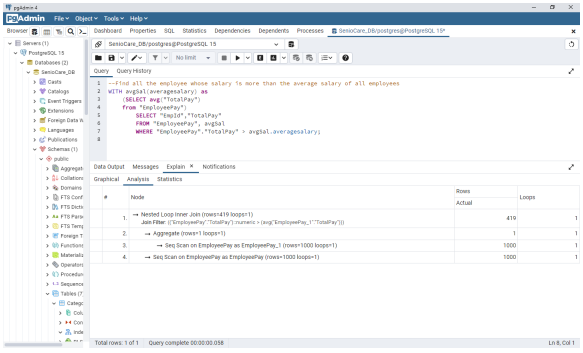

Fig. 39. Select Query after Indexing

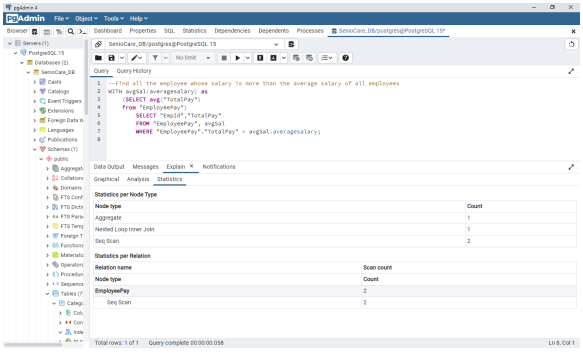

Fig. 40. Select Query after Indexing

8.5 Employees with Maximum salary from each category

WITH maxSal(maxSalary, category) AS (select max(epIN."TotalPay"), eIN."Role"
FROM "Employee" eIN

```
JOIN "EmployeePay" epIN ON epIN."EmpId" = eIN."EmpId"
GROUP BY eIN."Role")
SELECT pr."FirstName",pr."LastName", e."Role" FROM "Employee" e
JOIN "EmployeePay" ep ON ep."EmpId" = e."EmpId"
JOIN "Person" pr ON pr."PersonId" = e."PersonId", maxSal
WHERE ep."TotalPay" = maxSal.maxSalary AND maxSal.category = e."Role";
```

8.5.1 ***Analysis.*** This query was initially taking 164 microseconds to run.

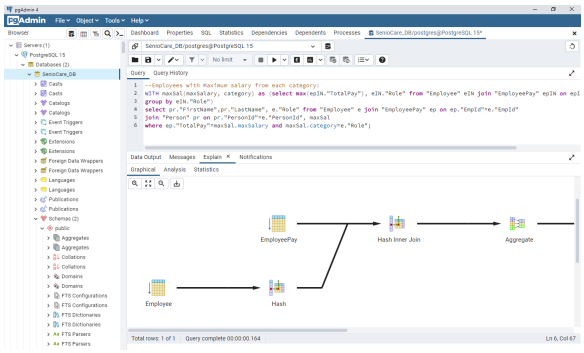

Fig. 41. Select Query before Indexing

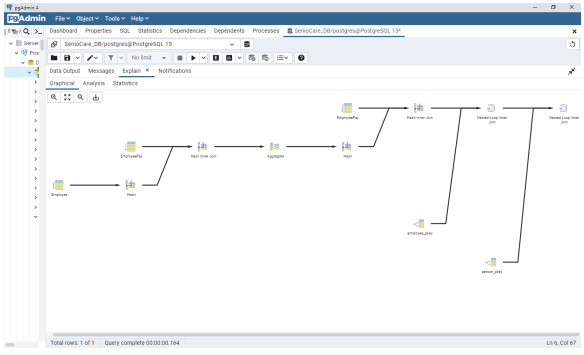

Fig. 42. Analysis Graph before Indexing

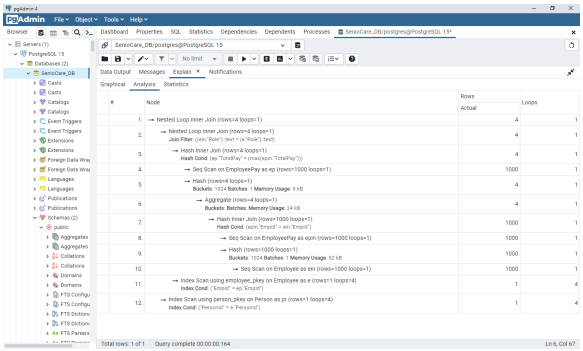

Fig. 43. Analysis before Indexing

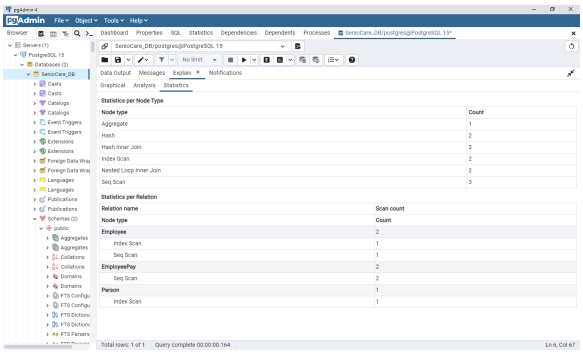

Fig. 44. Statistics before Indexing

In order to optimize the run time an index Employee_Role was created on Employee table.

```
CREATE INDEX "Employee_Role" ON "Employee" ("Role");
```

The run time was significantly reduced from 507 microseconds to 61 microseconds.

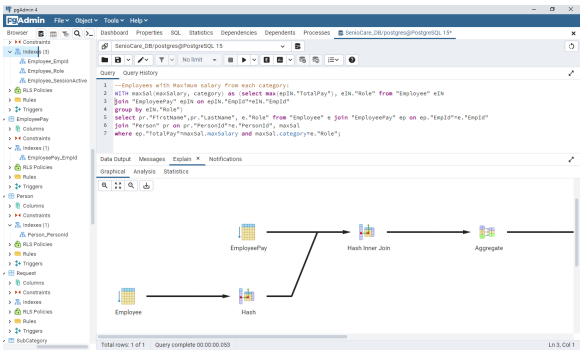

Fig. 45. Select Query after Indexing

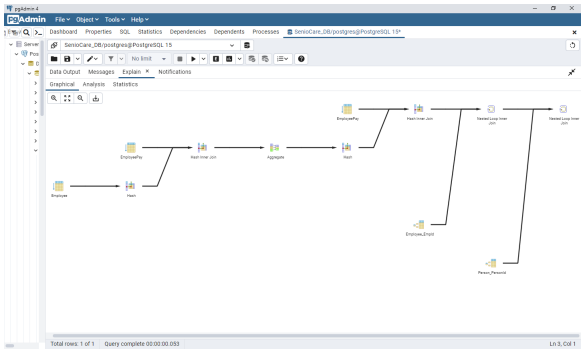

Fig. 46. Analysis Graph after Indexing

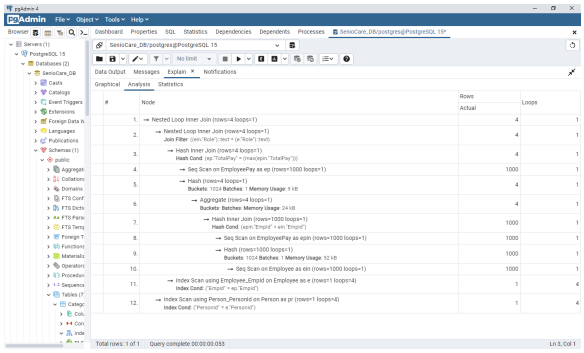

Fig. 47. Analysis after Indexing

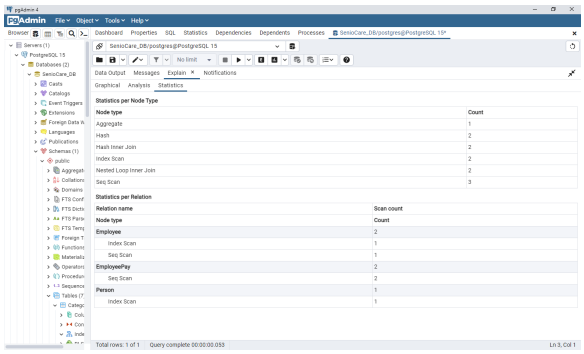

Fig. 48. Statistics after Indexing

9 DEVELOPMENT PROCESS

The project was developed in two phases.

9.1 Phase One

- The idea of the project was formulated and the development areas were identified.
- Tables and attribute names were decided along with the datatype of each attribute.
- Constraints and functional dependencies identified.
- Entity Relationship Diagram was created.
- Limited amount of data was inserted to satisfy milestone 1.
- Milestone 1 document was created.

9.2 Phase Two

- Normalization was performed to ensure each of the Table follows the BOYCE-CODD NORMAL FORM (BCNF).
- Scripts were written for data insertion in each of the tables.
- Multiple queries were created and executed on the database.
- Query Analysis was performed on complex queries and indexing was done in order to reduce the run-time of complex queries.
- User Interface was developed.
  URL : https://github.com/dip3634/seniocare-demo
- A presentation was created for demo.
- Project Document was created.

10 CONTRIBUTIONS

- Database Scripts for insertion of records into tables, development of complex queries were handled by Yadvender Singh.
- Normalization of tables, indexing, analysis of complex queries, document and presentation creation were handled by Shawna Saha.
- Insertion of data into tables, User Interface design was handled by Diptangshu De.
- All authors contributed equally on database design and entity relation model.

11 ACKNOWLEDGEMENTS

We would like to take advantage of this opportunity to express our gratitude to Dr.Sreyasee Das Bhattacharjee, our lecturer, and Thelma Melissa Gomes, our teaching assistant, for their insightful comments, patience, and time.