UNIT II SPATIAL AND TEMPORAL DATABASES 9

Active Databases Model – Design and Implementation Issues - Temporal Databases - Temporal Querying - Spatial Databases: Spatial Data Types, Spatial Operators and Queries – Spatial Indexing and Mining – Applications -– Mobile Databases: Location and Handoff Management, Mobile Transaction Models – Deductive Databases - Multimedia Databases.

TEMPORAL DATABASES

A Temporal Database is a database with built-in support for handling time sensitive data. Usually, databases store information only about the current state, and not about past states. For example, in an employee database if the address or salary of a particular person changes, the database gets updated, the old value is no longer there. However for many applications, it is important to maintain the past or historical values and the time at which the data was updated. That is, the knowledge of evolution is required. That is where temporal databases are useful. It stores information about the past, present and future. Any data that is time dependent is called the temporal data and these are stored in temporal databases.

Temporal Databases store information about states of the real world across time. Temporal Database is a database with built-in support for handling data involving time. It stores information relating to past, present and future time of all events.

Examples Of Temporal Databases

- **Healthcare Systems:** Doctors need the patients' health history for proper diagnosis. Information like the time a vaccination was given or the exact time when fever goes high etc.
- **Insurance Systems:** Information about claims, accident history, time when policies are in effect needs to be maintained.
- **Reservation Systems:** Date and time of all reservations is important.

Temporal Aspects

There are two different aspects of time in temporal databases.

- **Valid Time:** Time period during which a fact is true in real world, provided to the system.
- **Transaction Time:** Time period during which a fact is stored in the database, based on transaction serialization order and is the timestamp generated automatically by the system.

Temporal Relation

Temporal Relation is one where each tuple has associated time; either valid time or transaction time or both associated with it.

- **Uni-Temporal Relations:** Has one axis of time, either Valid Time or Transaction Time.
- **Bi-Temporal Relations:** Has both axis of time – Valid time and Transaction time. It includes Valid Start Time, Valid End Time, Transaction Start Time, Transaction End Time.

Valid Time Example

Now let's see an example of a person, John:

- John was born on April 3, 1992 in Chennai.
- His father registered his birth after three days on April 6, 1992.
- John did his entire schooling and college in Chennai.
- He got a job in Mumbai and shifted to Mumbai on June 21, 2015.
- He registered his change of address only on Jan 10, 2016.

John's Data In Non-Temporal Database

In a non-temporal database, John's address is entered as Chennai from 1992. When he registers his new address in 2016, the database gets updated and the address field now shows his Mumbai address. The previous Chennai address details will not be available. So, it will be difficult to find out exactly when he was living in Chennai and when he moved to Mumbai.

Date	Real world event	Address
April 3, 1992	John is born	
April 6, 1992	John's father registered his birth	Chennai
June 21, 2015	John gets a job	Chennai
Jan 10, 2016	John registers his new address	Mumbai

Uni-Temporal Relation (Adding Valid Time To John's Data)

To make the above example a temporal database, we'll be adding the time aspect also to the database. First let's add the valid time which is the time for which a fact is true in real world. Valid time is the time for which a fact is true in the real world. A valid time period may be in the past, span the current time, or occur in the future.

The valid time temporal database contents look like this:

Name, City, Valid From, Valid Till

In our example, John was born on 3rd April 1992. Even though his father registered his birth three days later, the valid time entry would be 3rd April of 1992. There are two entries for the valid time. The Valid Start Time and the Valid End Time. So in this case 3rd April 1992 is the valid start time. Since we do not know the valid end time we add it as infinity.

Johns father registers his birth on 6th April 1992, a new database entry is made: Person(John, Chennai, 3-Apr-1992, ∞).

Similarly John changes his address to Mumbai on 10th Jan 2016. However, he has been living in Mumbai from 21st June of the previous year. So his valid time entry would be 21 June 2015.

On January 10, 2016 John reports his new address in Mumbai: Person(John, Mumbai, 21-June-2015, ∞).

The original entry is updated.

The table will look something like this with two additional entries:

Name	City	Valid From	Valid Till
John	Chennai	April 3, 1992	June 20, 2015
John	Mumbai	June 21, 2015	∞

Table:Uni-temporal Database

Bi-Temporal Relation (John's Data Using Both Valid And Transaction Time)

Next we'll see a bi-temporal database which includes both the valid time and transaction time. Transaction time records the time period during which a database entry is made. So, now the database will have four additional entries: the valid from, valid till, transaction entered and transaction superseded.

The database contents look like this:

Name, City, Valid From, Valid Till, Entered, Superseded

First, when John's father records his birth the valid start time would be 3rd April 1992, his actual birth date. However, the transaction entered time would be 6th April 1992.

Johns father registers his birth on 6th April 1992:

Person(John, Chennai, 3-Apr-1992, ∞, 6-Apr-1992, ∞).

Similarly, when John registers his change of address in Mumbai, a new entry is made. The valid from time for this entry is 21st June 2015, the actual date from which he started living in Mumbai. whereas the transaction entered time would be 10th January 2016. We do not know how long he'll be living in Mumbai. So the transaction end time

and the valid end time would be infinity. At the same time the original entry is updated with the valid till time and the transaction superseded time.

On January 10, 2016 John reports his new address in Mumbai:

Person(John, Mumbai, 21-June-2015, ∞, 10-Jan-2016, ∞).

The original entry is updated.

Person(John, Chennai, 3-Apr-1992, 20-June-2015, 6-Apr-1992, 10-Jan-2016).

Now the database looks something like this:

Name	City	Valid From	Valid Till	Entered	Superseded

Name	City	Valid From	Valid Till	Entered	Superseded
John	Chennai	April 3, 1992	June 20, 2015	April 6, 1992	Jan 10, 2016
John	Mumbai	June 21, 2015	∞	Jan 10, 2016	∞

Bi-temporal Database

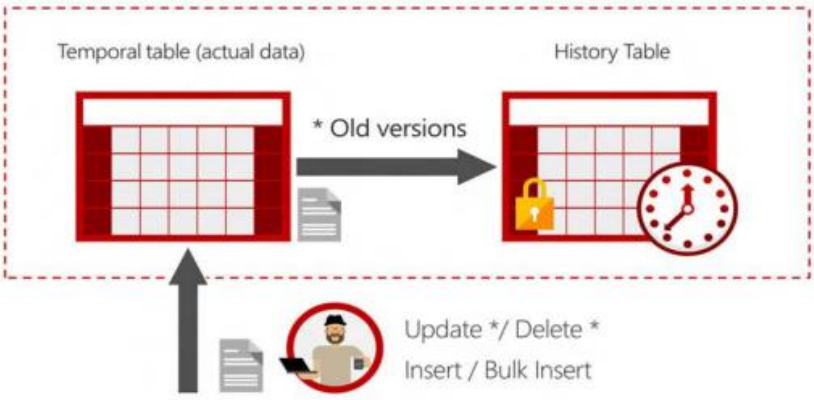
Advantages

The main advantages of this bi-temporal relations is that it provides historical and roll back information. For example, you can get the result for a query on John's history, like: Where did John live in the year 2001?. The result for this query can be got with the valid time entry. The transaction time entry is important to get the rollback information.

- Historical Information – Valid Time.
- Rollback Information – Transaction Time.

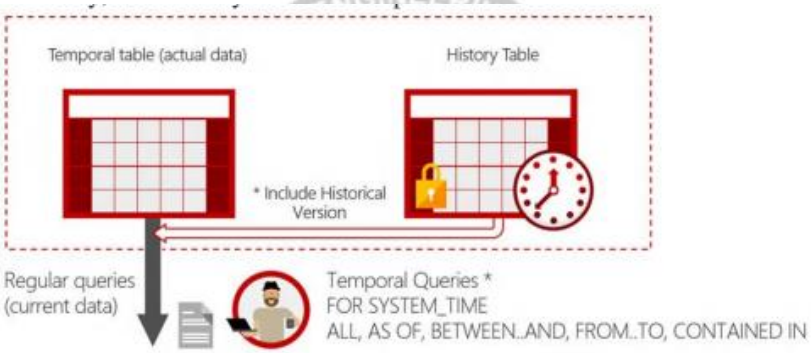
Temporal Query

A TemporalQuery can be used to retrieve information from a temporal-based object. When a temporal table is created in SQL Server, a history table is created behind the scenes. The main table contains the records as they exist at the current point in time and the history table contains all the previous versions of records. You can query the main table as normal or add temporal clauses to your query to find historical records.



Querying Temporal Tables

There are two main ways to query the history table. The first is looking at previous versions of the table by adding time-based clauses to your queries. The second is to look into the history table manually, which lets you see all the previous versions of records.



Creating a Temporal Table

First let's consider a normal table that we can use as our sample main table. For this post, we'll work with a basic person record:

```
create table dbo.Person
(
    [PersonId] int not null primary key clustered,
    [Name] nvarchar(max) not null,
    [Email] nvarchar(max) null,
    [Address] nvarchar(max) null,
    [PhoneNumber] nvarchar(max) null
)
```

That table is not temporal, but having the normal definition to compare it to will highlight what we need to make it temporal:

```
create table dbo.Person
(
    [PersonId] int not null primary key clustered identity(1,1),
```

```
    [Name] nvarchar(max) not null,
    [Email] nvarchar(max) null,
    [Address] nvarchar(max) null,
    [PhoneNumber] nvarchar(max) null, [SysStartTime] datetime2 generated always as row start hidden not null,
    [SysEndTime] datetime2 generated always as row end hidden not null,
    Period for system_time (SysStartTime, SysEndTime)
)
with (system_versioning = on (history_table = Person_History))
```

There are two main changes. First, we added SysStartTime and SysEndTime as generated columns then used that to create the Period column. These are required columns for temporal tables. Making the start and end time columns hidden is optional, but can help to hide the versioning when it's not needed. Second, we added with (system_versioning = on (...)) to the end of the statement. This will create the history table using the Period column defined above. The default naming for the history table is kind of messy, so we also defined what the table should be called. I like the _History suffix, so that's what we will use here.