Chapter 6

Implementation of the Proposed Model

6.1 System Requirements

The proposed model is implemented using PostgreSQL version 9.5 for the underlying database and Java NetBeans Integrated Development Environment version 8.2 for the design of application interface. All the experiments are carried out on a system with 8GB of RAM and Intel(R)Core(TM) i5-3210M CPU@2.5GHz running on Windows Operating System. The brief descriptions about these softwares are covered in the subsequent sub-sections.

6.1.1 PostgreSQL

PostgreSQL or Postgres [6] is a leading object relational DBMS (ORDBMS) which is available as open source and have the majorly emphasized on extensibility, robustness and scalability. It supports all the major operating system and can successfully operate in environment ranging from a single user to multi-user internet-interfaced with concurrent access environment. It gives a strong security support along with extensive data storing capabilities. It is developed by a PostgreSQL global development group and is released open source licensing terms. It offers a very vibrant community support and have a development background of over two decades.

The features that PostgreSQL implemented were fully defining types and also describe relationships which was something earlier maintained by users. The ability of PostgreSQL to "understand" relations and naturally retrieve data from related tables by using rules makes this a thing of future.

Multiversion Concurrency Control [93] is a technique implemented by PostgreSQL to maintain the ACID (Atomicity, Consistency, Isolation, and Durability). It gives to the transaction a snapshot of the database to work with where the changes made by current transactions are not visible to other transaction until they are committed. The Multi version concurrency control and easiness in handling high volumes of data added with functionalities like Nested transactions [94], [95], Patch access, Temporal support via time dependent data types, powerful set of operations and extensive data types gives it the extra edge over other popular Database vendors.
PostgreSQL provides us with the capability of in-build binary replication. The changes made at a node are passed on asynchronously to the replicas also. Also, while reading due to the presence of these replica nodes the read traffic is divided and query performance is enhanced. PostgreSQL hosts a mix of synchronous and asynchronous servers which provides the capability to define the transaction durability according to the need. The basic architecture of the replication mechanism is node based. There are a number of synchronous and asynchronous masters present in the line. The first one in the line is always used and if the first one is not free then the next free standby server is put into use.

Table 6.1 gives the information about the database size specifications of the PostgreSQL. The table gives the details about the maximum size of tables, maximum number of rows, maximum number of columns, maximum field size, indexes per table and the maximum size of database.

**TABLE 6.1: POSTGRESQL SIZE SPECIFICATIONS**

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Database Size</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Max Table Size</td>
<td>32 TB</td>
</tr>
<tr>
<td>Max Row Size</td>
<td>1.6 TB</td>
</tr>
<tr>
<td>Max Field Size</td>
<td>1 GB</td>
</tr>
<tr>
<td>Max Rows per Table</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Max Columns per Table</td>
<td>250 - 1600 depends on column types</td>
</tr>
<tr>
<td>Max Indexes per Table</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>

Indexing [71] in PostgreSQL is quite extensive with support for a B-tree and Hash indexes. B-tree indexes are the best suited for almost all the database applications. In addition to these, PostgreSQL also provides the following indexing techniques as well:
(i) Generalized Search trees (GiST).

(ii) Generalized Inverted Index (GIN).

(iii) Space Partitioned GiST (SP-GiST).

(iv) Block Range Indexing (BRIN).

Each index type follows a different approach that is well-matched with the different types of queries. In PostgreSQL database, “CREATE INDEX” command is used to create B-tree index on the database table. Apart from the built-in support for indexing, PostgreSQL also provides the facility to create and implement user defined indexes. PostgreSQL is the first open source database which have implemented BRIN (Block Range Indexing) which provides an efficient way to index large ranges of data with comparatively less storage needed for the index itself.

PostgreSQL is hugely popular among the developers not only because it has great capabilities but also because it is also very easy to use. PostgreSQL provides with two different tools: Command Line Interface and PgAdmin.

Fig. 6.1: Command Line Interface in PostgreSQL

Figure 6.1 shows the command line tool provided by the PostgreSQL. Basically, it is a traditional approach and provides the same functionality as provided by the Graphical user interface (GUI). It does not have any distinguishable feature and is more of an obligation. Since majority of the users prefer the GUI based interactive
tool, this CMD interface is used only for the experimental purposes. To operate this, we need to enter certain information like

(i) **Server** - By default it is the local host, otherwise, the user can type in the IP address of the server machine.

(ii) **Database** - By default it is the Postgres, otherwise, one can include the name of valid existing database.

(iii) **Port** – The default port number is 5433. Else, one can set the correct port on which PostgreSQL is installed.

(iv) **Username** – The default username is postgres. The user can set any valid username.

(v) **Password** – The password is set during user creation and the same will be used thereafter.

Fig. 6.2: PgAdmin III Interface

Figure 6.2 shows the PgAdmin III tool interface. It is one of the most popular, easy and extremely rich in features development platform for the PostgreSQL database. It has been developed using JQuery and python language. The tool provides
a user-friendly interface with a built-in SQL query editor and allows to perform various functions like navigating between databases, browsing the schemas, table structure, objects and keeping track of all transactions going in the particular database.

Fig. 6.3: Object Browser of PgAdmin III

The GUI provides a user-friendly way to browse objects. The area under black box shown in Figure 6.3 is called object browser. The SQL Query editor is an application program that runs the query and shows the output. The SQL Query editor window has two panes: the first is the query editor pane and other is the output pane, as seen in Figure 6.4. The SQL query editor pane works in two different modes. In the first mode, the user enters the query manually. Whereas, the second mode is Graphical Query editor in which the user generates the query graphically. The output pane is used to display the result of the query.

Fig. 6.4: SQL Editor
The output pane has four tabs. The first tab is DATA OUTPUT which shows the output of the query. The second is captioned as EXPLAIN and it describes the relationship among the table. The third tab, MESSAGES shows the error messages and the count of rows retrieved in the result. HISTORY is the last tab which displays the earlier executed queries with their execution time.

Some of the latest and powerful features of PostgreSQL are:

(i) Foreign Data Wrappers - PostgreSQL has the capability to interface itself with other systems and can extract or retrieve data using the concept of FDW (Foreign Data Wrappers). Foreign data wrappers can take various forms like that of some file system, some other relational DBMS, web application etc. The main advantage of foreign data wrappers is that they can act as a pool of resources. Simple queries can use the foreign data wrappers to combine multiple sources of data together. This makes PostgreSQL one of the most extensible databases.

(ii) Interfaces - One of the major reason for the growing popularity of PostgreSQL is its capability to interface and a wide support through language libraries. Some of the most important and crucial interfaces are:

   a) Libpq (for C language).
   b) ECPG (for embedded C).
   c) Libpqxx (for C++ language).
   d) JDBC (Java interface).

Since it enjoys such a large support from every platform and language library, its use as a database for the next generation application is trivial. PostgreSQL is a really powerful and flexible database. It enjoys a very active community support and is moving towards being the database for the applications of next generation.

PostgreSQL has a very long feature list which is still growing. Since it is maintained and developed by its users, many flavors of the database are now into existence and each provides a varied set of functionalities which are developed in
response to the specific problems faced by the developer. PostgreSQL is considered ideal for Business intelligence and Data warehousing as both of these require past and future query results.

PostgreSQL extends supports for the efficient management of temporal data also. PostgreSQL provides temporal support through time dependent data types. It handles time zones very intelligently through 8601 standards. It provides very efficient data types capable of storing time related values like past values present and future values.

The timestamping capability with a format of date followed by time makes it very easy for PostgreSQL to handle temporal data. PostgreSQL handles temporal data very efficiently with the help of built-in time and interval date types such as tsrange, date and tstzrange. Their power comes from the ability to store an interval up to 178 million years with a precision up to 14 decimal digits.

### 6.1.2 Java NetBeans

NetBeans (originally called Xelfi) was initiated as a student project in 1996 with the objective to code a Delphi-like Java Integrated Development Environment in Java language itself. It was the first Java IDE written in Java, with its first pre-releases in 1997. At that time, the Java IDE space was unexplored zone. Therefore, the project fascinated plenty of interest and hence it was decided to market it as a commercial product.

An open source project is an existing unit that requires time to find the perfect equilibrium of people and contribution. Its growth is always an ongoing process. Subsequent releases continued to build on the success of previous releases and the demands of a changing industry. Table 6.2 highlighted all the earlier and current releases of NetBeans.

<table>
<thead>
<tr>
<th>Version</th>
<th>Salient Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBeans 3.2</td>
<td>Very initial stage. Project tried to find its feet.</td>
</tr>
<tr>
<td>NetBeans 3.5</td>
<td>Huge strides in performance</td>
</tr>
</tbody>
</table>
NetBeans 3.6  windowing system and property sheet were re-implemented
NetBeans 4.0  It is possible to replace infrastructure
NetBeans 4.1  J2EE support
NetBeans 5.0  support for developing IDE modules, CVS support, GUI builder
NetBeans 6.0  Productivity improved by integration of external NetBeans product
NetBeans 7.0  Support JDK 7 and JavaFX 2.0, editor tools,
NetBeans 7.2  Support HTML 5
NetBeans 8.0  GlassFish 4.1 and Tomcat 8.0.15 bundled with the IDE
NetBeans 8.1  Improved Git support, JRE bundled with HTML 5, PHP, C/C++.
NetBeans 8.2  Experimental ECMAScript 7 and oracle JET Support

NetBeans IDE is recognized as the original free Java IDE. The NetBeans IDE is compatible for several languages (PHP, JavaFX, C/C++, JavaScript, etc.) and frameworks. NetBeans is an open-source project devoted to provide rock solid software development products that serves the needs of developers and the users.

Fig. 6.5: NetBeans 8.0.2 GUI
It enables them to develop these products quickly, proficiently and effortlessly by enhancing the strengths of the Java platform and other relevant industry standards. A GUI of NetBeans 8.0.2 is shown in Fig. 6.5.

NetBeans was made open source by Sun Microsystems in June 2000. They were the project sponsor until January 2010 when Sun Microsystems became a subsidiary of Oracle. The two base products, the NetBeans IDE and NetBeans Platform, are free for commercial and non-commercial use. The source code is made available to anyone for reuse as per the terms of use. Anyone can easily communicate with the database by using database connection commands. The NetBeans project is an open source project. Some of the salient features of NetBeans are as follows:

(i) The NetBeans Platform and IDE are available free of cost for the commercial and non-commercial use that makes it popular among available

(ii) The source code for both is available to everyone for reuse if they are compatible with the license. As NetBeans platform and IDE are developed in public domain, the decisions are made through consensus.

(iii) Any individual or firm may add code to the project. It ensures that the software meets the needs of its users by making the users participate in its creation. It ensures that the software is modifiable by the users according to their changing needs.

(iv) This approach is perfectly applicable for geographically distributed software development. This approach helps in creating the software in such a way that high quality by peer review and social dynamics is ensured.

(v) It avoids the hierarchy and organizational politics while developing the software, which can hamper the efficiency of some commercial development efforts.

(vi) With its ever-improving Java Editor, many amusing features and a widespread range of tools, templates and samples in NetBeans IDE establishes the benchmark for developing with cutting edge technologies.
NetBeans IDE is the official IDE for Java 8. With its editors, code analyzers, and converters, we can swiftly and seamlessly advance our applications to use new Java 8 language constructs, such as lambdas, functional operations, and method references. Batch analyzers and converters are available for searching through numerous applications simultaneously, and matching patterns for adaptation to new Java 8 language constructs.

The NetBeans Editor indents lines, ties words and brackets, and acmes source code both syntactically and semantically. It lets us effortlessly reflect the code, with an array of handy and powerful tools, while at the same time also offers code templates, coding tips, and code generators.

The editor supports many languages from Java, C/C++, XML and HTML, to PHP, Groovy, Javadoc, JavaScript and JSP. As it is extensible, support for many other languages can be plugged in. The editor and drag-and-drop tools in the IDE enable the quick and smooth designing of GUIs for different applications. NetBeans GUI Builder robotically takes care of proper spacing, alignment and in-place editing.

NetBeans IDE delivers different views of the data like from multiple project windows to helpful tools for setting up the applications and handling them efficiently. It lets us drill down the data very quickly and easily. The code is well organized. When new developers link into the project, they can comprehend the structure of the application as the code is well-organized.

NetBeans offers static analysis tools for recognizing and fixing common problems in Java code. The NetBeans Debugger provides features like placing breakpoints in our source code, adding field watches, stepping through our code, running into methods, taking snapshots and monitoring execution as it occurs.

The NetBeans Profiler provides skilled support for optimizing the application's speed and memory usage, and helps in building reliable and scalable applications. NetBeans IDE comprises a visual debugger for Java SE applications, which lets us fix user interfaces without viewing into source code.
6.2 Data Set

In view of evaluating the proposed tuple timestamp multiple historical relation data model, a dataset of one lakh employees for any hypothetical organization is used and several queries are executed on the data set for testing the accuracy and performance of the data model. The dataset made available to us was in Excel sheet. Hence, the dataset need to be pre-processed to make it compatible for the database use.

The pre-processing of this employee data set is done to convert it into specific usable format. The various steps involved in pre-processing of the dataset are depicted in Figure 6.6. Pre-processing is performed in three phases:

(i) **Conversion of data values:** As the dataset initially available was in MS Excel format, it should be converted to the required format. The output of this phase of pre-processing is the dataset in comma separated values (CSV) format where a comma is used to separate the values of the attributes.

(ii) **Addition of temporal component:** The time range data type “tsrange” is used to time-stamp the tuples of the temporal relation and it is used to show the temporal evolution of an object. A timestamp is used to show the validity period of any event which holds the value of valid start time and valid end time.

(iii) **Splitting of tables:** The original temporal relations are decomposed into sub-relations to enable the extermination of redundancy that has occurred due to presence of static and dynamic attributes together. The dataset that is considered
in this model is temporally heterogeneous in nature. It indicates that the dynamic attributes in the database varies asynchronously. Therefore, if these heterogeneous dynamic attributes are put together in a relation, they shall cause lot of redundancy. Hence, all the static attributes are placed in single relation. Whereas, separate independent relations are created for every dynamic attribute. This is done due to the reason that the values of these temporal attributes vary at different time points.

6.3 Data Flow Diagram

A data flow diagram is a graphical demonstration of the data flow in the system model. It is frequently used as a first step to generate an outline of the system, which can later be explained. It can also be used for the visualization of data flow in the model and shows the type of input and output information of the system. This also includes the information about where the data will be stored and the direction of the flow of data. It does not represent the timing of process or the information about whether processes will operate in sequence or in parallel.

Fig. 6.7: Data Flow Diagram
As shown in Figure 6.7, the Data Flow Diagram for the proposed model is divided into three sections:

(i) **User Interface:** The user interface is designed for accepting query as the input. The designed form with various related fields is filled by the user to query the database. All the data manipulation queries used for the insertion, modification and retrieval of the temporal records are executed by the use of this interface.

(ii) **Java NetBeans IDE:** After the submission of query by the user, it will be automatically checked by the validity checker. Valid time is used as the time element in this proposed temporal model. Therefore, the validity of the time is checked by comparing the time range mentioned in the query with the time range of the respective temporal relation in the database. If the time point or the time range is valid, then the query will be sent to the third section for its processing. Otherwise, a proper notification with all the information related to the error occurred, is generated and passed on to the user interface.

(iii) **PostgreSQL Database:** This section is responsible for the processing and execution of the query. The processing of the query includes parsing and query optimization. After optimization of query, an optimal plan is prepared for efficient execution of respective queries. Conventional query optimization technique considers that query plans are equated on the basis of single cost metric, execution time. The results of the query shall then be passed on to the user interface. The information about the flow of data from the user interface to the database or from database to the user is hidden from the user.

### 6.4 Experimental Results

A temporal database application efficiently stores the time varying data, typically by having some fixed timescale. Several types of temporal queries are executed on the database for performing temporal operations. A temporal value is only compatible with the associated temporal data type, which involves similar date and time fields. Temporal values may not be directly compared and assigned to the non-compatible date and time fields. Implicit type conversions can occur in all the temporal expressions.
The set of temporal operations includes insert operation, update operation and retrieval of temporal information from one or more relations. All these operations are well implemented in the proposed data model. The delete operation is not permitted in the temporal databases. The model implemented all the constraints successfully in view of maintaining the data integrity and consistency. The following sub-section describes the different temporal operations with suitable examples.

### 6.4.1 Insert Operation

Insert operation in this model is performed by entering the new data values for the various static and dynamic attributes in the temporal relation. Insertion of new records takes place in the current relations only. Insert statement includes value of non-temporal attributes, temporal attributes and validity time period of an object. A key attribute is automatically generated by the system which is used to reference the particular tuple of a relation. The user will set the valid start time and valid end time of the time range attribute time_range. The examples of the insert operation are included as follows:

**Query 1:** Insertion of basic details of an employee in “employee” relation.

```sql
insert into employees (emp_no, birth_date, first_name, last_name, gender, hire_date)
values(100015,'1992-10-21', 'Yozad', 'Khan', 'M', '2016-10-10');
```

**Query 2:** Insertion of salary details of an employee in “current salary” relation.

```sql
insert into salaryfront (emp_no, salary, time_range) values (100015, 28200, TSRANGE('2016-10-10 00:00:00', '9999-01-01 00:00:00'));
```

**Query 3:** Insertion of title details of an employee in “current title” relation.

```sql
insert into titlefront (emp_no, title, time_range) values (100015, ‘Engineer’, TSRANGE('2016-10-10 00:00:00', '9999-01-01 00:00:00'));
```

**Query 4:** Insertion of department details of an employee in “current emp-department” relation.

```sql
insert into deptempfront (emp_no, dept_no, time_range) values (100015, d005, TSRANGE('2016-10-10 00:00:00', '9999-01-01 00:00:00'));
```
The above insert queries are used to enter the employee details in the database. All these operations are performed with the help of the application developed using Java NetBeans. The snapshot of the above queries is shown in figure 6.8.

The employee number is incremented each time a new insertion is performed. The validations are applied in the field of the form in order to maintain the integrity of data. The user can enter the details in the form very easily as the interface is designed in an efficient way. The salary is calculated according to the hire date entered during insertion of new employee record.

![Fig. 6.8: Snapshot of Insertion in Employee Details](image)

### 6.4.2 Update Operation

Update operation in the proposed model is carried out on the current tables. This will activate the explicit trigger which eventually leads to insertion of the tuple into the current table which holds the new values of temporal attributes. The old values of temporal attributes shall get transferred into the history relation automatically by the action of triggers. The value of time_range attributes are also changed automatically through these triggers. An upper bound of the time_range column set equal to the lower bound of the time_range attribute of the current instance. The examples of the update operation are included as follows:

**Query 5:** Update employee salary where employee number is 100015.

```
UPDATE salaryfront SET salary=32150, time_range=TSRANGE('2016-12-12
```
WHERE emp_no=100015;

**Query 6:** Update employee title where employee number is 100015.

```
UPDATE titlesfront SET title='Senior Engineer', time_range= TSRANGE('2016-12-12 00:00:00', '9999-01-01 00:00:00', '[]')
WHERE emp_no=100015;
```

The snapshots for the queries 5 and 6 are shown in figures 6.9 and 6.10.

**Fig. 6.9:** Snapshot of Update in Salaryfront Table

**Fig. 6.10:** Snapshot of Update in Titlesfront Table
6.4.3 Retrieval Operation

There are three different categories of data retrieval in the proposed model. In the first case, the user wants to extract only the current snapshot of the data which is currently valid in the real world. The current relation is used in this category of retrieval. Therefore, the user can directly query the current table and get the latest instance of data. In the second case, time_period defined in the predicate of the query lies in the time_range of history table. All the past instances of an object are stored in the history table. Therefore, such query will be satisfied by the history table only. In the third case, the query includes a time period that overlaps the time_range attribute values of both the current and history table. Hence, such queries retrieve data from both the tables.

Query 7: Retrieve salary details of an employee from the database where employee number is 10026 and point query date is ‘2002-03-19’

```sql
SELECT emp_no, salary, time_range from salaryfront where emp_no='10026' and lower(time_range)<= '2002-03-19' and upper(time_range)>= '2002-03-19'

UNION

SELECT emp_no, salary, time_range from salaryback where emp_no='10026' and lower(time_range)<= '2002-03-19' and upper(time_range)>= '2002-03-19';
```

Fig. 6.11: Snapshot of Query 7
**Query 8:** Retrieve title details of an employee from titlesfront or titlesback relations where employee number is 10027 and point query date is ‘2002-03-19’

```
SELECT emp_no, title, time_range from titlesfront
WHERE emp_no='10027' and lower(time_range)<='2002-03-19' and upper(time_range)>='2002-03-19'

UNION

SELECT emp_no, title, time_range from titlesback
WHERE emp_no='10027' and lower(time_range)<='2002-03-19' and upper(time_range)>='2002-03-19';
```

![Figure 6.12: Snapshot of Query 8](image)

**Query 9:** Retrieve salary details of an employee from salaryfront or salaryback relations where employee number is 47025 and tsrange is (‘1990-03-15’, ‘1997-12-12’)

```
SELECT emp_no, salary, time_range from salaryfront where emp_no= '47025' and lower(time_range) < '1997-12-12'

UNION

SELECT emp_no, salary, time_range from salaryback where emp_no= '47025' and lower(time_range) < '1997-12-12' and upper(time_range) > '1990-03-15' order by
```
time_range"

**Figure 6.13: Snapshot of Query 9**

**Query 10**: Retrieve department information of an employee from deptempfront or deptempback relations where employee number is 47025 and tsrange is ('1990-03-15', '1997-12-12').

SELECT emp_no, dept_no, time_range from deptempfront where emp_no='47025' and lower(time_range)<'1997-12-12'

UNION

SELECT emp_no, dept_no, time_range from deptempback where emp_no='47025' and lower(time_range)<'1997-12-12' and upper(time_range)>'1990-03-15' order by time_range";

**Fig. 6.14: Snapshot of Query 10**