4-2016

Temporal and Contextual Dependencies in Relational Data Modeling

Rajvardhan Bhaskar Patil
University of Nebraska at Omaha

Temporal and Contextual Dependencies in Relational Data Modeling

By

Rajvardhan Bhaskar Patil

A DISSERTATION

Presented to Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Information Technology

Under the Supervision of Dr. Peter Wolcott and Dr. Zhengxin Chen

Omaha, Nebraska

April, 2016

Temporal and Contextual Dependencies in Relational Data Modeling

Rajvardhan Bhaskar Patil, Ph.D.

University of Nebraska, 2016

Advisors: Dr. Peter Wolcott and Dr. Zhengxin Chen

Although a solid theoretical foundation of relational data modeling has existed for decades, critical reassessment from temporal requirements' perspective reveals shortcomings in its integrity constraints. We identify the need for this work by discussing how existing relational databases fail to ensure correctness of data when the data to be stored is time sensitive. The analysis presented in this work becomes particularly important in present times where, because of relational databases' inadequacy to cater to all the requirements, new forms of database systems such as temporal databases, active databases, real time databases, and NoSQL (non-relational) databases have been introduced. In relational databases, temporal requirements have been dealt with either at application level using scripts or through manual assistance, but no attempts have been made to address them at design level. These requirements are the ones that need *changing metadata* as the time progresses, which remains unsupported by Relational Database Management System (RDBMS) to date.

Starting with shortcomings of data, entity, and referential integrity in relational data modeling, we propose a new form of integrity that works at a more detailed level of granularity. We also present several important concepts including temporal dependency, contextual dependency, and cell level integrity. We then introduce cellular-constraints to implement the proposed integrity and dependencies, and also how they can be incorporated into the relational data model to enable RDBMS to handle temporal requirements in future. Overall, we provide a formal description to address the temporal requirements' problem in relational data model, and design a framework for solving this problem. We have supplemented our proposition using examples, experiments and results.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Before relational databases were introduced, storage of data followed the norms of flat file system. As the flat files had no mechanism to specify the metadata (data about data), querying and lookup of data was difficult. As there was no way to impose any sorts of constraints, flat file system proved inefficient to filter or segregate correct data from the incorrect data. Because of its inability to inspect the incoming data, flat file system also became prone to data redundancy, due to which manipulation of data came with risk of inconsistency. In addition, it was cumbersome to alter the data-type of a field, or add extra fields to an existing file. And finally, when the data resided in more than one file (sometimes in different file-formats), associating the scattered data was difficult. The relational data model was introduced to address the issues raised by the flat files. Semantics of such relational data model is explained in [64], and the importance and distinction between different logical data models (network, hierarchical, and relational) are described in [63], and [74]. Navathe [53] captures this evolution or transition of data models from flat file to relational data model.

The relational data model proposed by Codd [15] was a milestone because it provided a mathematical basis for modeling data, and logical view (away from the physical implementation details) of data based on the notions of sets and relations. The database management system based on relational data model is termed Relational Database Management System (RDBMS). Relational data model uses a collection of tables to represent or project the data [65, 16]. Relational data model ensures that the data is accepted or denoted in structural format making it simpler to query and locate them. Relational data model also made it easier to manipulate the data, and interrelate them across tables using relationships (cardinality mapping). Furthermore, integrity constraints were proposed to classify the data into acceptable and non-acceptable datasets, where only the data falling in acceptable domain was stored, while the rest was rejected. In addition, the normalization process made sure that data redundancy is eliminated, and transactions do not introduce anomalies (inconsistencies) into the database. All this was made possible because of the Relational data model's ability to impose a schema, which helped determine the structure and nature of incoming data in advance. Batini et. al. [6] talks about the importance of this schema, and its integration.

Although the relational data model surpasses the flat file system in many ways, it fails to address temporal requirements, where the data are time sensitive. When the datum is time sensitive, validity and past imprints (instances) of the data become important in deciding its authenticity. Relational data model fails to retain the correctness of such timestamped data, as it neither considers the time factor nor the previous instances of the given data. Therefore, RDBMS has to currently rely either on manual assistance or external scripts to address such time-oriented requirements.

In relational data model, there are five constraints to ensure the correctness of data that need to be stored. These constraints are: not-null, primary key, foreign key, unique, and check [refer 'Appendix-A' for definition on the database terminologies]. These constraints are often sufficient to ensure authenticity of data, but they fail to reflect the legitimacy of data when their temporal aspect is taken into account. The existing constraints do not take into consideration the timestamp of when the data were inserted or last updated. As a result, they cannot capture the contextual dependency that can exist between the incoming datum and its *own* previous instances (or past imprints). If the inserts and updates are untimely, then the database might be put into an inconsistent state. For example: the values appearing in the graduation-year column of a student's record have to follow this temporal ordering, with a minimum time-interval in place: {freshman, sophomore, junior, and senior}. In this example the temporal ordering represents the 'contextual-dependency', and time-interval (in this case: at-least a year) indicates the 'temporal-dependency'.

Such requirements, if not handled properly, can bring a change into the underlying database that is premature, undesired, and not consistent with the timeline. We present the solution where such dependencies are taken care of through cellular constraints (to be defined in Chapter-4). Through these constraints, we capture the past, current and future possible states of the events in advance, so that the state of a change (through inserts and updates) is permitted only if it is in accordance with temporal requirements dictated by the business logic. In other words, there are real world applications that require a timeline to be imposed in order to dictate when the values should undergo a change, or which value should appear when. Hence there is a need to address these issues where the subsequent inserts or updates are contextual, and also where their appearance is constrained by a time window, and blocked otherwise. In a traditional relational database system, the above considerations are not taken into account. As a result, the time factor has the potential to induce inaccuracy by bringing in the corrupt data, as such invasions are left unfiltered or unnoticed by the existing set of constraints.

To complement the existing relational database constraints, we propose cellular constraints that act as filtering mechanisms, where the incoming event is first confronted with the time to check its timeliness (temporal depedency), and then be judged in the presence of its own previous instance(s) to test the contextual dependency, if any. Temporal dependency ensures that the data undergo a change at the appropriate time; contextual dependency ascertains that the degree or extent to which the data undergo a change at that given time span is as expected. Overall, if both of these conditions are satisfied then the incoming data have been time authorized.

The database system provides a data-definition language (DDL) to specify the schema, and a data-manipulation language (DML) to query and update the data. Just like the existing integrity constraints, our constraints are specified using DDL's 'create' statement. Our work currently focuses on scrutinizing the 'insert and update' DML statements. We will see in coming sections, how our makes the data manipulation process time. Our research applies only to applications where the temporal aspect of data matters (is of significance).

1.1 Goal

The objective of this research is threefold: 1) enumerate temporal requirements unaddressed by the relational data model; 2) establish a framework using constraints that captures all the possibilities enlisted in previous step; 3) and simulate the specification and implementation of proposed constraints at the metadata level, to show how relational data model can address temporal requirements without any external script or manual intervention.

1.2 Motivational Example

Consider a scenario from university where the students' assignment collection and grading is managed online. Following table schema associates student and assignment information:

Grading (<u>student-id, Assignment-id, course-id</u>, due-date, submission-date, points)

Apart from the student and assignment ids, there are four additional columns in the *Grading* table. The *course-id* identifies the course managed by an instructor; *due-date* specifies the date of when an assignment is due; *submission-date* captures the date on when a student had turned in the assignment, and the *points* column stores the grade a student (student-id) had received for a particular assignment (*assignment-id*).

One of the requirements is that if the assignment is not submitted by the original due date, then the students can still earn a maximum of 50% points by submitting it before the extended due date (typically a week after the original due date). In addition, the students cannot resubmit the same assignment more than two times.

In the above example, the two time oriented requirements that cannot be handled by the existing set of relational constraints are:

i. As the time progresses, the range of possible values (domain) allowed in the *points* column changes. For example, if the initial range of points before the due date expires is say 0-100, then for the upcoming week after the due date it is 0-50, and finally after the extended due date it is 0-0. This cannot be handled by the existing CHECK constraint.

ii. The requirement allows students to submit the same assignment twice. At the metadata level, as the student-id along with course-id and assignment-id form the primary key, there is no chance the student can resubmit the same assignment for the same course since that action will result in duplication of the primary-key in the table. There is no way to restrict the cardinality between student-id and assignment-id to some exact number (in this case, it is two).

Although to some extent application code (scripts) can take care of such temporal requirements, there is no unified approach or framework that can capture all the variations possible in temporal requirements. Such requirements are currently handled at programming level, where the programming logic is maintained manually, demanding extra and external effort. This research attempts the specification and enforcement of temporal requirements in an application independent manner; i.e., the logic that was outside the database is now been moved inside the database schema.

The motivating example suggests the need for an in depth examination of integrity constraints at a more detailed or granular level, i.e., at the level of individual cells (value in a row of a certain column). This is a new revelation, since all the existing 6 relational constraints analyze the data, either at the table or column level only.

Another motivation for this research was the limitations of temporal databases. Although temporal database could handle or avoid data's version overlapping, we ran into lot of temporal requirements that required manual intervention or support from scripts and application-logic. We looked out for such time oriented requirements and gathered as many as we could before we started with the abstraction process. We were looking for scenarios where the values could be prohibited or constrained using time.

Based on these requirements, we started out by identifying types of values with respect to time. For example, the value could be seasonal (repeating periodically), sequential, eternal, expiring, incremental, prohibitory (non-existing), successor, predecessor, compulsory, optional, repetitive, random/arbitrary, functional/derivative, aggregated, isolated etc. Based on these categories, we initially identified fifty-one ways to prohibit the incoming value, which was faulty with respect to time. We saw similar patterns repeating in most of these categories. We isolated these patterns from the values to build constraints for each of these patterns. We eventually ended up with nine constraints, as to be detailed in chapter 4.

1.3 Dissertation Outline

Chapter 1 discussed the need, objective, and motivation behind our proposed work. The balance of this dissertation is structured as follows: Chapter 2 enumerates the shortcomings in relational data model's existing data integrity constraints. This chapter also enlists new terminologies and contributions made in this dissertation, followed by the overview of our work. In Chapter 3 we explain the temporal and contextual dependencies that come into play when correctness of data with respect to time has to be justified. Chapter 4 begins by defining each cellular constraint, describes the ways of monitoring events, and concludes by explaining the association or mapping that exists between cellular constraints and proposed dependencies. Chapter 5 shows how each of the proposed constraints has its own style of dealing with the incoming data (event), as each of them associates the incoming event with the previous events in the window in a unique manner. This chapter depicts the scenarios, explaining how the faulty incoming events are identified and prohibited by the cellular constraints. Chapter 6 explains the mechanism for constructing, locating, and populating the event tables that store the previous events. This chapter also discusses the importance of prebuilt (mapping and linkage) tables, along with CET and PET algorithms. All these tables and algorithms are required to scrutinize the incoming event in the presence of past events so as to make a call on whether to accept or prohibit the incoming event. Chapter 7 talks about how all of the proposed cellular constraints are specified (and hence imposed) using SQL’s ‘create table’ statement. In Chapter 8 we talk about techniques of how a time window is initiated and terminated on the timeline. Chapter 9 showcases the results; chapter 10 discusses the related work; and chapter 11 concludes along with the limitations and future work.

Chapter 2

Extending Relational Data Model to Incorporate Temporal Aspects

In Chapter 1, we discussed the motivation for this research and what we aim to accomplish. In this chapter, we first discuss the temporal aspects that are implicitly being addressed in relational data model design, and then the temporal aspects that are missing. Later we list the unaddressed temporal requirements in relational data model, and propose new terminologies to address these requirements. We end the chapter by presenting overview of this research.

2.1 Some shortcomings of existing Relational Data Model

We examine the fact that to what extent the existing relational constraints deal or are associated indirectly with the time factor.

2.1.1. Relational data model's Values and Constraints from Time Perspective

We can divide the values in relational data model into three categories: current, past, and future values. The past values are the ones that already exist in the database. Current values are in the presently executing inserts or update statements. The future values belong to a pool from which the values can be selected for future insertions or updates. Traditional constraints can also be classified or seen from time perspective, based upon the type of values they are dealing with. For example, a 'foreign key' constraint ensures that the incoming value is a part of the already existing values (past values) located in a certain column of a database table. A 'primary key' or 'unique' constraint ensures that the incoming value is not found among the already existing values in a certain column of the database table. Although by specifying the data-type, domain of valid values for a column is established, ‘check’ constraint can further be imposed to narrow down this domain to a smaller subset. The ‘check’ constraint scrutinizes whether or not the incoming value is a part of the past or future pool of values by imposing a condition that mainly consist of relational and logical operators. The Not-Null constraint ensures that, except for the ‘NULL’ value, all values (from the domain) can act as the present value. Here, the past, or future pool of values are not considered.

As discussed above, although relational data model to some extent deals with time factor implicitly by categorizing the data into past, present and future pool of values, it fails to address the following two dependencies which come into play when time factor is explicitly associated with data.

2.1.2. Unaddressed Contextual and Temporal Dependencies

In relational data model, although there exist such implicit classification (past, present, and future) of ‘values’ and ‘constraints’ with respect to time, it only make sense when all the values across a column are taken into account. For instance, the incoming value has to be compared with *all* the existing values in a column on which the primary key, foreign key or unique constraint has been applied. We narrow this concept further and apply it at cell level granularity. We propose comparing the new or incoming value to the previously appeared values in the cell under consideration. By doing so, we capture the contextual dependency among the values appearing in a cell over certain span of a time window. The ‘cell’ denotes the intersection or junction of the row and column in consideration.

A record passes its life cycle through inserts, updates, and deletes, but there is no way to make the inserts and updates happen at a particular point in time of that life cycle, and avert them otherwise. In addition, there are certain applications that demand a limit on the number of insert/update statements, or that a certain gap be left between the executions of consecutive inserts/updates. We impose a timeline concept, where the inserts and updates are constrained by time, and hence are prohibited until an appropriate time has been reached. In short, time dictates the appearance of new values, and indirectly controls the disappearance of prior data values from that cell. Our work classifies timestamped data as either timely or untimely. Untimely or ill-timed data are those which do not show up on time and are either premature or delayed in their appearance.

Overall, we make the temporal anomaly explicit where inserts and updates take place at the wrong time when they were unsought for; in addition we also bring to notice the contextual anomaly arisen when the incoming insert/update statements fail to take the past value(s) of the cell into account.

2.1.3. Unsettled Temporal Requirements

Temporal requirements are not enforced by the relational database constraints, but by application logic. Our efforts lie in abstracting such application dependent logic and to theorize them by bringing them within the reach of database's schema. Some of generic unaddressed temporal requirements are as follows:

For a cell in database table, time can specify ordering or sequencing within a certain set of values; time can dictate a compulsory appearance of a value over a regular interval; time can set a threshold on the recurrence of the same value again and again; time can prohibit the occurrence of a value for a certain time period; time can command whether a given value remains eternal, or comes with an expiry so that it can be updated further; time can limit the number of inserts/updates allowed on the cells of a column; time can specify the interval between the consecutive inserts/updates; time can change the range or domain of possible valid values as it progresses; and time can accept the incoming value only if it satisfies certain conditions with previous values in consideration. We explain below how the existing data integrity can be supplemented using proposed cell integrity to handle temporal and contextual dependencies in the relational data model.

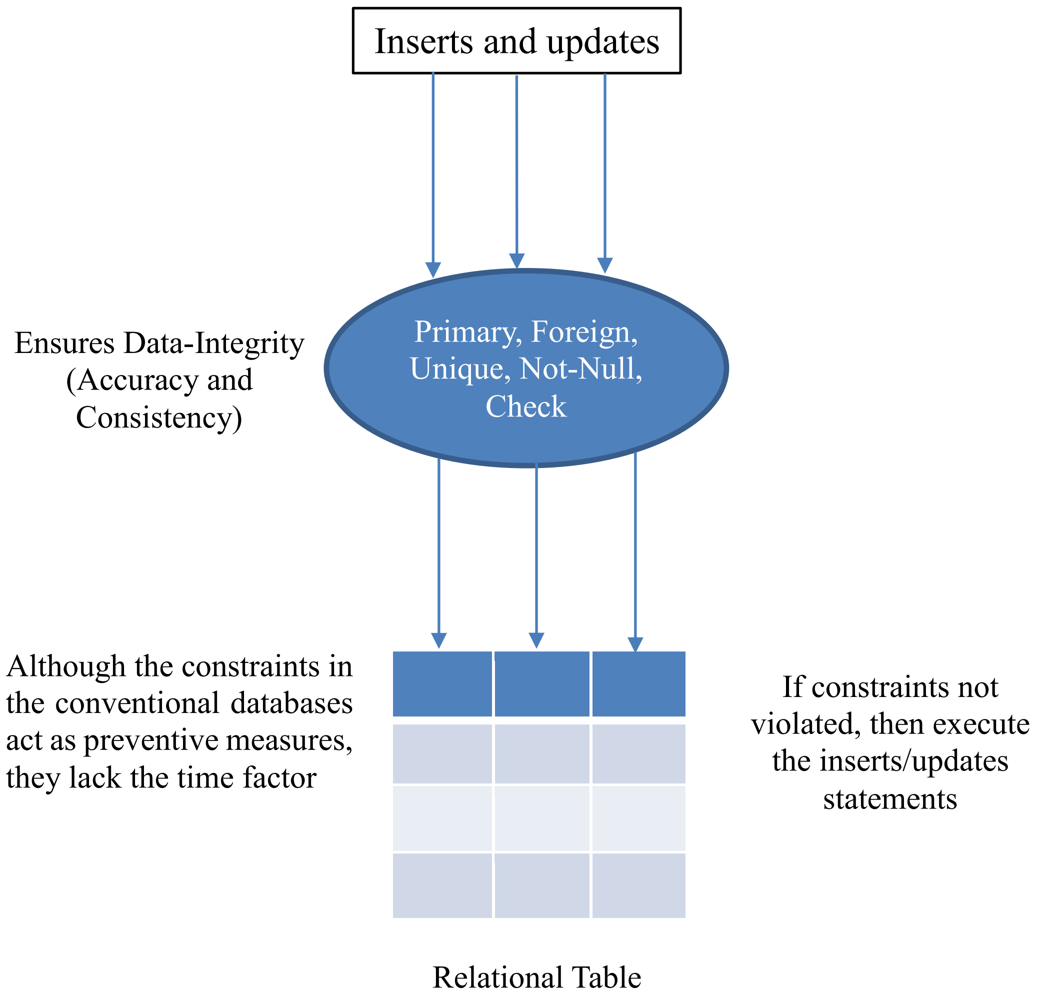

Inserts and updates
Ensures Data-Integrity (Accuracy and Consistency)
Primary, Foreign, Unique, Not-Null, Check
Although the constraints in the conventional databases act as preventive measures, they lack the time factor
If constraints not violated, then execute the inserts/updates statements
Relational Table

Figure 2.1 Data Integrity Workflow

The five constraints in relational database cover three types of data integrity: domain, entity, and referential. Although these constraints take care of consistency and accuracy of data, they do not address validity of data. We use the following two diagrams to illustrate. Figure 2.1 displays generic steps involved while executing inserts and updates in traditional RDBMS; whereas, Figure 2.2 displays the steps involved while executing insert and update statements when our proposed work is in place.

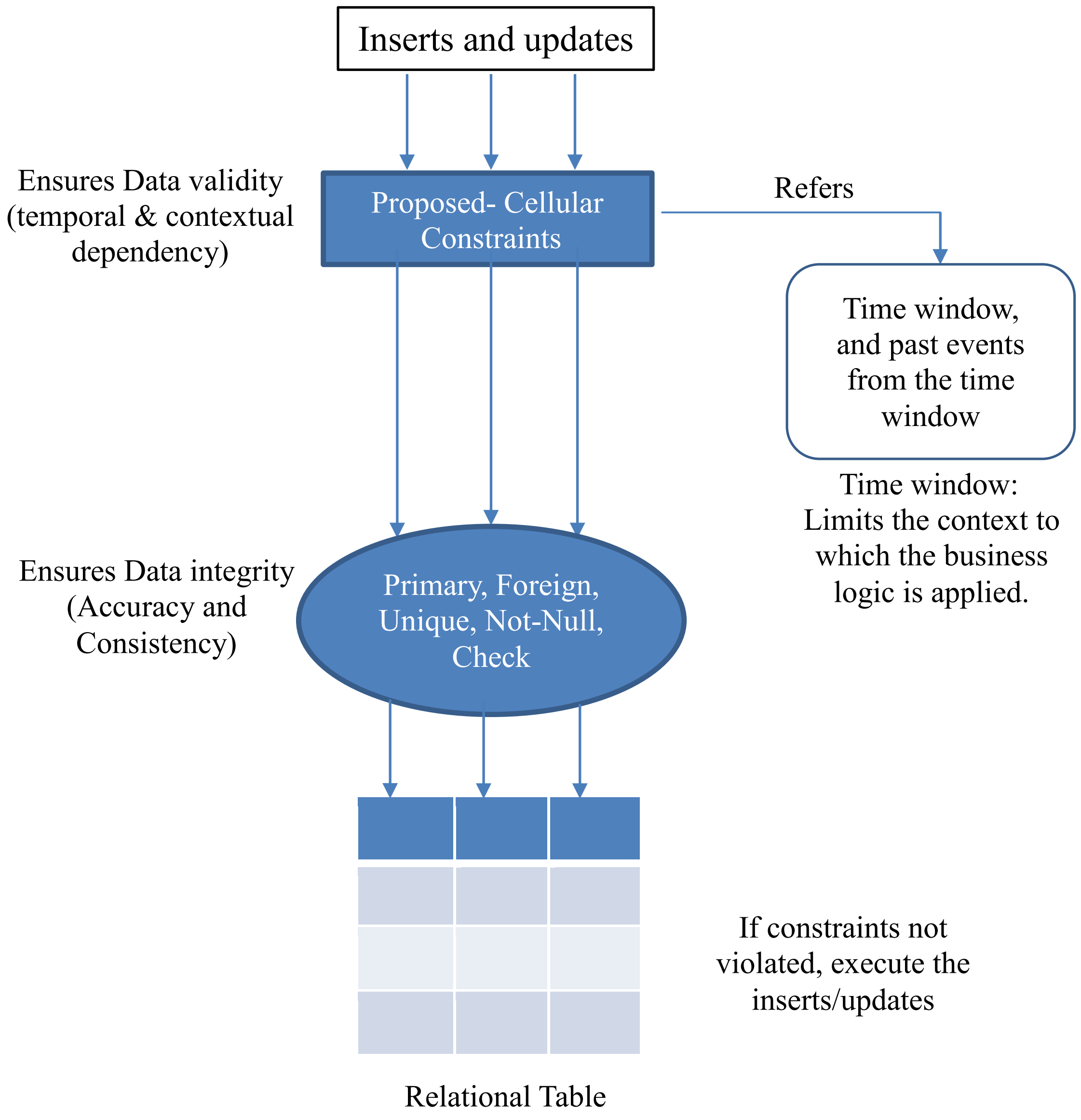

Inserts and updates
Ensures Data validity (temporal & contextual dependency)
Proposed- Cellular Constraints
Refers
Time window, and past events from the time window
Time window: Limits the context to which the business logic is applied.
Ensures Data integrity (Accuracy and Consistency)
Primary, Foreign, Unique, Not-Null, Check
If constraints not violated, execute the inserts/updates
Relational Table

Figure 2.2 Data Validity Workflow

Figure-2.2 shows how the proposed cellular constraints supplement the existing constraints so that the incoming data has to first satisfy the more granular (cell level) integrity to advance towards the traditional data integrity constraints. We have designed a parser to simulate the filtering mechanism depicted in Figure 2.2, but it can always be made part of the database engine itself.

The inserts/updates are blocked by our proposed parser if:

i. the incoming event is untimely; one cannot go back in time or ahead of time to make changes to database, as it violates the temporal dependency.
ii. incoming event is unfit with respect to past events on the timeline. The past events determine whether or not the change brought in by the incoming event is drastic or uncalled for. If yes, then such change violates the contextual dependency established by the past events, and the incoming event is therefore prohibited from occurring.

2.2 Proposed Terminologies to handle Temporal Requirements

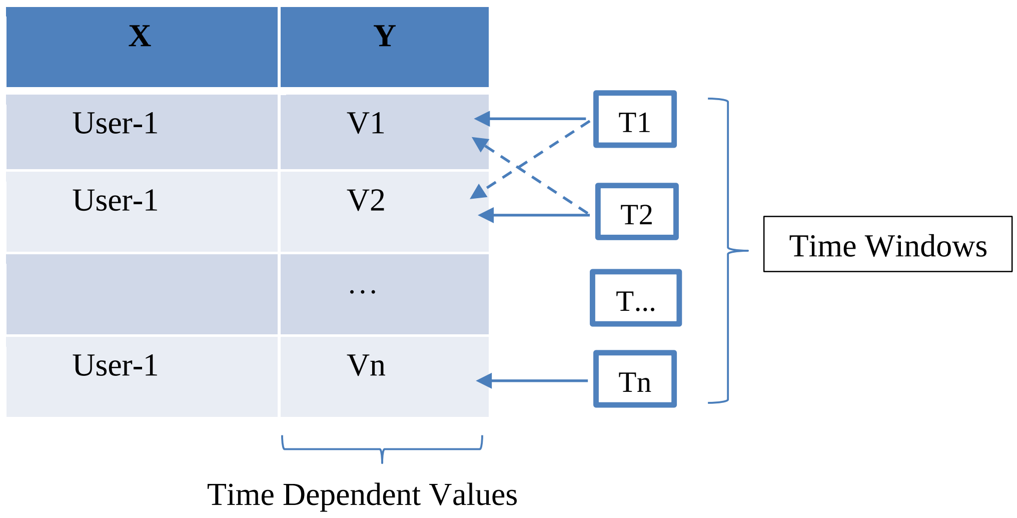
We propose and elucidate the concepts that give due consideration to temporal aspect of the data.

2.2.1 Temporal-dependency

Column ‘Y’ is temporally dependent on column ‘X’, if for a particular object in column ‘X’ the corresponding values appearing in column ‘Y’ (during inserts or updates) are time dependent. In other words, values in column ‘Y’ can appear only when the appropriate time window is open, and blocked otherwise. Table 2.1 depicts this scenario, where the dotted-line indicates the prohibition of values, since these attempts were either after or before the right time window had appeared. Here we assume that T1 is the correct time window for V1, and similarly T2 is for V2.

Furthermore in Table 2.1, X is a non-primary column and hence every time a new value of Y appears, a new row is *inserted* into the table. But when column-X is a primary key, then the same cell associated with row ‘X’ and column ‘Y’ gets *updated* over and over again. This situation is depicted in Figure 3.1 and will be detailed in chapter 3.

Table 2.1 Temporal Dependency



X	Y
User-1	V1
User-1	V2
	…
User-1	Vn

2.2.2 Contextual Dependency

Column ‘Y’ is contextually dependent on column ‘X’, if for a particular object in column ‘X’ the corresponding values appearing in column ‘Y’ (during inserts or updates) are dependent on one or more previously appearing values, as shown in Table 2.2. The incoming value might not be necessarily dependent only on one previous value, as is depicted in Table 2.2 (to be discussed later in Chapter-4).

Table 2.2 Contextual Dependency

X	Y
User-1	V1
User-1	V2
	…
User-1	Vn

Context Dependent Values

Inserts and updates are prohibited if they violate temporal and/or contextual dependency. Contextual dependency deals with events in the time window, whereas temporal dependency focuses on the occurrence of time window.

2.2.3 Contextual Anomaly

A contextual anomaly occurs when the new event to be inserted/updated is not in congruence or harmony with the past events in time window. That is, here past events consider the change made by an incoming event as inappropriate, because of the reasons to be detailed in chapter 4.

2.2.4 Temporal Anomaly

A temporal Anomaly occurs when the new information is inserted/updated before the appropriate time has arrived, or after the appropriate time has elapsed. We will see later how we eradicate temporal and contextual anomalies but at the cost of redundancy, as there is a need to construct event tables to store past events' information.

2.2.5 Constraint Level

In relational databases, we can broadly classify constraints as table-level and column-level. The column-level constraint is applied on a single column, and gets evaluated when any changes to that column are made. In other words, it imposes a filtering mechanism on that column to restrict the values that it can store. The table-level constraint spans multiple columns, and gets evaluated when a change to any one of those columns is made. Our proposed constraints are neither column nor table level but are cell level constraints, since they get evaluated when any changes to a cell of the column are made.

2.2.6 Integrity Level

When applied to columns, domain, entity, and referential integrity specify what values should appear and what values should not appear in those columns. Cell integrity, on the other hand, decides which values should appear and not appear in the cells of a column at a given point in time. That is, when cellular constraints are imposed on a certain column, they are unconcerned with the values appearing across that column; rather, based on the previous values appearing in the cell, they decide whether the cell can hold the incoming value or not.

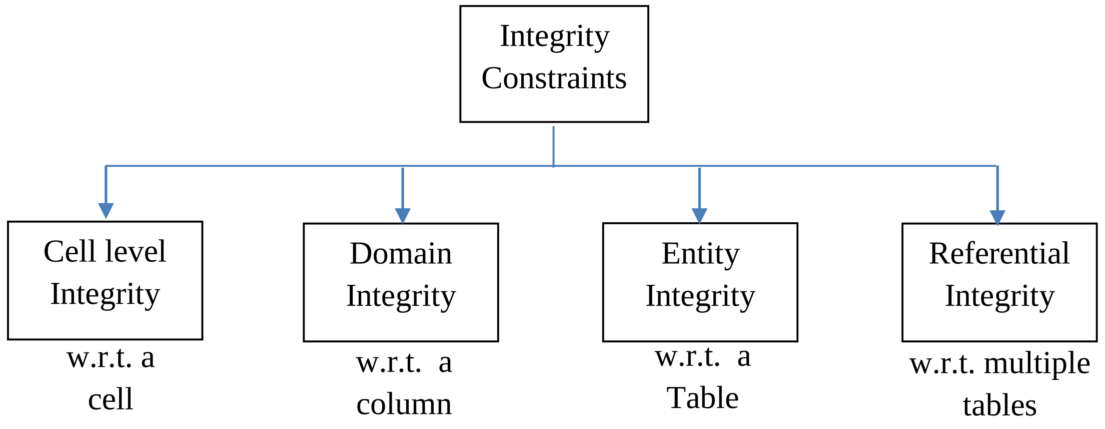


Figure 2.3 Types of Integrity

As our proposed constraints work at the cell level, we call them cellular or cell level integrity constraints. Integrity classification is presented in Figure 2.3, where ‘w.r.t.’ stands for: ‘with respect to’.

2.2.7 Events

Event reflects an action that creates a datum or value at a certain point in time. Often, events are generated by users or objects. In relational table, certain attribute (column) of an object, captures or stores this event’s value. Different attributes of the object are associated with different type of events. Two characteristics associated with any event are: the *time* at which the event occurs and the *value* generated by that event. We propose nine cellular constraints: window, interval, aggregate, sequence, threshold, range, compulsory, penalty, and frequency to monitor and capture the intricacies (relationships) between the events. Also, we assume that an event generates only one value at a given point in time.

2.2.8 Time Window

The relational table is composed of two dimensions: rows and columns. But we explore an implicit dimension called ‘time’ that helps us preserve previous instances of each cell in the column on which the proposed cellular constraints are imposed. The extent to which we should be traversing the time in backward direction (to consider previous events), is limited or determined by a ‘time window’. A ‘time window’ is used to mark the boundary that will be of relevance to the incoming event; in other words, it is used to limit the context in which the incoming event will be judged. A ‘time window’ acts as a default constraint and needs to be applied before any one of the other eight cellular constraints are imposed.

Without the ‘time window’ constraint the other constraints cannot stand alone. It can be compared to the ‘data-type’, which by default exists on each column of the relational database table. Without having the ‘data-type’ on a column, the other 6 integrity constraints cannot be applied. Similarly, the ‘time window’ constraint represents the foundation on which zero or more cellular constraints can be imposed.

2.2.9 Changing (Dynamic) Metadata

Current database schema or metadata (data about the data) has no ability to change as the time progresses. We will see later how the business logic requires dynamic-metadata to capture the changes demanded by temporal requirements. In temporal requirements, domain of values for a column, changes as the time progresses. We call this behavior ‘dynamic metadata’, which has the ability to make the same data valid or obsolete as the time progresses. We enable the specification and implementation of such dynamic metadata with the help of proposed cellular constraints.

2.3 Contributions

Mainly, this research analyzes the effect of time on the insert and update Data Manipulation Language (DML) statements. Some of the other highlights of our research are:

- We offer a thorough and critical analysis on integrity constraints of the relational data model, and identify a new form of integrity at a more detailed granularity level.
- We propose cellular constraints to enable relational databases to handle temporal requirements.
- This research takes a timeline into account to showcase how the presence or insertion

of a value is dependent on time and is contextual to the previously appearing values on the timeline, or how the absence of a value on the timeline can affect subsequent values to come.

- Our approach addresses temporal requirements that demand limited attempts for data manipulation (inserts or updates), indirectly reflecting an exact cardinality number. In addition, proposed work is useful when the window for data manipulation is to be kept finite.
- Due to the proposed cellular constraints, execution of these insert and update DML statements become time restricted (monitored).
- We propose temporal and contextual dependencies, in addition to functional-dependencies in relational databases.

Overall, temporal requirements implicitly convey that a user or object (usually represented by primary key column) has only limited chances and limited opportunities to perform actions (which are recorded into one of the non-key attributes/columns), and that these actions are constrained by past actions. To address such temporal requirements, we propose cell integrity, and implement cellular constraints to achieve temporal and contextual dependency.

2.4 Overview of our Work

We used relational database design as foundation for our research, and then performed simulation to show how the proposed constraints can be integrated or made part of the existing database design. We use the create-DDL statement to specify our constraints (using comments though), and then ensure that DML statements (inserts/updates) obey the imposed cellular constraints. Furthermore, similar to existing relational data model constraints, the cellular constraints are specified at the metadata (schema) level. The high level architecture diagram of our work is presented in Figure 2.4.

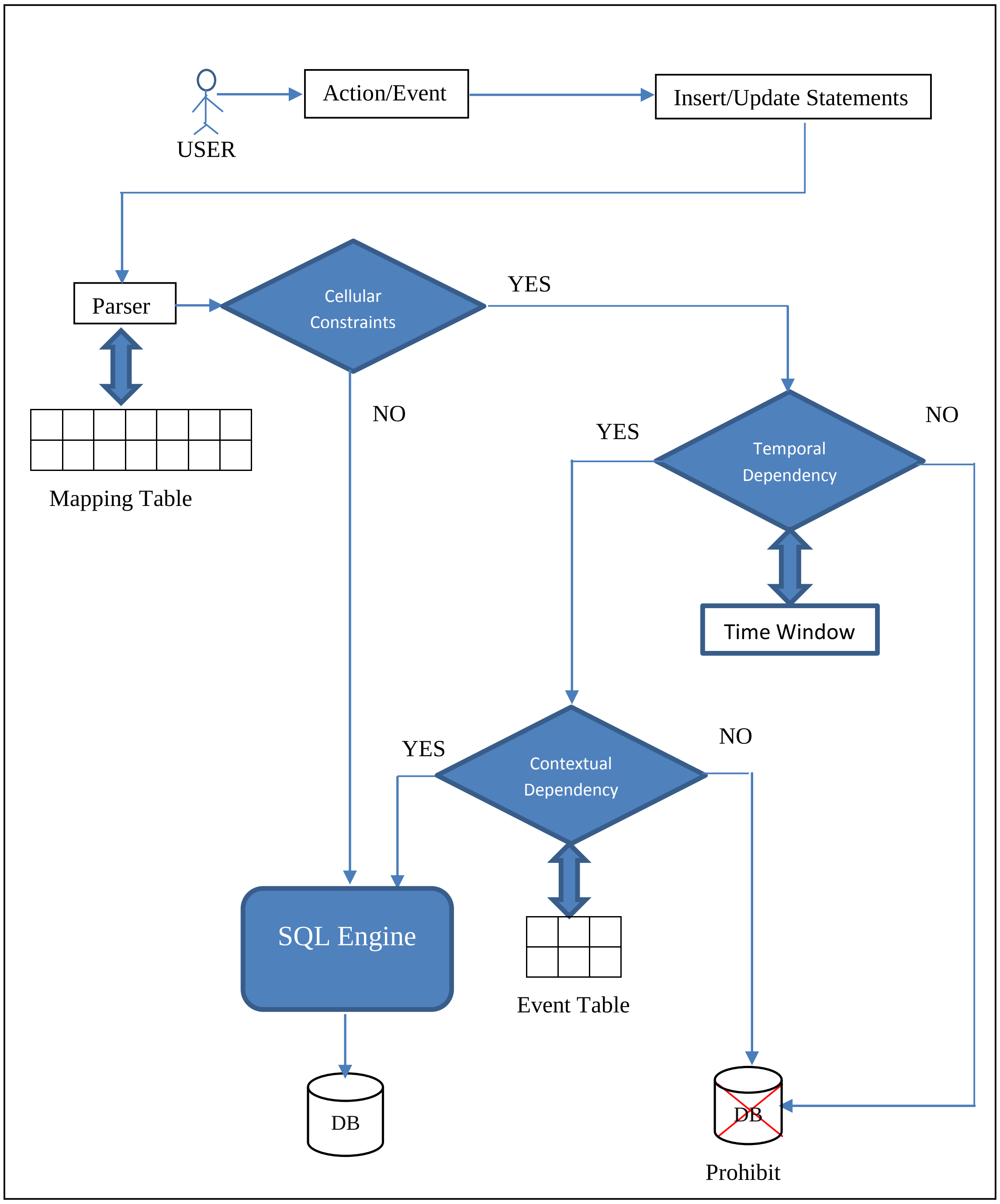


Figure 2.4 High-Level Architecture

In the context of data management, events can be represented as insert or update statements that are used to record the information associated with the user's/object's action, occurred at a certain point in time. Before executing the insert/update statement, our parser consults the metadata of the relational table involved. This metadata is stored in the mapping table (one of the prebuilt tables), as discussed in chapter 6. If there are any cellular constraints imposed on any one of the columns of the relational table, then the first step is to check the timeliness of the incoming event, and ascertain that the event is punctual, and not premature or delayed in its appearance. This check ensures the satisfaction of temporal dependency. In second step, the contextual dependency is checked by referring to the particular event table (to be discussed in chapter 6).

The required past actions/events of the user are then selected from the event table. Based on the type of cellular constraint imposed, the parser checks whether the contextual dependency between the incoming event and past events is satisfied or not. If not satisfied then the statement is prohibited, or else the insert/update statement is passed on to the SQL-engine for execution. If no cellular constraints are imposed, then all the above steps are bypassed, and the statement is directly forwarded to the SQL-engine for execution. Figure 2.4 shows the control flow of the proposed work. Later, we describe the syntax and set of keywords used for construction of the cellular constraints. We also explain how these constraints are specified in the 'create table' statement of SQL language. An outcome of our research is a program that understands the semantics of cellular constraints. We simulated our research work using the java programming language and MySQL database. Although our contribution lies more at conceptual level, the implementation part helped us test the feasibility of incorporating proposed work with existing set of integrity constraints.

Chapter 3

Temporal and Contextual Dependencies

In Chapter 2 we explained the need for this research by enumerating the shortcomings in existing integrity constraints of relational data model, and defined terms that will ensure temporal consistency of data in relational data model. In this chapter we discuss the two dependencies, contextual and temporal, arisen when the temporal aspects of data need to be preserved.

3.1 Temporal Dependency

Temporal dependency is implemented using time window, time interval, and time frame concepts. Below we define each one of them, and establish the relationship between them.

3.1.1 Time-Window (TW)

Time-window represents a slice on the timeline where the inserts and updates are permissible (i.e., where database can be subjected to change). It also indirectly indicates the timespan in which the business application is interested.

3.1.2 Time-interval (TI)

Time-interval represents forbidden timeslots where the change to the database is prohibited. Except for these timeslots, the event can appear anywhere else on the timeline. Such time interval period can appear either within a time window or between the time windows.

3.1.3 Time Frame (TF)

The time taken to execute an insert/update statement on a database. That is, every incoming event takes some marginal amount of time (greater than zero) to get recorded into a database. We term this span of time required to store an event into the database as the time frame.

Our research makes it possible to specify the above aspects of time (window, interval, and frame) using ‘window’ constraint. Overall, the temporal dependency can be ensured by configuring this ‘window’ attribute (‘interval’ attribute is derived, and ‘frame attribute is implicit), so as to ensure that the events are appearing on time and rejected otherwise.

3.2 Contextual Dependency

The contextual dependency represents a relationship between an incoming event and the past events. We have enumerated 8 types of relationships that can exist between the incoming event and past events. More on this is detailed in chapter 4. To understand contextual dependency, consider Figure 3.1.

When an insert or update statement is executed, first it has to be checked whether the event (statement) is occurring at the right time or not. If not, the event is banned from further consideration. If the event shows up on time, then it implies that the temporal dependency has been satisfied. Next, the statement has to satisfy the contextual dependency, before it is forwarded to the traditional integrity constraints.

‘Insert’ statement in Figure 3.1 is trying to insert a tuple in relational table ‘T1’, where the specified value for column1 is ‘Val-1’, for column2 is ‘Val-2’, and for column3 is ‘Val-3’. Let us focus on the cell in row#2 of column#3. Cellular constraint has been imposed on column#3. The cell of row#2, column#3 presently has value 'CV'. As we go back in time using the event table (to be discussed later), we see that there were three values that appeared previously in this particular cell.

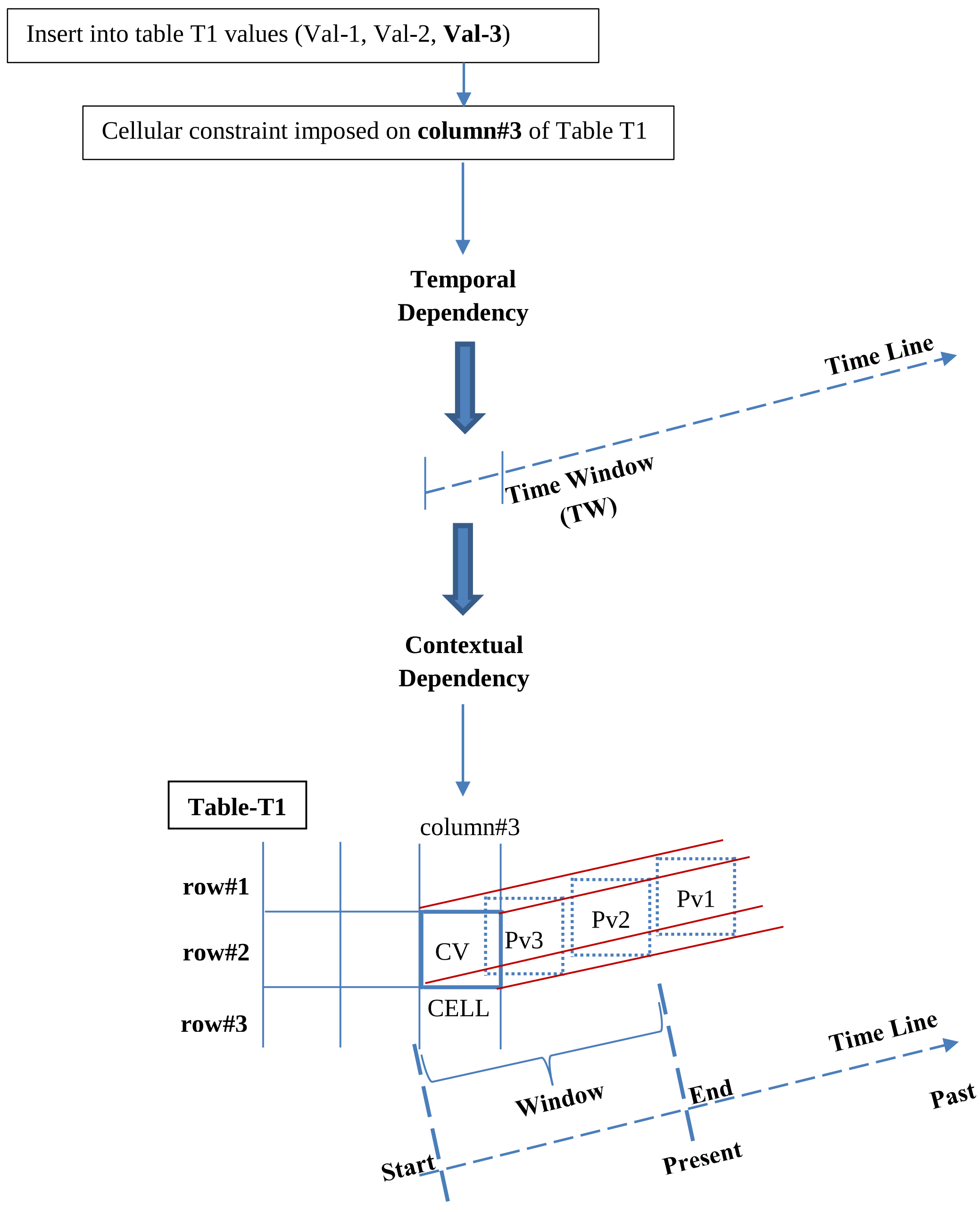


Figure 3.1 Granular Integrity (Cell level Granularity)

Past events are stored in the event table, and window constraint is used to select only the required past events from this event table. Let us say, out of those three, the window captures two of them (we will see later how the window-size represents a slice on the timeline in which the business-application is interested in). The window has a start and an end point; we will discuss in Chapter-8 how these points are explicitly specified using 'window' constraint.

Before the incoming value (in this case: Val-3) is inserted into the cell, it has to be checked in the context of previous values captured by the time-window. If it satisfies the particular contextual dependency then it is accepted as the current value, and the existing current value is pushed back as a previous value onto the timeline. If the contextual dependency is violated then the incoming statement is banned from execution. In the next chapter we will see the kind of contextual dependency each cellular constraint has to offer. In this example, the end point of the window coincides with the present time on the timeline, which might not be necessarily be the case.

To summarize, incoming event first needs to satisfy the temporal dependency; later, to testify contextual dependency, the event is judged in the presence of previous events captured by the time window. If the contextual dependency is not violated either, then the event is passed onto the database engine for execution.

Chapter 4

Constraints Definitions and Classifications

In Chapter 3, we explained the temporal and contextual dependencies that are useful in determining correctness of time centric data. In this chapter, we narrow down our focus to the proposed cellular constraints. We begin by defining each constraint and then based on their characteristics, we classify them either as base or composite.

To begin with, we need these cellular constraints whenever we want to ensure that events should be initiated or resumed by time, and not by someone's undisciplined, whimsical or prone to error nature. In section 4.1, we will discuss how the correctness of time centric data is defined, and then in section 4.2 we will see how this correctness can be achieved through the monitoring of event parameters.

4.1 Correctness (or validity) of Events

The definition of which values are legitimate and which are not changes over the time. Our constraints capture this dependency between time and the varying correctness of a value.

The correctness of value can be determined either:

i. based on the time when event occurs (temporal dependency), or
ii. by the past events in the time window (contextual dependency).

We propose 9 constraints to capture the correctness of a value with respect to time and past events.

Past events are of two types:

i. failed attempts (inserts and updates) that could not change the database
ii. successful attempts (inserts and updates) that changed the database

We will see later how the ‘penalty’ constraint relies on failed attempts whereas the rest of the constraints take into account successful attempts. Table 4.1 describes the functionality associated with each one of the proposed cellular constraints.

Table 4.1 Cellular Constraints

<u>Cellular Constraints</u>	<u>Mechanism</u>
Window	Timespan where the execution (inserts/updates) of events on the timeline is permissible.
Interval	Timespan where the execution of events is prohibited. It indirectly specifies the gap or time that needs to elapse between two events or groups of events.
Sequence	This constraint ensures that the incoming event is in right order relative to the recent past event in the time window.
Threshold	This constraint specifies how many times the same event can appear or repeat in a time window.
Range	The domain of valid values changes as the time progresses. This constraint enables such specification of varied domains required for different timeslots.
Aggregate	This constraint makes sure that the incoming event satisfies the imposed equation composed of: arithmetic and relational operator, and a constant. (More on this is section 5.9, and 7.2)

Penalty	Maximum number of wrong events permitted in the time window.
Frequency	Maximum number of correct events possible in the time window
Compulsory	Minimum number of correct events required to keep the time window active or sliding.

4.2 Monitoring of Events

Controlling of events is nothing but to scrutinize their parameters, which includes: time, value, and cardinality. We propose prime or base constraints to monitor a single parameter, and composite constraints to monitor more than one parameter. Base constraints are: window, compulsory and frequency. Composite constraints are: aggregate, sequence, penalty, threshold, interval, and range. Table 4.2 shows the constraints and the parameters they control. As three parameters are involved, there are a total of eight combinations possible for controlling of the events.

Table 4.2 Possible ways to monitor an Incoming Event

Parameters Monitored of an Incoming Event w.r.t. past events	Cell Level Constraints
Time	Window
Value	Aggregate and Sequence
Cardinality	Compulsory and Frequency

Cardinality and Value	Threshold
Cardinality and Time	Interval
Value and Time	Range
Cardinality, Value and Time	Penalty
None	RDBMS (based on relational model)

Now the cardinality parameter (which represents number of past events taken into account from the time window), can either consider zero, one, some, or all events from the time window. Based on this reasoning and the table above, we get the following sixteen combinations. Table 4.3 shows the cardinality associated with each of the cellular constraint.

Note that out of these sixteen possibilities we propose constraints only for nine combinations, as at this point in time we did not come across temporal requirements that could capture the leftover possibilities. This remains one of our future works where we find temporal requirements and hence propose new cellular constraints to capture the rest of the unaddressed combinations.

Table 4.3 Constraints based on Incoming and Past Events

	Parameters Monitored [of an Incoming event with respect to past event(s)]	
Constraints	Cardinality (Number of Past Events from the window)	Value/Time

None	None/Zero	
	One	
Compulsory	Some	None
Frequency	All	
Relational data model	None/Zero	
Sequence	One	
Threshold	Some	Value
Aggregate	All	
Window	None/Zero	
	One	
Interval	Some	Time
	All	
Range	None/Zero	
	One	
	Some	Value and Time
Penalty	All	

Note that the semantics of composite constraints cannot just be deduced or inferred using the combination of two or more base constraints, since such combination will altogether lead to different semantics than what the composite constraints stand for. For example, just because the interval constraint deals with cardinality and time parameters, it does not follow that it is a combination of window and compulsory, or window and frequency constraints.

4.3 Constraints and the Dependencies

Here we see which constraint is associated with which dependency.

4.3.1 Constraint for Temporal dependency

We propose one constraint to capture the temporal dependency discussed in section 3.1. This is the ‘window’ constraint, which focuses upon the occurrence time of the event. It checks whether the incoming event appears in the correct timespan of the time window or not. If not, then the event is termed as ill timed (premature or delayed) in its appearance. If the event is verified by the ‘window’ constraint then the event is considered on time. The parser then goes through other cellular constraints (if imposed) to check whether the incoming event is contextual or in harmony with the previous events on the timeline or not.

4.3.2 Constraints for Contextual Dependency

We propose eight constraints to capture the contextual dependency discussed in section 3.2. Contextual dependency captures the relationship between the incoming event and past events. This dependency can either be based on cardinalities of past events, on values of past events, on the occurrence time of past events, or on a combination of these.

These eight cellular constraints linked with contextual dependency take past events from the time window into account. The extent to which these constraints traverse backward in time to collect past events is limited by the span of a time window. These constraints ensure contextual dependency by checking whether the previous events can hinder the execution of an incoming event or not. If the contextual dependency is violated then it implies that the past events do not accord or agree either with the incoming event's time, value, cardinality, or combination of these.

4.4 Mathematical Representation

The relationship among time window, incoming and previous events can be conveniently captured in a formula. We consider the Time Window (TW), and the Previous Event (PE) in a cell of the row-R and column-C as independent variables; and New Event (NE) that might appear next in that cell, as the dependent variable. As a result, we have:

$$\mathbf{NE \rightarrow TW\ [\ n\ (\ PE_{v,t})\ ]} \qquad \text{Equation – (4.1)}$$

The above equation or dependency states that the new or incoming event's occurrence is dependent upon the value and/or time of 'n' previous events from the time window (TW). We consider the cases where the new or incoming event is dependent either on zero, one, subset (some), or all events that previously appeared in that cell over a given time window. This variation is specified using the cardinality variable 'n'. The value of 'n' can vary between: 0 and all. We focus on 4 categories, where n=0, n=1, 0< n<all (some), and n = all. The value of 'n' for the constraints defined above are:

n=0: for window and range constraint; n=1: for sequence constraint

0 < n<all: for threshold, compulsory, and interval constraints

n=all: for aggregate, penalty, and frequency constraints.

In the examples to come in chapter 5, 7 and 9, it will be clear on why the constraints have particular cardinality value (specified by 'n'). The equation indirectly conveys that the existing value of a cell is subjected to change as the time changes. Through these proposed constraints the transition phase of a cell is well captured in advance.

Chapter 5

Identifying and Prohibiting Faulty Events

In chapter 4 we defined and classified the constraints based on their characteristics. Now in this chapter we bring the concepts described in chapter 3 and 4 together, to explain how each constraint associates the incoming event with the past events (in the window) in unique manner, to see whether the temporal, contextual dependencies are preserved or violated. Below we provide diagrammatic presentation for each of the requirements and, indirectly, the requirements discussed in section 2.1.3. We use following notation in the figures to come:

1) V – Any new value
2) CV – Correct Value
3) IV – Incorrect Value
4) PV – Previous value
5) NS – No Show (No Value)
6) TW: Time Window
7) TF: Time Frame
8) TI: Time Interval

In the examples below, we assume that cellular constraint in discussion is imposed on certain column of the relational table, and that we are focusing on one of the cells in that column in which the incoming event (insert/update statement) is trying to bring in a new value. Also, for the sake of simplicity we ignore additional details like window size, constraint specifications, or why and how the time window has certain number of past events.

5.1 Window Constraint

Time window is specified using start point and end point on the timeline. This constraint makes sure that the event is prohibited if it appears outside the time window. Figure 5.1 depicts this scenario, where the crossed lines indicate prohibition of incoming events, as they appear outside the time window.

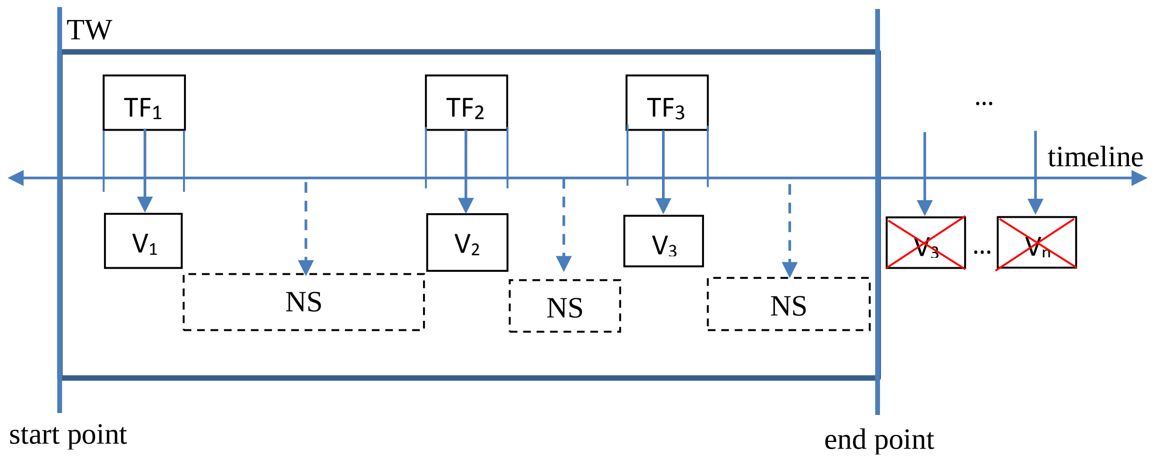


Figure 5.1 Window Constraint Violation

5.2 Interval Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'interval' constraint is being imposed, where within a time window of a year only one event can appear in an interval of 6 months. Before the insert/update statement is executed, 'interval' constraint refers to the previous events captured by the time window. In Figure 5.2, a second attempt was made while the first interval was still in progress, and hence blocked. Similarly, when the last interval was in session, the second and third attempts were blocked for the same reason. That is, if any additional inserts/updates are attempted before the interval period is over, then those events are prohibited.

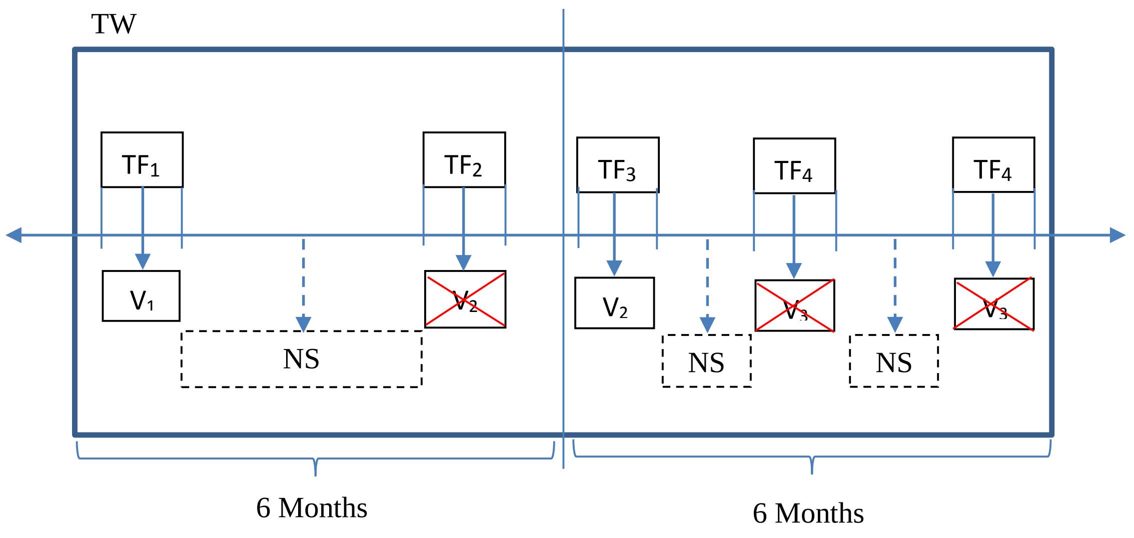


Figure 5.2 Interval Constraint Violation

5.3 Frequency Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'frequency' constraint is being imposed, where no more than three events may be stored in the time window. Before the insert/update statement is executed, 'Frequency' constraint refers to the previous events captured by the time window. But as depicted in Figure 5.3, the time-window already has 3 events from the past; the forth incoming event is therefore prohibited as it violates the 'frequency' constraint.

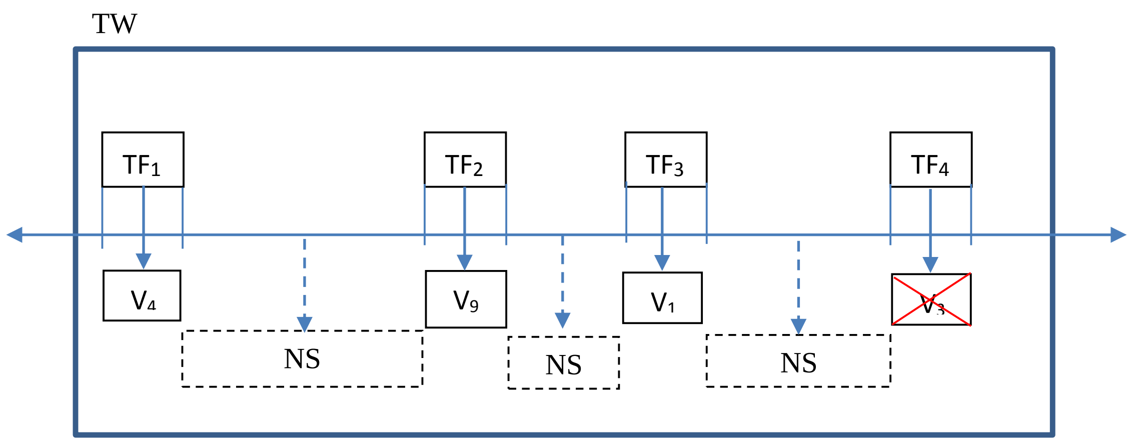


Figure 5.3 Frequency Constraint Violation

The 4th incoming value is forbidden because the window's frequency is set to 3.

5.4 Threshold Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'threshold' constraint is being imposed, where the same event cannot repeat more than twice in a time window. Before the insert/update statement is executed, 'threshold' constraint refers to the previous events captured by the time window. As shown in Figure 5.4, window of the cell under consideration has three past events, and the 'threshold' constraint is analyzing the incoming event in presence of these previous ones. As the threshold was set to two, the third appearance of 'V_1' is prohibited because the value had already been repeated twice in the time window.

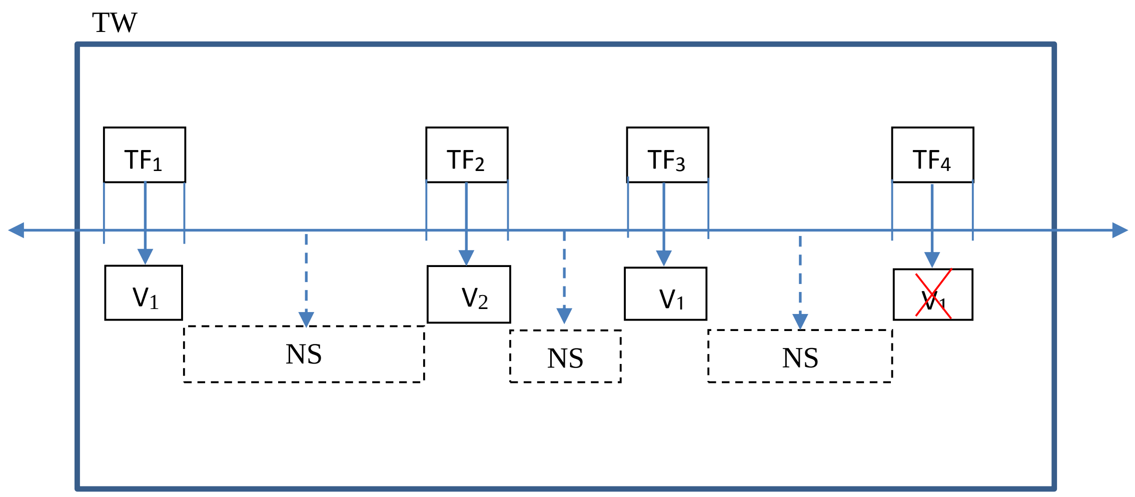


Figure 5.4 Threshold Constraint Violation

Figure 5.4 shows that the forth incoming event is banned as it violates the threshold constraint.

5.5 Sequence Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'sequence' constraint is being imposed, where within a time window the events should appear in specified order: {V_1, V_2, V_3, and V_4}. Before the insert/update statement is executed, 'sequence' constraint refers to the previous events captured by the time window. This constraint ensures that the values are accepted only if they appear in the required order (which indirectly reflects the business requirement). If the sequence is not followed, the values are prohibited. Figure 5.5 shows that if any other value appears apart from expected one in the queue, then that value is prohibited.

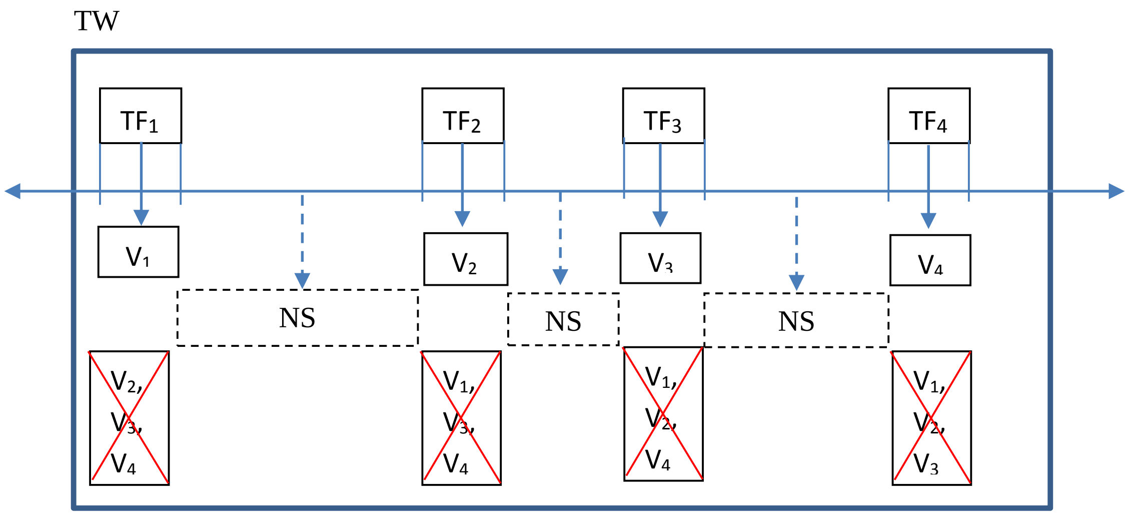


Figure 5.5 Sequence Constraint Violation

5.6 Compulsory Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'compulsory' constraint is being imposed, where within a time window at-least one event should have already been occurred. Before the insert/update statement is executed, 'compulsory' constraint refers to the previous events captured by the time window.

In this constraint, if the minimum inserts/updates are not attempted in the current window, then all the subsequent windows are frozen and attempts are then prohibited. Figure 5.6 depicts the scenario where the count for the minimum number of inserts/updates to appear in the window of a cell is set to one. As shown, the moment a window is short of minimum required events, all the future windows are sealed, and the incoming event is therefore discarded.

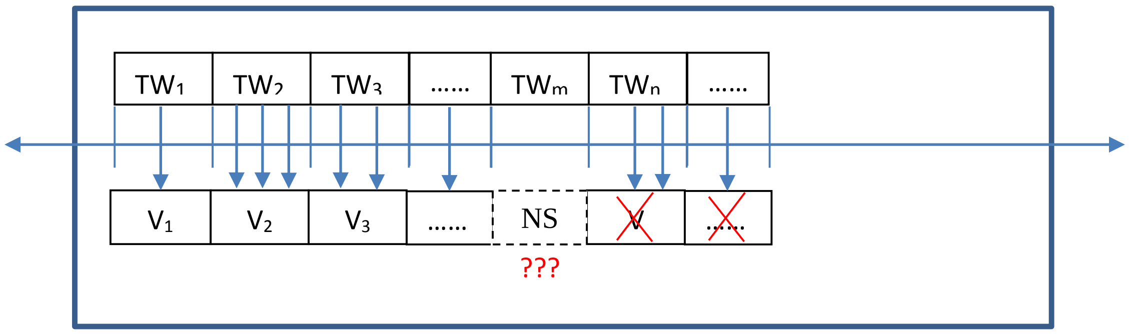


Figure 5.6 Compulsory Constraint Violation

5.7 Range Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'range' constraint is being imposed, where within a time window the set of valid values keeps changing over time. Before the insert/update statement is executed, 'range' constraint identifies the domain of values valid at that point in time. In order to get inserted/updated, the incoming value has to be a subset of this set identified by range constraint, else it's prohibited. In Figure 5.7, the value V_2 is not a subset of the set-2, and hence is banned from insertion in the cell of a column on which this range constraint is imposed.

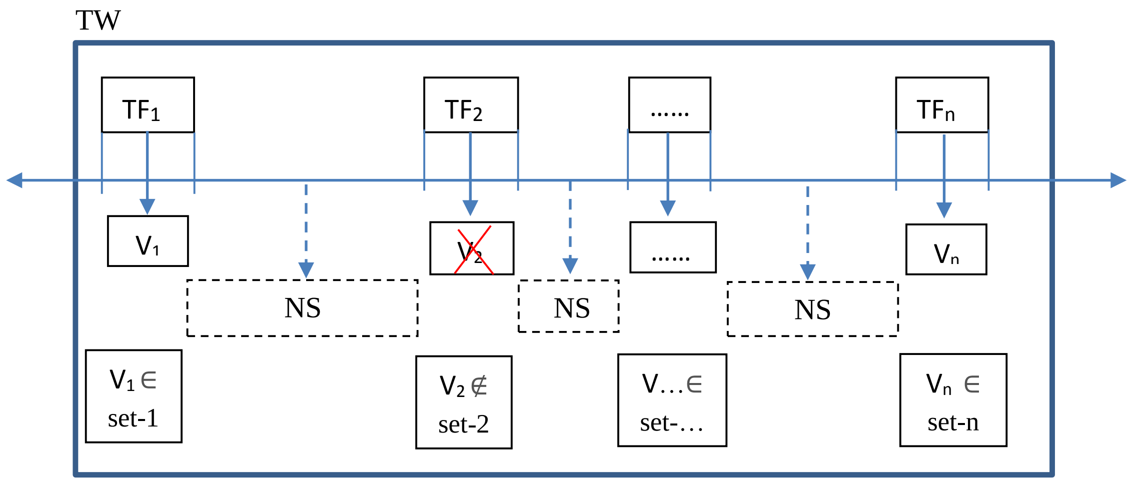


Figure 5.7 Range Constraint Violation

5.8 Penalty Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the 'penalty' constraint is being imposed, where within a time window the correct value is expected in certain number of attempts.

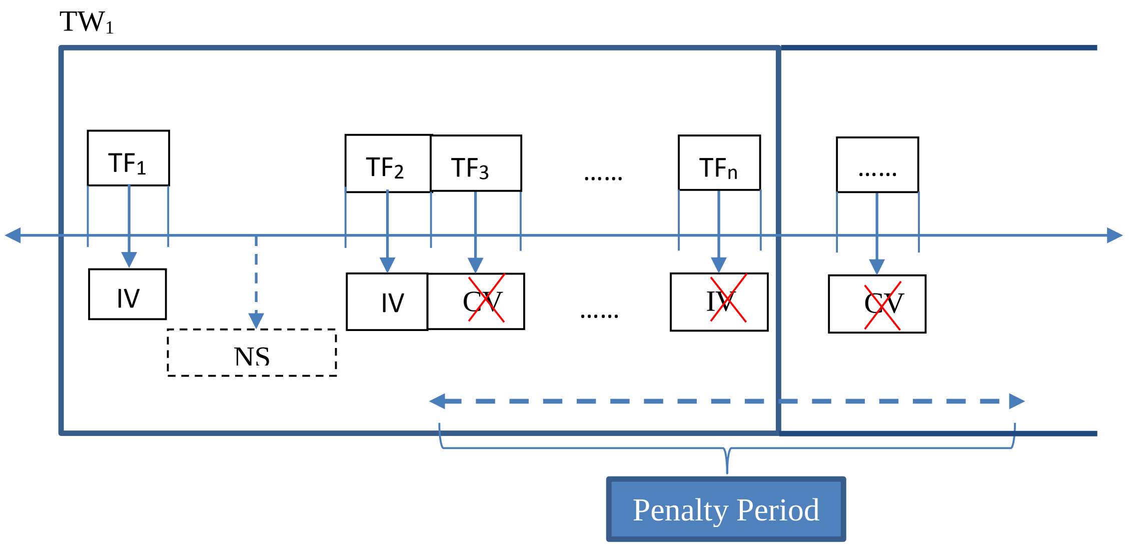


Figure 5.8 Penalty Constraint Violation

Before the insert/update statement is executed, ‘penalty’ constraint refers to the previously appeared faulty events captured by the time window. If the correct value (CV) does not show up in certain number of attempts (let us say that the limit is set to 2), then a penalty is imposed, where even the correct incoming value gets discarded. As shown in figure 5.8, after first two incorrect attempts, all attempts are blocked until the penalty period expires. Depending upon the span, the penalty period can certainly extend into the subsequent window(s).

5.9 Aggregate Constraint

Suppose an incoming event attempts to insert (or update) a value into the cell of a column on which the ‘aggregate’ constraint is being imposed, where the incoming event has to satisfy the imposed arithmetic equation. Before the insert/update statement is executed, ‘aggregate’ constraint refers to the previous events captured by the time window.

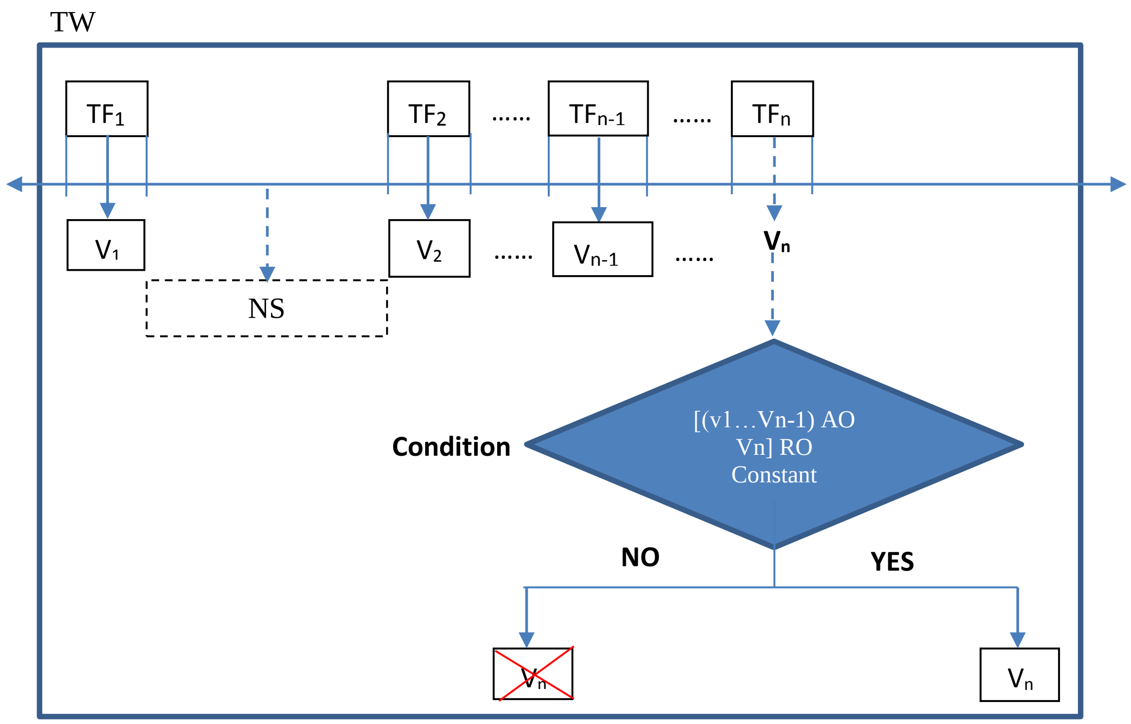


Figure 5.9 Aggregate Constraint Violation

For an aggregate constraint, the incoming value is associated with previous events' values using an Arithmetic Operator (AO), and then using the Relational Operator (RO) the result is compared to the specified constant. If the condition is satisfied, then the value is eligible for insertion, else prohibited. This concept is depicted in Figure 5.9. We will see in Chapter 7 and Chapter 9 how one or more cellular constraints can simultaneously be imposed on the same column; i.e., the combination of these constraints on a particular column is possible.

5.10 Constraints Complementing Semantically

Let us take a look at the pairs of constraints that complement each other semantically.

5.10.1 Window and interval constraint

Window constraint specifies the timespan when the events should occur; whereas, interval constraint specifies the gap that should be left between two events or group of events.

5.10.2 Compulsory and Frequency constraint

Compulsory constraint represents the minimum number of events that should occur in the time window; whereas, frequency constraints sets the maximum number of events that should occur in the time window. Here, the value of the event is not inspected.

5.10.3 Penalty and Threshold constraint

Penalty constraint dictates maximum wrong events that can appear in a time window; whereas, threshold constraint specifies a limit on the maximum number of times a correct event can appear in a time window. Here both the value and the cardinality of events are inspected. We do not consider cases where the minimum number of wrong events need to be monitored, as we did not come across any such temporal requirement.

5.10.4 Sequence and Aggregate constraint

Sequence constraint considers only one event from the time window; whereas aggregate constraint takes into account all the events from the time window.

5.10.5 Range and Relational data model's check constraint

The domain of valid values specified using the 'check' constraint from relational data model remains the same forever; whereas, our proposed range constraint makes sure that the domain of legitimate values changes as time progresses. Range constraint enables specification of unique domain for each of the time periods in a window.

Chapter 6

Creating and Populating the Event Table

In Chapter-5 we saw how cellular constraints deal with the incoming event by taking past events from the window into account. We looked at the scenarios where the proposed constraints identified and then prohibited the faulty incoming events as they violated the temporal and/or contextual dependency. In this chapter we discuss how through event tables the previous events of a cell are stored, located, and retrieved (made accessible) so that the incoming event can be judged with these earlier events on the timeline.

The necessity for event table arises because relational table stores only the current data; but our cellular constraints require the history of previously occurring events as well. Event tables are required to store the previous 'events' that are generated by users or objects. Event tables record information that is sufficient to implement cellular constraints. A separate event table is constructed for each of the imposed cellular constraints.

Event table records event's 'time and value' information along with the user or object-id responsible for its generation. The event table is composed of three columns: (a) The 'user-id' column that identifies the user whose action resulted in an event; (b) the 'event-value' column, which records the value of the events; (c) and the 'event-time' column, which stores the occurrence time of the event. There exist one to many relationships between users and events, with respect to time. That is, a user can generate many events, but only one at a given time. Multiple events can appear at the same time, provided that they are associated with different users. We therefore create a composite key that captures these mappings. The schema for an event-table is: (*user/object-id, event-time*, event-value).

The event table records information only for the columns on which cellular constraints are being imposed. Separate event-table gets constructed for each column on which the cellular constraint is imposed. Note that our research affects only the ‘create’ DDL statement, as rest of the SQL statements remain pretty much unaffected. The last three lines of the following ‘create table’ statement reflect the way cellular constraints can be imposed through SQL’s ‘create’ DDL statement. As of now, we specify the cellular constraints using the opening and closing comments (/*…*/). This part needs to be commented out so that SQL engine does not throw an error or exception. Our parser later segregates this commented section to understand the semantics of cellular constraints, and to implement and bring into effect the specified cellular constraints. Below, we will first go through the ‘CET-Algorithm’ that is invoked when ‘create table’ statement is executed, and then discuss the ‘PET-Algorithm’, which is referenced when insert/update statements are processed.

6.1. Create Event Table (CET) Algorithm

Let us consider the following example, where we create a relational table called ‘RT’. We will later see how this statement implicitly results in the creation of an event table (for the column3 of table ‘RT’ on which some cellular constraint is being imposed).

```
Create table RT
(
    column1  Data-type,
    column2  Data-type,
    column3  Data-type,
             …
```

```
     column-x Data-Type,
     Primary key (column1, column2)
     Foreign key (column-x) references table-y(column-z)
     /*
     object (column2),
     cellular constraint (column3) : [specification],
     window(column3) = [2 hours]        */
);
```

When our parser analyzes the above ‘create table’ statement, it creates the following event table, since in the comment section (/*…*/) it encounters a column in table RT on which one of the cellular constraints is being imposed.

```
 Create table RTcolumn3cellularconstraint
 (
     object id  Data-type,  //string
     Event-value  Data-type,     //string
     Event-time  System-time,  // datetime
     Primary-key (object-id, Event-time)
); //event-table for column3 of RT
```

The name of the event table is constructed by concatenating the table name with the column name and the name of cellular constraint that is been imposed upon it. In that way, the parser knows which event tables to refer to when an insert or update is being executed over a certain relational table with cellular-constraints.

For instance, consider the example from the introduction section. One of the temporal requirements in universities is that the values appearing in the graduation-year column of a student’s record has to follow this temporal ordering, with minimum time-interval in place: {freshman, sophomore, junior, and senior}. The minimal interval is of 1 year between the successive inserts/updates, and student should complete the program within six years. The create table statement for this requirement is as follows:

```
Create table progress_report (
     student_id varchar(10),
     graduation_year varchar(10),
     gpa double,
     foreign key (student_id) references student_info(id)
     /*
     object(student_id),
     sequence(graduation_year)   :   [Freshman,   Sophomore,
     Junior, Senior],
     interval(graduation_year) :[1, 1 year]
     window(graduation_year) : [first-event, +6 years] */
);
```

When the parser goes through the commented section, it notices the imposition of cellular-constraints, and therefore create the following event tables.

```
Create table progress_reportsequencegraduation_year (
     student_id varchar(50),
     graduation_year  varchar(50),
     Event-time  datetime,
    Primary-key (student_id, Event-time)
);
```

The ‘progress_reportintervalgraduation_year’ also has the same ‘create table’ statement as above, except for the table name. In addition, no event tables are created for ‘window’ and ‘range’ constraints, as they do not deal with past-events. Examples in Chapter-9 will illustrate these scenarios.

In addition to creating the above event tables, the parser updates the ‘mapping-table’, composed of seven columns. Data recorded in ‘mapping table’ after execution of the create-table statement for ‘progress_report’ table, is displayed in Table-6.1.

Table 6.1 Mapping Table

User/ object-id	*Relational -table*	*Constraint -type*	*Constrained -column*	*Constraint -values*	*Window - size*	*Event-table (Primary Key)*
student_id	progress_ report	sequence	graduation_ year	{F, S, Jn, Sn} //specification	6 years	progress_reportsequencegraduation_year
…	…	…	…	…	…	…

Primary-key of the mapping table is ‘Event-table’ column. It implies that a particular kind of cellular constraint can be imposed only once on a given column.

Table 6.2 shows that, apart from the Mapping-table, we have a ‘linkage’ table that stores the primary-foreign key relationship information. After executing the create-table statement for ‘progress_report’ table, the ‘linkage’ table will have the following info:

Table 6.2 Linkage Table

Child-table	*Foreign-key*	*Parent-table*	*Primary-key*
progress_report	student_id	student_info	id

The Linkage table is required when there is involvement of any foreign object in establishing the window's start or end points (to be discussed in chapter 8). In section 6.2, we will see that how these Linkage and Mapping tables are utilized in the background by PET algorithm.

Above we saw how the event table was created; below we will discuss how the event-table will get populated using the PET-algorithm. Consider the following 'insert' statement to be executed on table 'progress_report'.

```
– insert into progress_report (student_id, graduation_year,
  gpa) values ('Sam', 'Junior',3.7);
     //this statement was executed on 2015-10-05, at 17:54:06.
```

Because there are cellular constraints imposed on one of the columns of progress_report table, before the execution of the insert statement on the 'progress_report' relational table begins, the following PET algorithm is referred by the parser to ensure that the imposed cellular constraints (and hence the temporal and contextual dependencies) are not violated. The above insert statement will be used as a supporting example as we walk through the steps of the algorithm listed below. As the algorithm progresses, one will notice that these four tables are referenced quite often: progress_report (relational), event-table(s), mapping, and linkage table.

6.2. Populate Event Table (PET) Algorithm

Input: Insert/update statement

Output: either prohibit the statement, or else forward it to the database engine

Procedure:

i. Parse the insert/update statement (representing an incoming event) to identify the relational table involved (‘progress_report’ in the example). Also note the system-time, when insert/update statement was requested for execution.

ii. Refer to the mapping table, and look for the table from above step (progress_report). If the table name does not exist go to step-ix; else note down the window-size.

iii. Calculate window’s start and end points, based on the window’s specification provided in the mapping table (chapter-8 details this process).

iv. If the event’s time of occurrence lies within time window’s range, then it implies that the temporal dependency is satisfied. If not, prohibit the incoming insert/update statement and exit the algorithm.

For each entry in the mapping table where column *relational-table*’s value equals to the one identified in the insert statement (‘proress_report’), loop through step v to viii.

v. Note the following information from the mapping table:

   a. The column that represents user/object-id. (‘student_id’ in the example)
   b. The column on which cellular constraint is being imposed. (‘graduation_year’ in the example)
   c. Type of the cellular constraint. (‘sequence’ in the example)
   d. Name of the event table. (‘progress_reportsequencegraduation_year’ in the example)

e. And the window's time-limit ('6 year' in the example)

vi. Parse the insert/update statement and note the values associated with the columns identified in step v-(a), and v-(b).

a. Column identified in step v-(a): represents user/object-id associated with the event (in the example, value 'Sam' is specified for this column).

b. Column identified in step v-(b): represents the incoming event's value (in the example, the value 'Junior' is specified for this column).

c. System time from step-i: represents the event's time of occurrence (this example's insert statement was executed at: 2015-10-05 17:54:06).

vii. Go to the event table identified in step v-(d) and select the rows associated only with the user/object-id in consideration ('Sam' in example), and also that which falls in the window's time-limit. ('6 years' in the example).

viii. The constraint identified in step v-(c) ('sequence' in the example), determines whether the incoming event preserves the contextual dependency with respect to previously appearing events (from the current window) or not. For example, in the given scenario, if the previous-value was not 'Sophomore', then the incoming 'Junior' value is prohibited since it does not fall in order, as specified or dictated by the 'Sequence' constraint. We will see in chapter-7 how the semantics of contextual dependency varies for each constraint.

a. If the incoming event's value does not satisfy imposed cellular constraint, then discard/prohibit the insert/update statement at hand, and exit the algorithm.

[Note: If the imposed constraint is 'penalty' constraint, then record the incoming event (even if faulty) into the respective event table.]

b. Repeat steps v-to-viii until no more rows in the mapping table, where value in the 'relational-table' column equals to the one identified in the insert statement ('proress_report'), are left dealing with. For example, there is still one more entry in Mapping-table for relational-table:= 'progress_report', where constraint_type:= 'interval'. This row is considered in the next iteration.

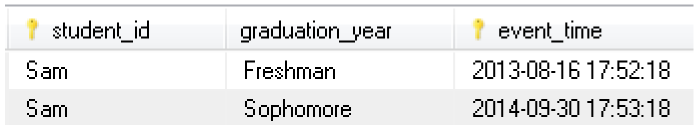
ix. Pass the statement through database engine, to see if the traditional data integrity constraints are satisfied or not. If yes, then the database engine executes the insert/update statement in question to update the relational table ('progress_report'); else prohibits the incoming event and exit the algorithm

x. Update all the involved event tables so that the current event becomes a past event ('progress_reportsequencegraduation_year', and 'progress_reportintervalgraduation_year' in the example).

xi. Commit

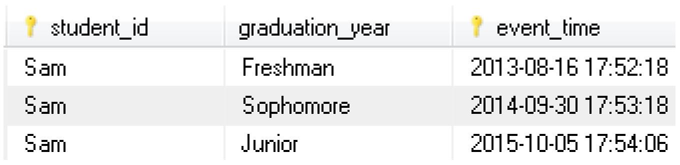
In the above example, if the insert statement was accepted, then both the event tables would end up having following entries for object 'Sam':

Table 6.3 Before acceptance of the event



student_id	graduation_year	event_time
Sam	Freshman	2013-08-16 17:52:18
Sam	Sophomore	2014-09-30 17:53:18

Table 6.4 After acceptance of the event



student_id	graduation_year	event_time
Sam	Freshman	2013-08-16 17:52:18
Sam	Sophomore	2014-09-30 17:53:18
Sam	Junior	2015-10-05 17:54:06

Overall, the CET-Algorithm is responsible for populating the mapping and linkage table, whereas, the PET-algorithm is responsible for populating event tables. To recap these algorithms, the entire background process can be summed up into the following steps:

i. When `CREATE TABLE` is executed:
   a. Mapping and Linkage table gets computed, and the required Event tables are created.

ii. When `INSERT OR UPDATE STATEMENT` is executed:
   a. Get the object and window-information from the mapping table.
   b. Find time window's start and end points.
      – Check whether the temporal-dependency is satisfied or not. If not quit; else proceed to next step. Repeat steps ii-(c) and ii-(d) for all the columns in the insert/update statement on which one or more cellular constraints are being imposed.
   c. For the object in consideration, get previous events falling in the time window, from the respective event table.
   d. Check for contextual dependency. If violated, block the incoming event and quit.
   e. Populate the relational as well as the involved event tables.

Chapter 7

Constraints Specifications

In Chapter-6 we discussed how the previous events of the object under consideration are selected; and, if the cellular-constraints are satisfied, how the incoming event is also recorded into the event tables and the relational table. In this chapter we explain the mechanism by which the constraints discussed so far are declared or instantiated using the `CREATE TABLE` statement of SQL. In this chapter we will take a closer look at the syntax, semantics, and illustrate an example for each of the proposed constraints, emphasizing the importance of each.

An incoming event can be contextually associated with old events, using a relational operator, an arithmetic operator, or a Boolean operator. While composing the syntax for each of these proposed constraints, we relied on arithmetic, relational, and Boolean operators. Below, we enlist syntax for all the constraints, and then begin addressing each of these constraints in detail.

Table 7.1 Constraints' Syntax

Declaration
`Window (column-name) : [start-point, end-point]`
`Aggregate (column-name): [arithmetic-operator, relational-operator, Constant]`
`Compulsory (column-name): [#events]`

`Interval (column-name) : [#events, timespan]`
`Frequency (column-name) : [#events]`
`Penalty (column-name): [column with correct-values, relational-operator, #attempts, penalty period]`
`Range (column-name) : [{(range, timespan) pairs}]`
`Sequence (column-name) : [Value-1, Value-2, …, Value-n]`
`Threshold (column-name) : [(list, count) pairs]` //implicitly uses Boolean: AND, OR

7.1. Window Constraint

7.1.1. Semantics

In the window constraint, the correctness of the value is dependent only on time; none of the previously appeared values are taken into account. All of the proposed constraints are time oriented, and there is a need to limit the infinite timeline so that the constraints can focus on a certain segment of it (area of interest) instead. The window constraint acts as a default constraint for the rest of the constraints. This window constraint limits the context of previous events in which the incoming event will be judged. In addition, this constraint also ensures that the incoming event is timely, and not premature or delayed in its appearance. We will also see why the default presence of this constraint becomes necessary and how rest of the constraints are implicitly dependent on this constraint.

7.1.2. Syntax

- `Window (column-name) : [start-point, end-point]`

  Here, the time-limit is specified using start and end points, which will be discussed in the next chapter.
- Any incoming event appearing in the given column is accepted only if it falls within the range of specified time-limit.

7.1.3. Example-1

Temporal requirement: Posting grades within the week, following the finals.

- Schema: class_roster (student_id, course, grade, semester)
- It implies that values in the column 'grade' will be subjected to change only in the first week from the date the semester has ended. In the worst case when manual intervention is required, to avoid the stringency of these proposed cellular constraints, database administrator's assistance should be sought for. In some universities, if the grades are not posted within a week's deadline then a form has to be filled out manually by the instructor for any further grade changes.
- Imposing 'window' Cellular constraint:

```
create table class_roster (
student_id varchar(10),
course varchar(10),
grade varchar(2),
semester varchar(10),
foreign key (semester) references
semester_info(semester)
/*object(student_id),
window(grade) : [semester_info.enddate, +1 week]
*/ );
```

The create table statement for the foreign table "semester_info" is:

```
create table semester_info(
semester varchar(10),
course_withdrawal datetime,
begin_date datetime,
end_date datetime,
degree_application datetime,
primary key (semester)
);
```

7.1.4. Additional Example

Temporal requirement: Enrolling of courses should take place in first 3 weeks of the semester.

- Schema: Enrollment(student_id, course, status, semester)

- It implies that values in the column ‘status’ (‘dropped’, ‘waitlisted’, or ‘enrolled’) will be subjected to change only in the first 3 weeks of the semester.
- In this example, the specification of ‘window’cellular-constraint is:
  - window(status) : [semester_info.begindate, +3 weeks]

7.2. Aggregate Constraint

7.2.1. Semantics

In this constraint, the correctness of event is dependent on all of the previously appearing events in the time window. From the time-window, all of the previous events of the object in question are taken into account, in order to decide the execution of an occurring event. Here the collective effort of all of the past events from the window plays an important role in deciding the execution of incoming event.

7.2.2. Syntax

- `Aggregate (column-name): [arithmetic-operator, relational-operator, Constant]`
- This constraint combines all the previous events’ values with the incoming event’s value using the given arithmetic-operator. Then the result is compared to the declared constant using the specified relational-operator. If the condition holds then the incoming event is accepted, or else rejected.

7.2.3. Example-2

Temporal requirement:

On any given day, irrespective of the number of transactions, the total ATM withdrawal amount of a customer should not exceed $400.

- Schema: transactions(customer_id, branch, withdrawal_amount)
- For a given customer, on a given day, the sum in ‘withdrawal_amount’ column should not exceed $400 amount.
- Imposing ‘Aggregate’ Cellular constraint:

```
create table transactions (
customer_id varchar(10),
branch varchar(10),
withdrawal_amount double
/*
object(customer_id),
aggregate(withdrawal_amount) : [+, <=, 400],
window(withdrawal_amount)  :  [-24  hours,  current-
time()]
*/
);
```

Whenever a customer accesses the ATM to withdraw money, the events of the past 24 hours are taken into account. This information is retrieved from the Event-table, discussed earlier. As long as the incoming event’s value along with the past events’ values do not violate the condition, the withdrawal is successful. If the condition is violated, then the user action is denied. Based on the requirements, required arithmetic and relational operator should be chosen, along with the constant.

7.3. Compulsory Constraint

7.3.1. Semantics

The compulsory constraint demands that some minimum number of events should have had occurred in past window to keep the current window active. In other words, the incoming event is permitted if minimum number of events were appearing on a regular basis in the past.

7.3.2. Syntax

- `Compulsory (column-name): [#minimum-events]`

7.3.3. Example-3

Temporal requirement: Free bus-passes are valid only if they are used at least once a month.

- Schema: commuters(user_name, buspass_id, busstop_location, balance)
- Imposing ‘Compulsory’ Cellular constraint:

```
create table commuters (
user_name varchar(20),
buspass_id varchar(20),
busstop_location varchar(50),
balance double,
foreign key (buspass_id) references bus_pass(pass_id)
/* object(user_name),
compulsory(busstop_location) : [1],
window(busstop_location):[bus_pass.valid_from,+1
month] */
);
```

The create table statement for the foreign table “buss_pass” is:

```
create table bus_pass(
pass_id varchar(20),
valid_from datetime,
expires_on datetime,
primary key (pass_id) );
```

7.4. Interval Constraint

7.4.1. Semantics

The interval constraint conveys the rate at which the events or actions of an object should occur over a certain period of time window. It indirectly specifies the minimum gap required between the occurrence of two events or groups of events.

7.4.2. Syntax

- `Interval (column-name) : [#events, timespan]`

7.4.3. Example-4

Temporal requirement: An international applicant, during his/her Optional Practical Training (OPT) that lasts for 29 months, can attempt for H1 visa no more than 3 times, with a gap of year between applications.

- Schema: H1_applicants (applicant_id, decision)
- Imposing ‘Frequency’ Cellular constraint:

```
create table H1_applicants (
applicant_id varchar(10),
decision varchar(20)
/*
object(applicant_id),
frequency(decision) : [3],
interval(decision) :[1,1 year]
window(decision) : [first-event, +29 months]
*/
);
```

7.5. Frequency Constraint

7.5.1. Semantics

The frequency constraint limits the maximum number of inserts/updates possible within a span of the given time window.

7.5.2. Syntax

- `Frequency (column-name) : [#events]`

7.5.3. Example-5

Temporal requirement: The graduate record examination (GRE) can be attempted once every 21 days, and up to five times within a year.

- Schema: GRE_attempts(student_id, location, score)
- Imposing ‘Interval’ Cellular constraint:

```
Create table GRE_attempts (
student_id varchar(10),
location varchar(20),
score double
/*
object(student_id),
interval(score) :[1, 21 days]
frequency(score) : [5],
Window(score) : [-1 year, current-time()]
*/
);
```

7.5.4. Additional Example

Another example is where some of the insurance companies allow a customer to file only two claims a year on lost, stolen, or damaged products.

7.6. Penalty Constraint

7.6.1. Semantics

The penalty constraint ensures that in certain *consecutive* attempts if the incoming event does not bring the expected result (correct event) determined by the relational operator, then a penalty interval will be imposed in which the future events are then banned, even if they turn out to be the correct ones.

7.6.2. Syntax

- `Penalty (column-name): [column with correct-values, relational-operator, #max event-attempts, penalty period]`

7.6.3. Example-6

Temporal requirement: Consider an administrative and support office that manages the logs of user accounts. One of the temporal requirements is allowing only 5 successive wrong password attempts to a user in a span of 12 hours. After the 5th successive wrong attempt, user account will be blocked and a penalty of say 24 hours will be imposed on that account.

– Schema: signin_info(user_id, pass_word, last_loggedin)

– Imposing 'Penalty' Cellular constraint:

```
create table signin_info (
user_id varchar(20),
pass_word varchar(20),
last_loggedin datetime,
foreign key (user_id) references credentials(users)
/*
object(user_id),
penalty(pass_word) : [credentials.passphrase, =, 5,
24 hours],
window(pass_word) : [-12 hours, current-time()]
 */
```

```
          );
```

The create table statement for the foreign table “credentials” is:

```
            create table credentials(
            users varchar(20),
            passphrase varchar(20),
            primary key (users)
        );
```

The sign-in (last logged-in) event has to check the password against the already stored correct passwords. The credentials.passphrase column stores this information. If the attempts are exhausted, then the user has to wait until the penalty period expires. Also note that, if the user enters a valid password before the limit is reached (i.e., if the incoming event’s value matches with the existing value), then the count (representing number of wrong events attempted) is reset to zero. Basically, the occurrence of correct-event logically deletes all the previous wrong attempts from the event table.

Penalty constraint is the only constraint in which the failed events or attempts are considered and stored in the event table. The rest of the constraints focus on successful events (events which result in data change/storage).

7.7. Range Constraint

7.7.1. Semantics

For the range constraint, the occurring event of an object is independent of the previously occurring events in the window. Only the time factor controls the range of legitimate values. This constraint takes into account the time an event occurs and then evaluates the possible range of values that are deemed valid for inserts/updates. That is, here the set of legitimate values changes as the time window progresses.

7.7.2. Syntax

- `Range (column-name): [{(range, timespan) pairs}]`

7.7.3. Example-7

Temporal requirement:

A credit card limit increases over time. Suppose a bank starts with an initial limit of $300 for the first year, and then raises it to $2000 for the next year, and then to $5000 for another 2 years, and finally to $15000, where it remains constant as long as the card is in use.

– Schema: card_users(customer_id, expenditure_amount, shop, card_id);

– Imposing ‘Range’ Cellular constraint:

```
create table card_users (
customer_id varchar(20),
expenditure_amount double,
shop varchar(20),
card_id bigint,
foreign key (card_id) references cards(card_number)
/*
object(customer_id),
Range(expenditure_amount) : [("0-300", 0000-12-31
00:00:00), ("0-2000", 0002-12-31 00:00:00), ("0-
5000", 0004-12-31 00:00:00), ("0-15000", 0099-12-31
00:00:00)],
 window(expenditure_amount) :
[credit_cards.issue_date, current-time()]
*/
);
```

The create table statement for the foreign table “credit_cards” is:

```
create table cards(
card_number bigint,
issue_date datetime,
expiry_date datetime,
ccv varchar(3),
primary key (card_number)
);
```

This constraint is similar to CHECK constraint in the relational databases [Example: age column, where value should lie between say: (1, 125) years]. Irrespective of the previous occurrences, the range of values gets affected, just because some time has been elapsed. For example, if the customer has been using the credit-card for the last 3 years, then at the beginning of the fourth year, his/her credit limit will automatically increase to $5000. This constraint indirectly takes care of the business activities (promotion/demotion) that occur in a timely manner. Also, the range indirectly indicates promotional offers available for loyal customers. The range also implicitly specifies the card categories such as: “bronze, silver, gold, platinum etc.”, made available as the time window progresses.

7.7.4. Additional Example

One of the requirements for international-students is that they can work no more than 20-hours a week, except during the summer. The requirement demands that the value occurring in the ‘hours’ column should be checked. In SQL, we have the ‘CHECK’ constraint which ensures that the value to be inserted should be within a specified limit. But in this example we cannot use ‘CHECK’, as the constraint on values will vary based on the current time. That is, the values between range of ‘0-40’ can only be inserted when the summer semester is in progress; otherwise, the range will change to ‘0-20’.

7.8. Sequence Constraint

7.8.1. Semantics

With the sequence constraint, the occurring event of an object is dependent on (compared to) the most recent event in the window. From the current window only one value (n=1) is taken into account. If the event is in sequence to the previous event then it is termed as *correct event*. Over a period of specified time window, this constraint ensures that the events appear in the specified order.

7.8.2. Syntax

- `Sequence(column-name)=[Value-1, Value-2,…,Value-n];`

7.8.3. Example-8

Temporal requirement: The values appearing in the graduation-year column of a student record have to follow the sequence, with minimum time-interval in place: {Freshman (F), Sophomore (S), Junior (J), and Senior (Sn)}. Also, the student should take no more than 6 years to complete the undergraduate program.

– Schema: progress_report (student_id, graduation_year, gpa)
– Imposing ‘Sequence’ Cellular constraint:

```
Create table progress_report (
student_id varchar(10),
graduation_year varchar(10),
gpa double,
foreign key (student_id) references student_info(id)
/* object(student_id),
sequence(graduation_year)  :  [Freshman,  Sophomore,
Junior, Senior],
interval(graduation_year) :[1, 1 year]
Window(graduation_year) : [first-event, +6 years]
*/
);
```

The create table statement for the foreign table “student_info” is:

```
create table student_info (
id varchar(10),
FullName varchar(50),
Address varchar(100),

Phone varchar(10),

emailid varchar(50),

birthdate datetime,

primary key (id)

);
```

For the object in consideration, whenever a new insert is made into the ‘progress-report’ table, the incoming value is checked against the exsiting value of the ‘graduation-year’ column to make sure that they are aligned with each other on the timeline. This constraint helps prevent the bypass on a temporally dictated ladder of events.

If the schema is: Progress-report (<u>student-id,</u> year, GPA), where ‘student-id’ being the primary key, then instead of ‘insert’, the ‘update’ statement is scrutinized by the imposed Cellular Constraint.

7.8.4. Additional Examples

- The prerequisite courses should be enrolled before the main course.
- Designation for professors usually follows the sequence of assistant to associate and to full professor.

7.9. Threshold Constraint

7.9.1. Semantics

The threshold constraint specifies the number of times the same value (event) can reappear or repeat in a time window.

7.9.2. Syntax

- `Threshold (column-name) : [(list, count) pairs]`

Here the pairs are implicitly associated using the logical OR operator. Within a pair, there is a ‘list’ composed of values which are connected using the logical AND operator.

7.9.3. Example-9

Temporal requirement: An international student can have at most two grades below B-, and one F grade in his/her entire curriculum program.

Grades that lie below ‘B-’ are: C, C+, C-, D, D+, and D-. For an international student, the combination of appearance of any of these grades in totality, should not exceed the count of 2.

- Schema: transcripts(student_id, course_number, grade)
- Imposing ‘Threshold’ Cellular constraint:

```
Create table transcripts (
student_id varchar(10),
course_number varchar(10),
grade varchar(2)
/*
object(student_id),
Threshold(grade)  :  [("C|C+|C-|D|D+|D-",  2),  ("F",
1)],
Window (grade) = [-3 years, current-time()].
*/
);
```

7.9.4. Additional Examples

- PhD Students:
  - Cannot take more than 3 independent study courses (CIST-9980) in the entire PhD program

    Threshold (course-id) = [(‘CIST-9980’, 3), …].
- Masters students:
  - Cannot enroll for the same graduate level course more than 2 times
  - Cannot repeat the prerequisite courses
  - Are not eligible for PhD level courses.

The above examples show that the information of past and future events are hard coded for the Window and Range constraint (‘time’ information in the windows constraint, and ‘time’ and ‘value’ information in the range constraint). In no other constraints is the event’s attribute information being explicitly specified.

Chapter 8

Time-Window Specification

In Chapter-7 we discussed how the proposed constraints were declared using the SQL `CREATE TABLE` statement. Although we also saw how the window constraint's start and end points were specified in general, in this chapter we will explore all the possible ways to initiate the start and end points of a window. We also explain how the start and end points of a window can be the same across all the objects, or how each object can have its own unique start or end point on the timeline.

8.1. Initiating Start and End points:

A window's start or end points can be decided: statically, dynamically, relatively, or derivatively/dependently.

- Static points are specified using the exact/precise time on the timeline.
- Dynamic points indicate that the points are decided at run time, and are unknown until first event of the object is triggered or generated.
- Relative points are calculated with respect to the window size.
- Dependent or derived points are determined by foreign objects.

8.1.1. Absolute (Static) initiation

Absolute time is specified by providing exact (precise) time on the timeline. Note: Current time reflects the real time, which is represented by system's clock time. Unlike absolute time, the current time keeps changing with the system's time. Also, the time in MySQL is specified in this format: YYYY-MM-DD HH:MM:SS.

8.1.2. Relative initiation

The start time can be relative to the end time, or vice-a-versa. If the start-time is specified, the end time is calculated using:

- End-time = start-time + window size

If the end time is specified, then the start time is calculated using:

- Start-time = End-time – window size

8.1.3. Foreign Object based initiation

Sometimes the object might need a foreign object to determine the start or end time of its window or both. With a foreign object in a one-to-many mapping to local objects, the same starting or end point can be set for all the native objects. With a one-to-one mapping, different foreign objects can be used to set up different starting or end points for each local object. In the assignment submission example (also the motivational example from chapter 1) below, the window's start date is the same across all the objects. The start time is nothing but the due date of the assignment (foreign object) involved. The window's end time will differ for each object, since the end time is the time the student submitted the assignment.

8.1.3.1.Example-10: Assignment Submission

```
create table students (
student_id varchar(20),
assignment_id varchar(10),
points varchar(10),
submission_date datetime,
primary key (student_id, assignment_id),
foreign key (assignment_id) references
assignments(assignment_number)
/*
object(student_id),
frequency(submission_date) : [2],
window(submission_date) : [assignments.due_date, current-
time()]
Range(points) : [("0-75",0000-1-7 00:00:00),("0-50",0000-1-
14 00:00:00),("0-0",0000-1-21 00:00:00)],
window(points):[assignments.due_date,students.submission_date]
*/
);
```

Foreign Object

The create table statement for the foreign table “assignments” is:

```
create table assignments(
assignment_number varchar(10),
due_date datetime,
primary key (assignment_number));
```

Foreign Object Schema

Another example is of ‘card_users’, where the window’s start time differs across objects. Here it is the ‘issue_date’ of the foreign-object ‘credit_cards’ that controls window’s start points for ‘card_users’. Overall, even the foreign object has the ability to initiate and/or terminate a window across one or all objects.

Whenever a foreign object holds widow’s start and/or end point then the user has to specify values for foreign keys in the insert/update statement. The reason being that the parser can then select from the foreign table the required window point(s) for the local object in question.

8.1.4. Dynamic initiation (First Event based)

The time window may be initiated dynamically when it is dependent on the first event of the object. For example, in the ‘progress_report’ example, some students get enrolled in the Fall semester, some in the Spring, and some in the Summer semester. In this example, students (objects) might not have their window’s initiated at the same point in time. The same applies in the ‘H1_applicant’ example, where the starting point is specific to a particular user, and reflects the first application filed for that user. Therefore, apart from foreign objects discussed in 8.1.3., difference in window’s start point can occur when the time is dependent on the object’s first event.

8.2. Start and End points Combinations:

Using the start and end points variables in the window-constraint, one can compose the following 4 combinations:

- Window’s start and end points are same for all objects in the table.
- Window’s start and end points are different for all objects in the table.
- Window start point is same but end points are different for all objects in the table.
- Window start points are different but the end point is the same for all objects in the table.

8.3. Notations:

For simplicity purpose, in examples 1 to 9, we had rephrased the window constraint's specifications in more general form. But when it comes to implementation, following notations were designed by us to specify the start and end points of a window in the parser.

i. Object's first event, represented as → object
ii. system's current time, represented as → now()
iii. foreign object, represented by → particular table.column name
iv. Absolute time, represented in → [YYYY-MM-DD HH:MM:SS] format
v. Relative time, represents window-size in → [YYYY-MM-DD HH:MM:SS] format. Also, as relative time represents the window-size, we have restricted its value to be no greater than [0099-99-99 23:59:59].

As we are talking about above 5 options that can be used either as start or end points, we have in total 30 combinations. The following 15 being the valid ones:

i. [Absolute, Absolute]
ii. [Absolute, Relative]
iii. [Absolute, now()]
iv. [Absolute, table.column]
v. [object, Absolute]
vi. [object, Relative]
vii. [object, now()]
viii. [object, table.column]
ix. [table.column, Absolute]
x. [table.column, Relative]

xi. [table.column, now()]

xii. [table.column, table.column]

xiii. [Relative, table.column]

xiv. [Relative, now()]

xv. [Relative, Absolute]

Whenever the specified window is absolute or fixed, we enable the repetition of such window by using the wild card (*) character. So let's say if the time specified was [****-**-03 10:10:10], then it implies that the current year and the current month with day 3 and time 10:10:10 will be picked. This way, if both the start and end time are specified using wild card format, then the window will reoccur. For illustration purpose, the additional example in subsection 7.7.4 has the following create-table statement:

```
create table internship_enrollment (
student varchar(20),
company varchar(10),
semester varchar(20),
hours varchar(2),
credit varchar(2)
/*
object(student),
Range(hours) : [("0-20",0000-05-10 01:01:01),("0-40",0000-
08-20 01:01:01),("0-20",0000-12-31 23:59:59)],
window(hours) : [****-01-01 01:01:01, now()]
*/
);
```

Note that from temporal semantics perspective, start-point can never have ‘now()’ time-format; similarly end-point can never have ‘object’ option. In addition, window-constraint should always have two arguments (start and end points). Finally, the two arguments or parameters cannot be ‘Relative’ at the same time. Based on this reasoning, the following 15 combinations of start and end points are not semantically valid.

i. [object, object]

ii. [Absolute, object]

iii. [Relative, object]

iv. [table.column, object]

v. [now(), object]

vi. [now(), now()]

vii. [now(), table.column]

viii. [now(), Absolute]

ix. [now(), Relative]

x. [object]

xi. [now()]

xii. [Absolute]

xiii. [table.column]

xiv. [Relative]

xv. [Relative, Relative]

Based on the valid combinations of these notations, here we list the actual cellular constraints’ specifications for the examples from 1 to 9.

```
Example-1:

/*

object(student_id),

window(grade): [semester_info.end_date, 0000-01-07 00:00:00]

*/

Example-2:

/*

object(customer_id),

aggregate(withdrawal_amount)     :     [+,     <=,     400],

window(withdrawal_amount)  :  [0000-01-01  00:00:00,  now()]

*/

Example-3:

/*

object(user_name),

compulsory(busstop_location) : [1],

window(busstop_location)  :  [bus_pass.valid_from,0000-01-31

00:00:00]

*/

Example-4:

/*

object(applicant_id),

frequency(decision) : [3],

interval(decision) :[1,0000-12-31 00:00:00],
window(decision) : [object, 0002-05-31 00:00:00]
*/
Example-5:
/*
object(student_id),
interval(score) :[1,0000-1-21 00:00:00],
frequency(score) : [5],
Window(score) : [0000-12-31 00:00:00, now()]
*/
Example-6:
/*
object(user_id),
penalty(pass_word):    [credentials.passphrase,    =,    5   ,
0000-00-00 23:59:59],
window(pass_word) : [0000-00-00 12:00:00, now()]
*/
Example-7:
/*
object(customer_id),
Range(expenditure_amount) : [("0-300", 0000-12-31 00:00:00),
("0-2000",   0002-12-31   00:00:00),   ("0-5000",   0004-12-31
00:00:00), ("0-15000", 0099-12-31 00:00:00)],
window(expenditure_amount) : [cards.issue_date, now()]
*/
```

```
Example-8:
/*
object(student_id),
sequence(graduation_year)  :  [Freshman,  Sophomore,  Junior,
Senior],
interval(graduation_year)     :[1,0000-12-31     00:00:00],
Window(graduation_year) : [object, 0005-12-31 00:00:00]
*/
Example-9:
/*
object(student_id),
Threshold(grade)  :  [("C|C+|C-|D|D+|D-",  2),  ("F",  1)],
Window(grade) : [0002-12-31 00:00:00, now()]
*/
Example-10:
/*
object(student_id),
frequency(submission_date) : [2],
window(submission_date) : [assignments.due_date, now()]
Range(points) : [("0-75",0000-1-7 00:00:00),("0-50",0000-1-
14 00:00:00),("0-0",0000-1-21 00:00:00)],
window(points):[assignments.due_date,students.submission_date]
*/
```

Chapter 9

Experiments and Results

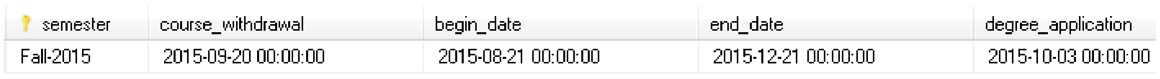
In chapter-8 we explained various ways to initiate and terminate the time window, and discussed how the window boundaries for all objects can either be same or different, depending upon the business needs. In this chapter we present the experiments conducted using the cellular constraints and display their results. Along with the relational tables, we represent the event tables that are responsible for recording the previous events' information. For brevity, we display only the data sufficient for demonstration purpose.

9.1. Window Constraint

Let us go back to Example-1, which provided the schema information. The temporal requirement is that the instructor should post grades within a week after the semester is over. Therefore, the value in the 'grade' column is subjected to change only in the week following the final day of the semester. Based on the required subset of data displayed in Table 9.1 and Table 9.2, we focus on a new incoming event as depicted in Figure 9.1.

Foreign Table:

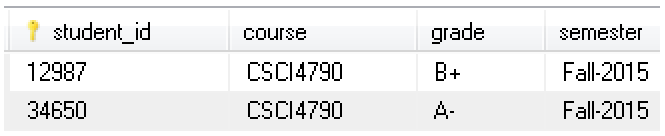
Table 9.1 semester_info



semester	course_withdrawal	begin_date	end_date	degree_application
Fall-2015	2015-09-20 00:00:00	2015-08-21 00:00:00	2015-12-21 00:00:00	2015-10-03 00:00:00

Relational Table:

Table 9.2 class_roster



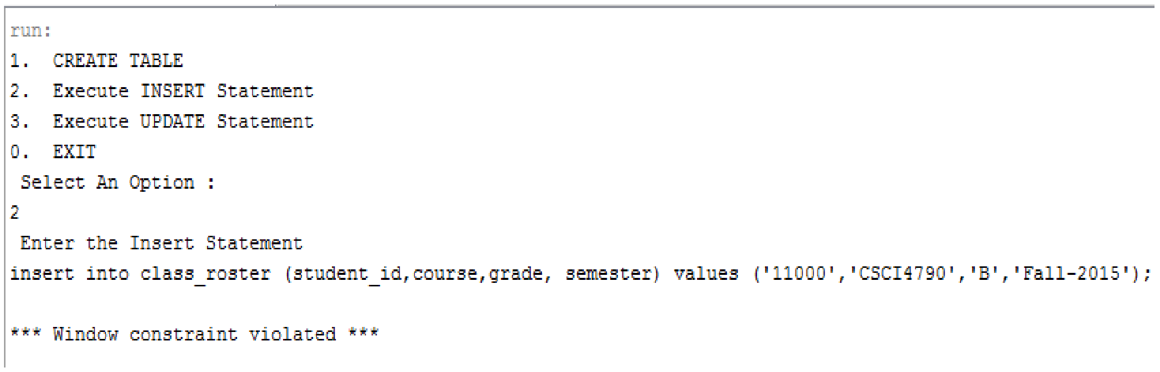
student_id	course	grade	semester
12987	CSCI4790	B+	Fall-2015
34650	CSCI4790	A-	Fall-2015

Event Table: No Event Table is constructed for window's constraint.

Incoming event (new data) attempted on: 2015-12-30 09:30:40am

- `insert into class_roster (student_id,course,grade, semester) values ('11000','CSCI4790','B','Fall-2015');`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into class_roster (student_id,course,grade, semester) values ('11000','CSCI4790','B','Fall-2015');

*** Window constraint violated ***
```

Figure 9.1 Window Constraint Violation

The result indicates that the grade assignment for student-id#11000 is blocked, as it was attempted by the instructor after the due date for posting grades (a week after the semester's end).

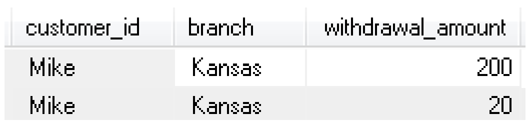
9.2.Aggregate Constraint

Example-2 explained the temporal requirement and provided the schema information. Suppose that user 'Mike' has withdrawn $220 from his bank account, and on the same day he attempts another withdrawal of $240.

Based on the data provided in Table 9.3 and the event information in Table 9.4, Figure 9.2 showcases the result for the new incoming event.

Relational Table:

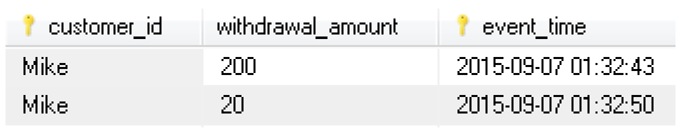
Table 9.3 transactions



customer_id	branch	withdrawal_amount
Mike	Kansas	200
Mike	Kansas	20

Event Table:

Table 9.4 transactionsaggregatewithdrawal_amount

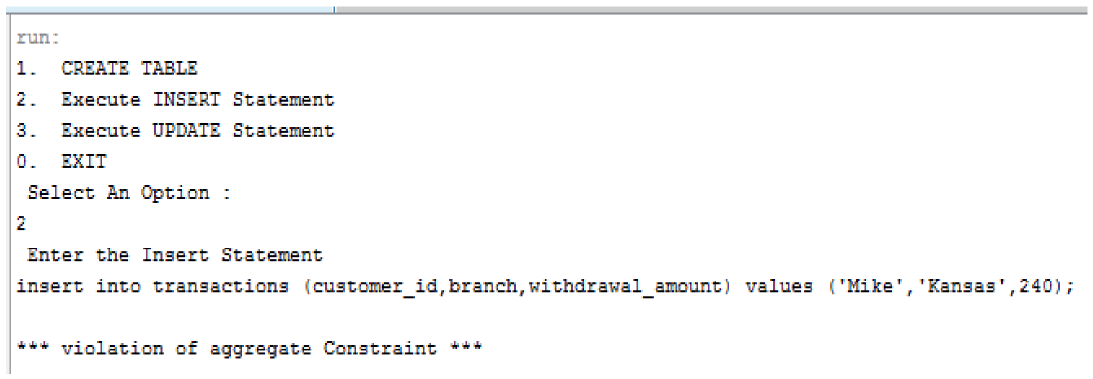


customer_id	withdrawal_amount	event_time
Mike	200	2015-09-07 01:32:43
Mike	20	2015-09-07 01:32:50

Incoming Event at: 2015-09-07 10:40:54am

– `insert into transactions (customer_id, branch, withdrawal_amount) values ('Mike','Kansas',240);`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into transactions (customer_id,branch,withdrawal_amount) values ('Mike','Kansas',240);

*** violation of aggregate Constraint ***
```

Figure 9.2 Aggregate Constraint Violation

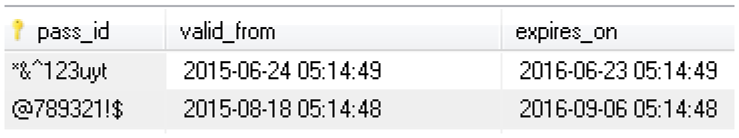
The result indicates that the third transaction attempted by customer ‘Mike’ got blocked as it violated the aggregate constraint.

9.3.Compulsory Constraint

Example-3 explained the temporal requirements and provided the schema information. Suppose that user James had been using the bus pass regularly, but has not used it recently for last 35 days. Based on the information available in Table 9.5, Table 9.6 and Table 9.7, we focus on a new incoming event as depicted in Figure 9.3.

Foreign Table:

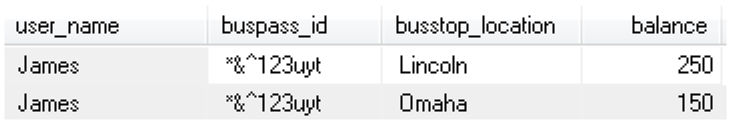
Table 9.5 bus_pass



pass_id	valid_from	expires_on
*&^123uyt	2015-06-24 05:14:49	2016-06-23 05:14:49
@789321!$	2015-08-18 05:14:48	2016-09-06 05:14:48

Relational Table:

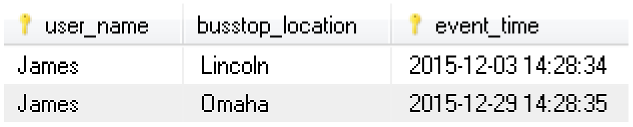
Table 9.6 commuters



user_name	buspass_id	busstop_location	balance
James	*&^123uyt	Lincoln	250
James	*&^123uyt	Omaha	150

Event Table:

Table 9.7 commuterscompulsorybusstop_location

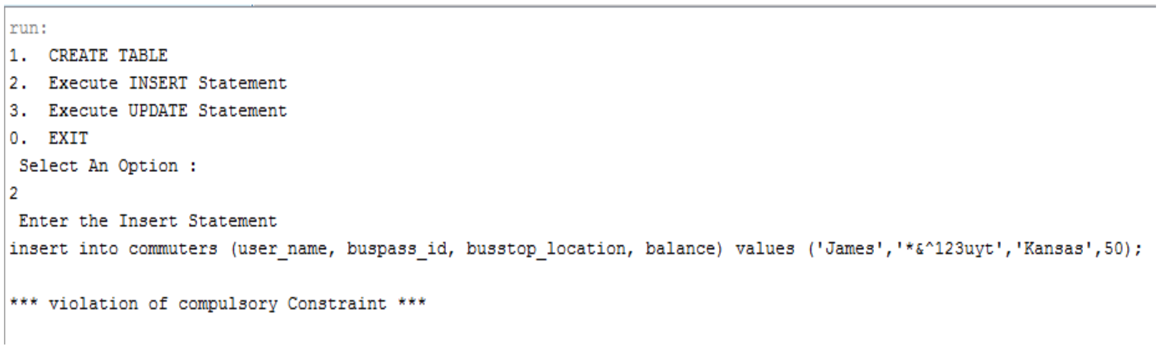


user_name	busstop_location	event_time
James	Lincoln	2015-12-03 14:28:34
James	Omaha	2015-12-29 14:28:35

Incoming event at: 2016-02-03 07:08:09am

– `insert into commuters (user_name, buspass_id, busstop_location, balance) values ('James','*&^123uyt', 'Kansas',50);`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into commuters (user_name, buspass_id, busstop_location, balance) values ('James','*&^123uyt','Kansas',50);

*** violation of compulsory Constraint ***
```

Figure 9.3 Compulsory Constraint Violation

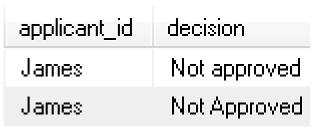
The bus pass is not accepted or considered valid anymore, as it was not used for an interval of more than a month, violating the imposed compulsory constraint.

9.4.Interval Constraint

Example-4 explained the temporal requirements and provided the schema information. Suppose user 'James' was declined H1 visa twice times, and he is filing the application one more time. Based on the sample dataset provided in Table 9.10 and the prior event information in Table 9.11, Figure 9.5 shows the verdict on the incoming event.

Relational Table:

Table 9.8 H1_applicants



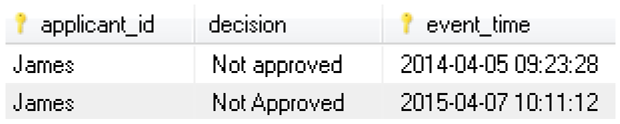
applicant_id	decision
James	Not approved
James	Not Approved

Here, because of the involvement of two cellular-constraints, there are two event tables that get populated.

Event tables: H1_applicantsfrequencydecision, and H1_applicantsintervaldecision

As both of the event tables have identical content, we display just one:

Table 9.9 H1_applicantsintervaldecision

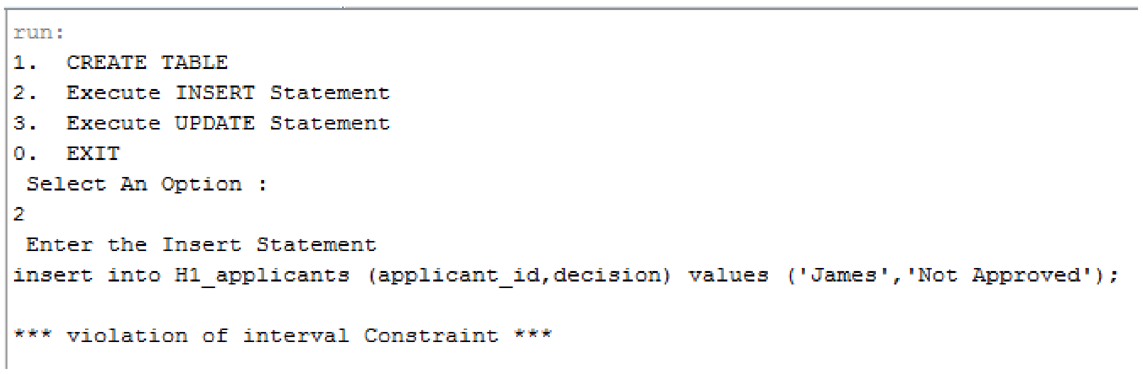


applicant_id	decision	event_time
James	Not approved	2014-04-05 09:23:28
James	Not Approved	2015-04-07 10:11:12

Incoming event (new data): 2016-02-21 07:15:43am

– `insert into H1_applicants (applicant_id,decision) values ('James','Not Approved');`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into H1_applicants (applicant_id,decision) values ('James','Not Approved');

*** violation of interval Constraint ***
```

Figure 9.5 Interval Constraint Violation

Although the frequency constraint is not violated by the new event, it does violate the interval constraint. The third H1 application of user 'James' is forbidden, because it violates the interval constraint, as the minimum gap between the two events is not met.

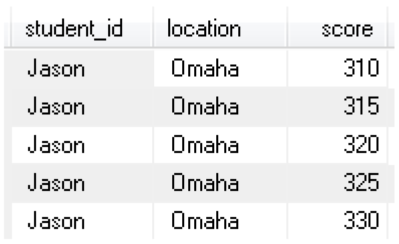
9.5.Frequency Constraint

Example-5 explained the temporal requirements and provided the schema information. Suppose user 'Jason' has attempted the GRE exams five time, and is appearing now for the

6th time. Given the data available in Table 9.8 and its respective event Table 9.9, we focus on Figure 9.4 to see the outcome of the incoming event at hand.

Relational table:

Table 9.10 GRE_attempts



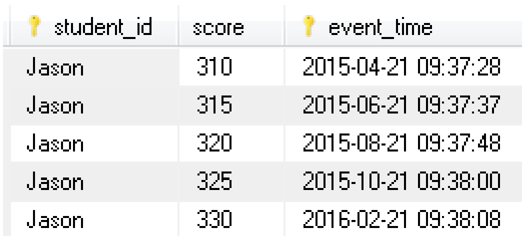
student_id	location	score
Jason	Omaha	310
Jason	Omaha	315
Jason	Omaha	320
Jason	Omaha	325
Jason	Omaha	330

Because of the involvement of two cellular constraints, there are two event tables that get populated.

Event tables: GRE_attemptsfrequencyscore, and GRE_attemptsintervalscore.

As both of the event tables have identical content, we display just one:

Table 9.11 GRE_attemptsfrequencyscore

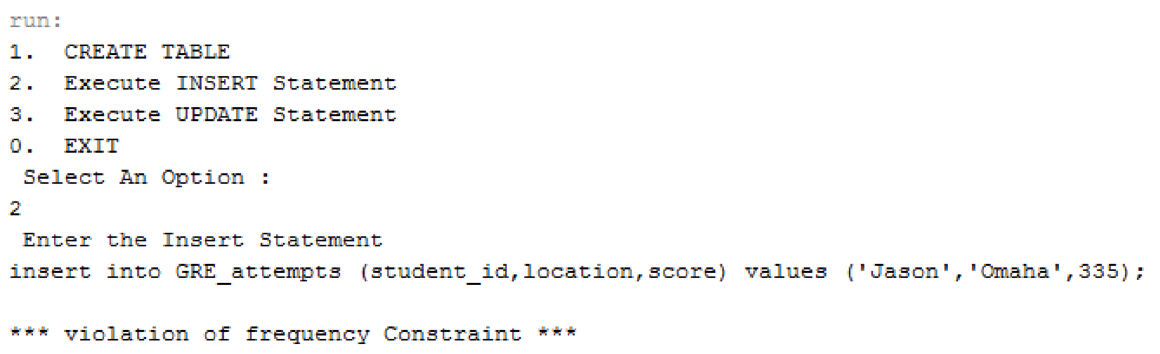


student_id	score	event_time
Jason	310	2015-04-21 09:37:28
Jason	315	2015-06-21 09:37:37
Jason	320	2015-08-21 09:37:48
Jason	325	2015-10-21 09:38:00
Jason	330	2016-02-21 09:38:08

Incoming Event at: 2016-02-26 10:10:09am

- `insert into GRE_attempts (student_id,location,score) values ('Jason','Omaha',335);`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into GRE_attempts (student_id,location,score) values ('Jason','Omaha',335);

*** violation of frequency Constraint ***
```

Figure 9.4 Frequency Constraint Violation

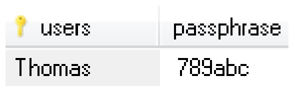
Jason's sixth exam attempt information was not recorded, since it did not obey the frequency constraint that demanded maximum of 5 attempts in a year. In this example, even interval constraint was violated. But after processing the frequency constraint the parser did not process rest of the imposed cellular constraints (in this case 'interval'), since they will not affect or change the outcome reached earlier (prohibition of incoming event).

9.6.Penalty Constraint

Example-6 explained the temporal requirements and provided the schema information. Suppose user 'Thomas' has failed to provide correct password in five successive attempts, and now he is trying to sign in again. The login attempts have been recorded in event table 9.13. The relational Table 9.12 stores only the most recent login information, long with the correct passphrase. Figure 9.6 shows the result of a new incoming event.

Foreign Table:

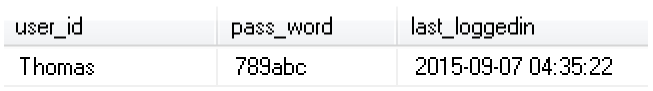
Table 9.12 credentials



users	passphrase
Thomas	789abc

Relational Table:

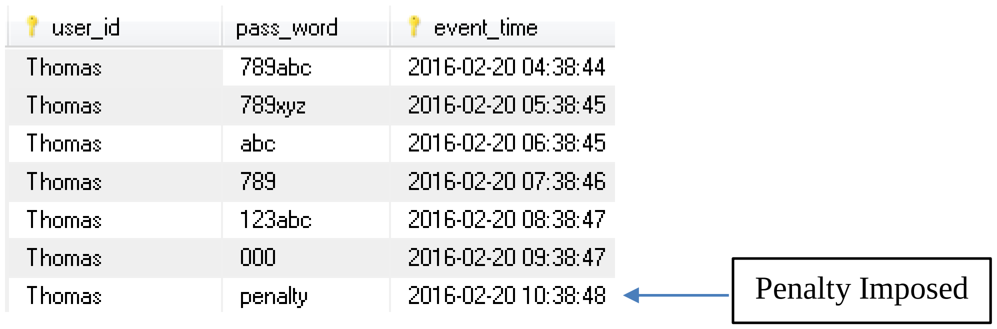
Table 9.13 signin_info



user_id	pass_word	last_loggedin
Thomas	789abc	2015-09-07 04:35:22

Event Table:

Table 9.14 signin_infopenaltypass_word

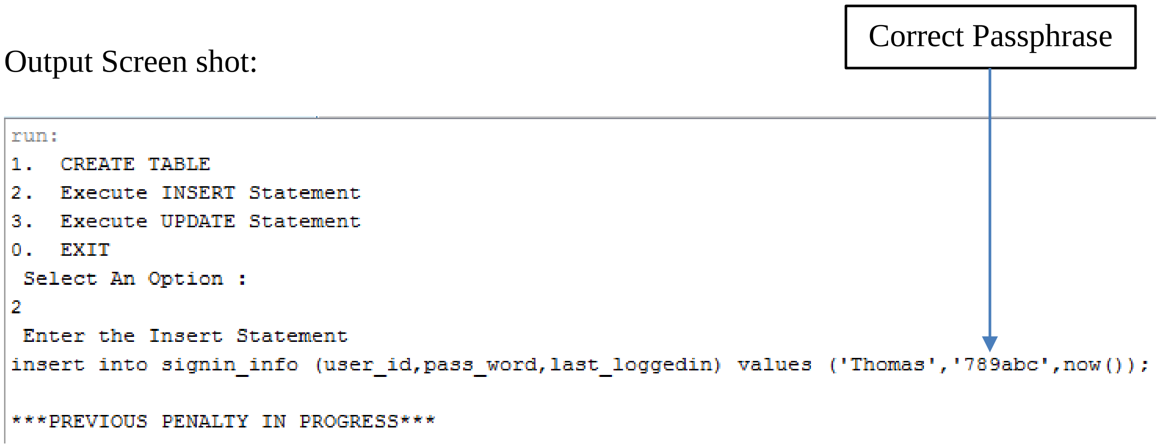

Penalty Imposed

user_id	pass_word	event_time
Thomas	789abc	2016-02-20 04:38:44
Thomas	789xyz	2016-02-20 05:38:45
Thomas	abc	2016-02-20 06:38:45
Thomas	789	2016-02-20 07:38:46
Thomas	123abc	2016-02-20 08:38:47
Thomas	000	2016-02-20 09:38:47
Thomas	penalty	2016-02-20 10:38:48

Incoming event at: 2016-02-21 09:43:07am

```
– insert into signin_info (user_id,pass_word,last_loggedin)
  values ('Thomas','789abc',now());
```

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into signin_info (user_id,pass_word,last_loggedin) values ('Thomas','789abc',now());

***PREVIOUS PENALTY IN PROGRESS***
```

Figure 9.6 Penalty Constraint Violation

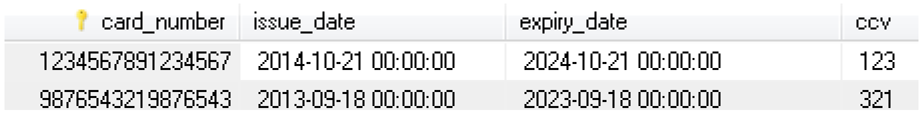
After the fifth wrong attempt, a penalty of 24 hours was imposed on user ‘Thomas’ account. Although the sixth attempt has the correct password, it gets blocked as the penalty is still in effect.

9.7.Range Constraint

Example-7 explained the temporal requirements and provided the schema information. Suppose user ‘Harry’ has been using the credit card since 2014 and has reached a credit limit of $2000. He goes on a shopping spree and swipes the credit card to make a payment of $2500. The parser retrieves the relevant data as displayed in Table 9.14 and Table 9.15. Figure 9.7 shows the result for this new incoming event.

Foreign Table:

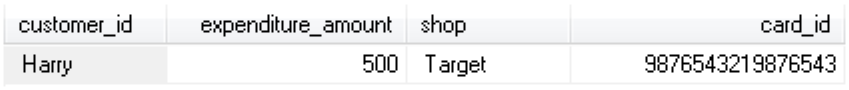
Table 9.15 cards



card_number	issue_date	expiry_date	ccv
1234567891234567	2014-10-21 00:00:00	2024-10-21 00:00:00	123
9876543219876543	2013-09-18 00:00:00	2023-09-18 00:00:00	321

Relational Table:

Table 9.16 card_users



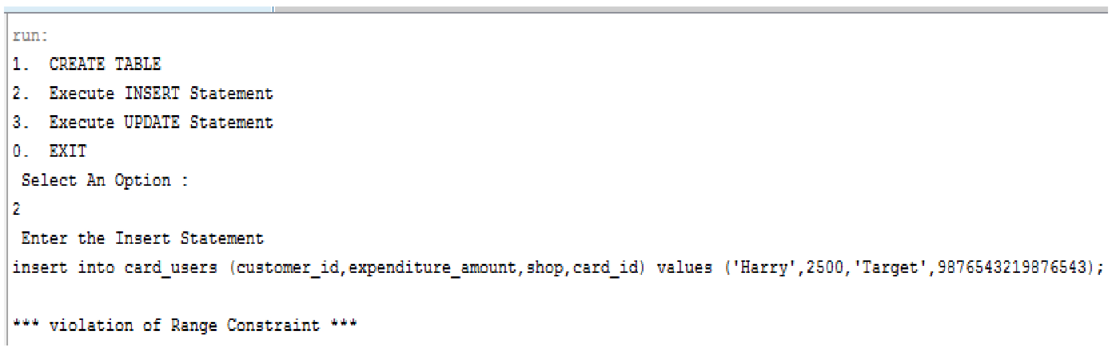
customer_id	expenditure_amount	shop	card_id
Harry	500	Target	9876543219876543

Event Table: No event table for the Range constraint.

Incoming event at: 2015-09-21 10:09:08am

```
– insert into card_users (customer_id,expenditure_amount,
  shop,card_id) values ('Harry',2500,'Target',
  9876543219876543);
```

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into card_users (customer_id,expenditure_amount,shop,card_id) values ('Harry',2500,'Target',9876543219876543);

*** violation of Range Constraint ***
```

Figure 9.7 Range Constraint Violation

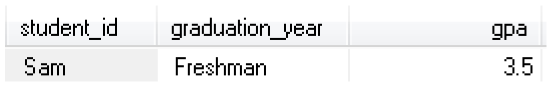
The transaction is aborted as the amount in the incoming event exceeds the limit specified in the range constraint. This amount would have been valid or legitimate after three years from the card's activation or issue date.

9.8. Sequence Constraint

Example-8 explained the temporal requirements and provided the schema information. Suppose user 'Sam' got enrolled as a freshman in 2014. Data relevant to this example are displayed in the relational table, Table 9.16 and event table, Table 9.17. Figure 9.18 shows the result of the new incoming event.

Relational Table:

Table 9.17 progress_report



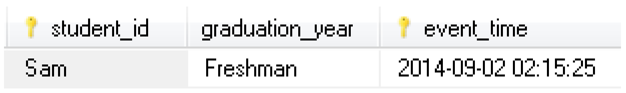
student_id	graduation_year	gpa
Sam	Freshman	3.5

Because of the existence of two cellular constraints, two event tables get populated. As both of them have identical content, we display results of just one:

Event tables: progress_reportintervalgraduation_year, and

progress_reportsequencegraduation_year.

Table 9.18 progress_reportsequencegraduation_year

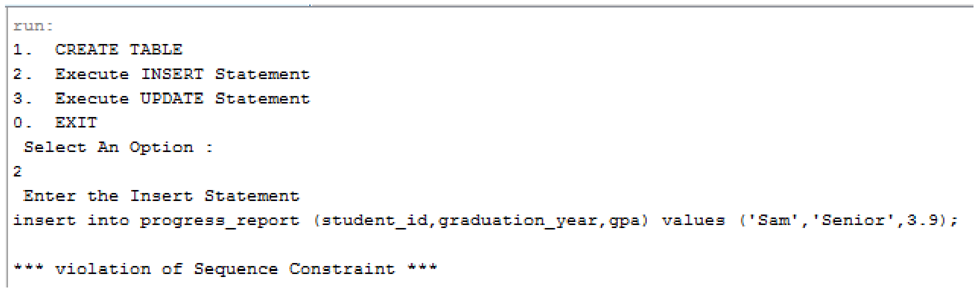


student_id	graduation_year	event_time
Sam	Freshman	2014-09-02 02:15:25

Incoming event at: 2015-09-04 09:45:07am

– `insert into progress_report (student_id,graduation_year, gpa) values ('Sam','Senior',3.9);`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into progress_report (student_id,graduation_year,gpa) values ('Sam','Senior',3.9);

*** violation of Sequence Constraint ***
```

Figure 9.8 Sequence Constraint Violation

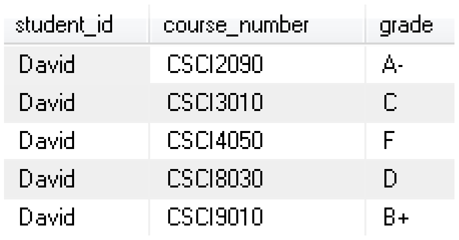
The incoming event does satisfy the interval constraint but then fails to obey the order specified in the sequence constraint. The event is eventually blocked.

9.9.Threshold Constraint

Example-9 explained the temporal requirements and provided the schema information. Suppose that user 'David' has received the following grades for five of his enrolled courses, and is waiting for the grade on sixth one. Based on the required data displayed in Table 9.18 and in Table 9.19, Figure shows the outcome on the new incoming event.

Relational Table:

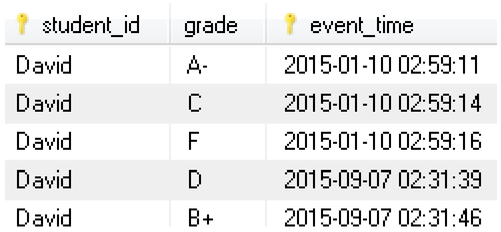
Table 9.19 transcripts



student_id	course_number	grade
David	CSCI2090	A-
David	CSCI3010	C
David	CSCI4050	F
David	CSCI8030	D
David	CSCI9010	B+

Event Table:

Table 9.20 transcriptsthresholdgrade

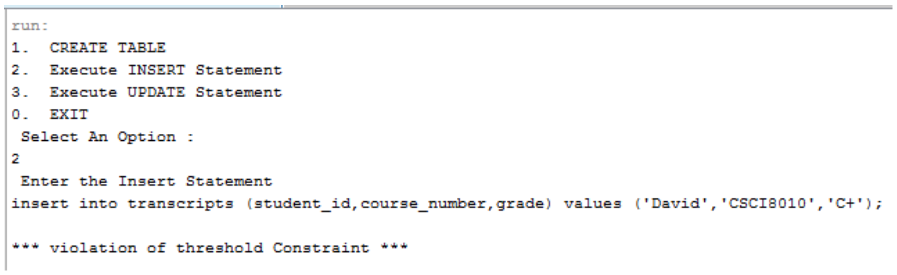


student_id	grade	event_time
David	A-	2015-01-10 02:59:11
David	C	2015-01-10 02:59:14
David	F	2015-01-10 02:59:16
David	D	2015-09-07 02:31:39
David	B+	2015-09-07 02:31:46

Incoming event at: 2015-09-07 14:30:54pm

– `insert into transcripts (student_id,course_number,grade) values ('David','CSCI8010','C+');`

Output Screen shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
2
 Enter the Insert Statement
insert into transcripts (student_id,course_number,grade) values ('David','CSCI8010','C+');

*** violation of threshold Constraint ***
```

Figure 9.9 Threshold Constraint Violation

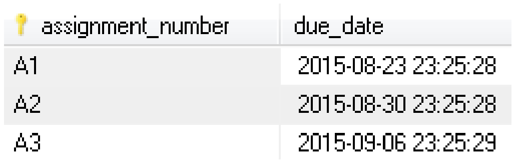
The incoming event violates the threshold constraint, which limits the number of grades below B- to two.

9.10. Update Statement Example

In this example we see how the 'update' statement gets scrutinized in the same manner as the 'insert' statements in the examples before. As along as there is any cellular constraint imposed, whether it's insert or update, the statement will be inspected by the parser. Example-10 explained the temporal requirements and provided the schema information. Suppose student Jacob submitted assignment 'A1' 10 days after the due date. After grading, the instructor decides to give 60 points to this student. Based on the subset of data shown in Table 9.20 and Table 9.21, Figure 9.10 displays the result of the parser on the incoming event.

Foreign Table:

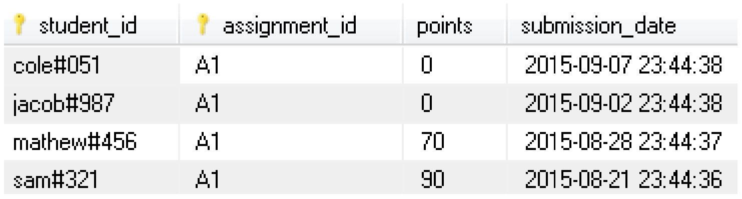
Table 9.21 assignments



assignment_number	due_date
A1	2015-08-23 23:25:28
A2	2015-08-30 23:25:28
A3	2015-09-06 23:25:29

Relational Table:

Table 9.22 students



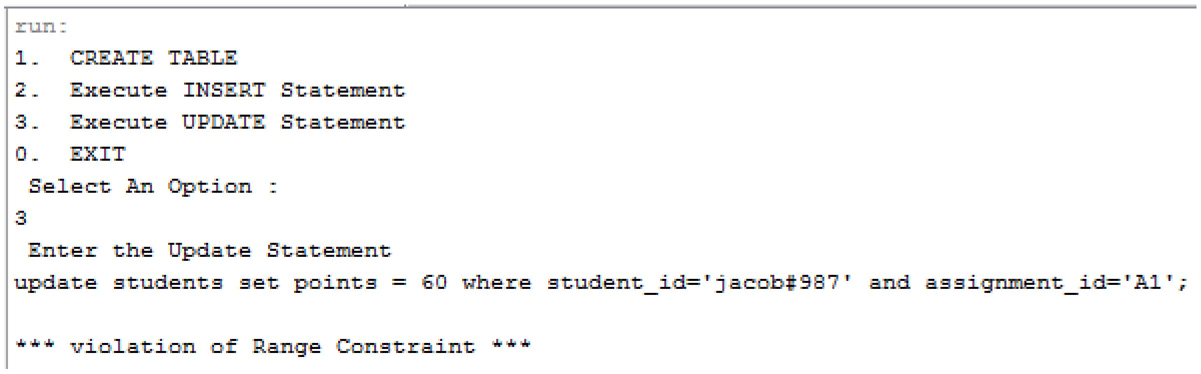
student_id	assignment_id	points	submission_date
cole#051	A1	0	2015-09-07 23:44:38
jacob#987	A1	0	2015-09-02 23:44:38
mathew#456	A1	70	2015-08-28 23:44:37
sam#321	A1	90	2015-08-21 23:44:36

Incoming Event at: 2015-09-09 05:43:21am

Event table: No Event table for Range constraint

```
update students set points = 60 where student_id= 'jacob#987'
and assignment_id='A1';
```

Output Screen Shot:



```
run:
1.  CREATE TABLE
2.  Execute INSERT Statement
3.  Execute UPDATE Statement
0.  EXIT
 Select An Option :
3
 Enter the Update Statement
update students set points = 60 where student_id='jacob#987' and assignment_id='A1';

*** violation of Range Constraint ***
```

Figure 9.10 Update Statement Violation

The update statement is prohibited because the points exceed the range specified in the constraint. The event is being submitted after the due date, when the range shrinks from 0-100 to 0-50. As the incoming event has a value of 60 (> 50), it violates the range constraint and is banned.

Another example where our research can be applied is in the ‘parking lot ticket management system’ of schools and universities. The schools issue different types of parking permits. Their costs vary based on how long they remain valid and the timespan they cover within the 24 hour day. They specify the time when cars should be allowed inside the parking lot, and when they should be prohibited. Currently manual intervention is required so that an authorized person with a scanner device in hand has to tour around and examine the parking permits to see whether any of the parked cars violate the permit rules, so that tickets can be issued to such cars.

Since the permits come with embedded barcode, a scanner integrated entrance door of the parking lot can scan the permits on cars when they arrive, and send them to a database enforcing our constraints. Based on the car’s arrival time and the type of permit (foreign object) associated with it, the ‘window’ constraint can then determine whether the pass is valid at that point in time or not. If valid, an entry is made into the database to record the arrival of a car having valid parking permission. A trigger can be associated with such entry to open the entrance door. By incorporating our research into the existing database design, we can make such manually operated systems automated, less error prone, more effective, and more efficient.

Chapter 10

Related Work

In the above chapters we explained, implemented, demonstrated, and illustrated our proposed work through results. Here, we will compare different database design techniques employed in the past that have considered (directly or indirectly) the 'time' factor while conceptualizing the data model.

Conventional databases are static, as they are populated or manipulated based on users' requests. But the applications demand intelligent databases that automate manual intervention and therefore act, or react when the appropriate time comes. It is the time factor that induces such dynamic behavior into the database. Some existing techniques that manage to introduce partial dynamism into the database are: real time databases, temporal databases, active databases, complex event processing, big-data, and data-warehouse. Each of them utilizes time factor in different light.

In general, time has in the past played different roles in the database design process, and can be mainly classified into: valid time, transaction time, response time, and time to leave (TTL). Valid-time indicates when the data/fact was true (or when the event occurred) in the real world; transaction time represents the time when the data was made known or recorded (through insert or update) into the database; TTL represents the futuristic time when the data will be deleted; and response time sets the deadline for the data or message to be delivered by the database to any dependent applications. Overall, the 'time' dimension in databases helps determine when the data was active (and not expired), or when and how many times the data underwent a change, or when the outdated data should be deleted, or what the deadline is before which the data should be dispatched or reported.

The validity, appearance, disappearance, and deliverance of data is well captured using the time dimension.

Recent advances in design techniques have shown how the databases can either automatically *act*, or *react* to the incoming events; on the contrary, we propose a technique where the incoming events are *preempted*. Below we will see how our research differs from the present approaches, and how it contributes to the existing state of the art. We begin with linear temporal logic, which although not a ‘database concept’, has been an influential factor behind our research.

10.1. Linear Temporal Logic

Propositional (or symbolic) logic deals with statements that have constant truth values regardless of time. But there are instances where truth values of statements are not meant for eternity. Propositional logic, due to its inadequacy to incorporate time, was not expressive enough to describe such systems. Linear Temporal logic (LTL) was introduced to address this issue and deal with statements that are qualified by time; i.e., where the truth value of statements are subjected to change as time progresses. Through its operators/symbols, LTL makes it possible to specify temporal ordering. In other words, LTL enables the encoding of temporal paths to capture the change in truth values of statements, using its set of operators. Similar to LTL, we hold the fact that the values present in the cells of a database table come with an expiry, as the cell is bound to undergo change (inserts/update/deletes) over time. Based on the temporal-business requirements, we propose a set of constraints that encode all possible changes that the cells in the database-column can go through. Incoming value comes with an expiry, and is soon overwritten by new instances. We control this iterative process, where for a given cell old value is replaced by new value.

The operators in LTL (neXt, Globally, Finally, Until, and Release) ensure that the prepositions or constructs follow the established path; i.e., how the state of a system should change over time. Similar to LTL, we ascertain through our constraints that the values appearing in a cell over time follow certain pattern. Our constraints ensure that after certain time has elapsed, the incoming values in the cells of a database table are legitimate or expected as per the business's temporal requirements. Also, just like LTL's 'safety' property, our constraints ensure that something bad (invalid values) never happens. But then unlike LTL's 'liveliness' property, our constraints are not certain of what good (through inserts/updates) awaits the cells.

The LTL concept was not considered or applied rigorously in database design theory. Through our research we bring the concept of linear temporal logic inside the database design by proposing our own constraints, and not the ones existing in LTL. We did not borrow the LTL operators directly because those LTL operators deal only with the time factor. Being the fusion of time and data, our research required constraints that deal with both events' value and time (sequence, range, threshold, aggregate). LTL operators provided no way to quantify time, so we proposed new constraint (window) for this purpose. To address the cardinality of events, we proposed four more constraints (penalty, compulsory, frequency, and interval). Although there is no correlation between LTL's operators and our cellular constraints, the objective remains the same. We did not focus on Computational Tree Logic (a superset of LTL), because it deals with more than one possible path, where the future is therefore non-deterministic. For the same reason, CTL has 2 additional operators: ALL & EXISTS (universal and existential quantifiers), dealing explicitly with paths. Banieqbal and Barringer [4], and Rescher [58] discuss temporal logic specifications in details.

10.2. Temporal databases

Conventional databases store a single instance or snapshot of the data. To store multiple snapshots of the same data over time, temporal databases were proposed. Temporal database management, and its valid and transaction time associated with the data are elaborated in articles [2], [18], [20], [30], [37], [38], [39], [40], [44], [45], [52], [61], [62], [69], [70], [71], and [72]. Valid time ensures that there can be exactly one valid version of the data in a given time period, ensuring the integrity of data over time. Valid time helps avoid version overlapping and is used to indicate when the data was deemed valid. Transaction time is used to record the time when any changes (insert, delete or update) were made to the data. Transaction time support is necessary when there is need to roll back the state of the database to a previous point in time. Transactional time comes in handy when there is unexpected failure, or natural calamities that requires the restoration of recently backed up data.

Our approach does take into account the valid time and transaction time, but we consider only those situations in which the facts or events are recorded as soon as they become valid in reality (i.e., valid time equals transaction time). One of our constraints named ‘sequence’, can be closely related to the temporal databases, as both ensure non-overlapping versions of data. The cellular sequence constraint knows ahead of time when the incoming data is going to expire (futuristic date), whereas that is not the case in temporal databases. Our research is not limited to data's non-overlapping requirements, since we also propose constraints (example: threshold constraint) in which the overlapping versions of the same data are valid. Although attempts have been made by temporal database to incorporate some time oriented aspects in relational database design and to modify SQL to facilitate temporal specifications, little effort has been made to recognize and formalize all possible temporal requirements.

For a given object, temporal databases allow only one copy of data to reside in the window; but based on the requirements our proposed cellular constraints ensures that either 1, or more values to reside in the same time window. As per Ozsoyoglu and Snodgrass [54], "temporal model presents all the time-varying behaviors of facts and objects". Our research complements or take this ideology further, by not only capturing the past behavior of the object but also deciding the legitimacy of current action at hand based on past deeds or facts. Allen [3] talks about time intervals in detail. When time is divided into smaller segments, with a beginning and an end it's termed as an interval. There are thirteen possible relationships that can exist between time intervals. Our work focuses on the non-overlapping characteristic of the time interval, which is one of the thirteen possibilities.

10.3. Real-time database systems

Traditional databases are not inherently aware of performance issues, and therefore are not of much use in real time applications where even a slight delay in handling data (storage or retrieval) is intolerable and could be catastrophic. To overcome this bottleneck, Real Time Database Systems (RTDBS) were proposed which rely heavily on distributed data and concurrency control to produce results which are meaningful or useful because they are generated before the specified deadline. The transactions in RTDB are time constrained. Papers [8], [35], [41], [42], [43], [49], [50], [54], [57], [66], [68], [73], [77], and [78] give a thorough understanding of the mechanisms involved in RTDB.

In RTDBS, data about the target environment is continuously collected from the real world and processed in a timely manner to generate real-time response. RTDBS, also viewed as time constrained databases, focuses on response time where database operations need to be completed within certain time constraints to generate timely feedback. Since the data are being continuously generated (in streams), the writes should complete in timely fashion so that the information is made available at the earliest for the applications that are dependent on this data stream. RDBMS are more focused on providing timely replies to the events in the real world, while we focus on the timely arrival of events into the database system.

One of the similarities between RTDBS and our approach is that the data must not only be logically consistent, but also temporally consistent; i.e., they must closely reflect the current state of the controlled environment, where valid-time equals transaction time. In addition, just like RTDBS where the correctness of data depends not only on the logical correctness but also on the timeliness of actions, our work also demands similar stringency (incoming events have to satisfy the temporal dependency).

10.4. Data Mining

Data mining is a discipline that builds a model in order to capture hidden patterns in the historical datasets. These discovered patterns (association rules) are applied to real datasets for prediction purpose. The historical data are clustered into training and testing datasets, where the training dataset is used to build up the model and the testing dataset is used to determine the accuracy of the constructed model. The higher the accuracy of the model, the higher the probability of correctly predicting the behavior in real data. Data mining takes into account user history for prediction or recommendation purpose. We, on the other hand, take the user history into account and use it more for prevention purpose to avoid the future mishap from happening. Concepts of data mining are detailed in [1], [13], [21], [22], [23], [26], [31], [32], [36], and [82].

In data mining, the historical repository is taken into account to predict with a certain probability the next possible appearance of values. Similarly, our work backtracks the timeline and consider the previous occurrences of events but to preempt the next incoming occurrences of values. Unlike data mining, where the independent variables play important role in determining the value of dependent variable, our work focuses on only one variable (attribute); in equation 4.1 the dependent and independent variable are the same.

There has been research, where there are no independent variables (columns), and the prediction of the dependent variable is based upon its own values. In such research, models have been proposed to predict whether an attribute is expected to change over a given time interval, or to predict if already occurred value reappears over time or not. The model by Dong, Maurino, and Srivastava [48] tries to predict whether or not an attribute value changes over a given time interval; whereas Chiang, Doan, and Naughton [14] talk about the model that estimates the probability of how for a given attribute particular value can reappear over time. Unlike the former, we do not try to predict the change in attribute cells, as our work assumes or is rather aware in advance of the change that a cell is about to go through. Unlike the latter, we do not estimate the probability that represents a particular value’s reappearance over time; instead, using the proposed threshold constraint, we limit the number of occurrences of the same value in the cell of a column over time.

10.5. Data Warehouse

Conceptual data model describing the Extract-Transform-Load activities of warehouse are presented in [12], [27], [28], [34], [47], [76], [80], [81], [83], [84], and [85]. A data warehouse is used to store time varying collection of data, and provides features to adjust the granularity level of time, based on the query requirements. One of the ways the data can be drilled down, rolled up, pivoted, sliced, or diced is by using the time dimension. Here time is taken into account for computational purpose. Data warehouse is mostly used for Online Analytical Processing (OLAP), to generate reports or charts as they help show behavior and trends of the data (business) over a period of time. Based on this analysis, companies can foresee the growth or decline of their business and take appropriate measures to boost profits or minimize losses.

Just as time factor is used to aggregate or segregate the data across different dimensions in the data warehouse, our research associates the data appeared over a time window. But instead of different dimensions, data computation in our work is limited to the values that appear in the same cell over a certain span of the time window. In addition, data warehouse allocates separate dimension for time, whereas we classify time into frame, interval, and window.

10.6. Apache Cassandra

Apache Cassandra is an open source distributed database management system designed to handle high volumes of data. In Cassandra, as the data is large in scale, deleting outdated data becomes a repetitive and laborious process. To automate this process, Cassandra relies on the concept of Time To Live (TTL). TTL [46] ensures that the data get deleted the moment they are no longer valid. As our research being limited to insert and update DML statements, unlike Cassandra, we do not utilize the time factor to inspect DML's delete statement.

10.7. SQL and PL/SQL

In Structured Query Language (SQL) one can schedule an event by using the 'create event' statement. Similarly in Procedural Language/Structured Query Language (PL/SQL), one can automate the data manipulation process using the after/before triggers [79]. But our research automates the prohibition mechanism of data manipulation (limited to: inserts and updates) process.

Triggers are executed implicitly whenever the triggering event happens. Just like our proposed work, triggers can be used to compose complex integrity constraints that are not possible through declarative constraints enabled at table creation. However, the correlation identifier in PL/SQL called ':old', can help one trace only up-to one previous instance of a cell, and not more. In addition, there is no way to specify or represent the concept of objects in triggers. The triggers do not take into account the concepts of time-window and interval either. Finally, the triggers work at row or statement level, whereas our work functions at cell level.

The analytic-functions in SQL [89], allow users to divide query result set into groups of rows called partitions. Using a windowing clause, for each row in a partition, one can define a sliding window of data. This window determines the range of rows to be used to perform the calculations on the current row. Window sizes can be based on either a physical number of rows or a logical interval such as time. The window has a starting row and an ending row. Depending on its definition, the window may move at one or both ends. But then in addition to such fixed/static window, our work also proposes sliding and repetitive windows. Analytic-functions act at row-level, are mainly used for computational purposes, and their input is limited to the snapshot of current data. They give more control to navigate across rows in output data resulted from some computation. But our work helps get more control over the incoming values in individual cells of the table.

10.8. Complex Event Processing and Active Databases

Complex Event Processing (CEP) is the production, detection, consumption, and reaction to events. CEP filters input streams events, does pattern-matching (using rules), and aggregates low-level events to generate higher-level (complex) events. CEP analyzes data streams to detect inexact patterns that occur due to uncertainties. These functionalities of CEP are elaborated in [5], [9], [19], [29], [33], [59], [60], [75], [86], and [87].

Active databases (AD) abide by the Event-Condition-Action (ECA) rule. When an event E occurs, the condition C is evaluated, and if satisfied then the specified action A is executed. In AD, actions are issued when a change to data in the database has made defined condition become true. AD discusses, how the occurring event(s) as input can satisfy one of the requirements in the form of conditions, and make the database take actions or generate alarms. The main focus of AD has been automatic monitoring of database state and taking appropriate actions when a certain condition is met. Such mechanism helps avoid manual scrutiny, which is prone to error and is less effective and less efficient. Concepts of AD are detailed in [7], [10], [11], [17], [24], [25], [51], [55], [56], [67], and [88].

Our research incorporates the concept of ECA, but in the following manner. In our work, we treat events-E as insert or update statements; condition-C are the cellular-constraints that take incoming event along with 1, some, or all previous events into account; and action-A is either to prohibit or execute the incoming event (insert/update statement). All this is done under the supervision of time. The notable difference between our work, and AD and CEP are:

1. As AD and CEP lack time factor, they deal only with current events, and do not consider previous events. They are focused more on real-time events (situations) and hence serve different purpose. The input to AD and CEP is one or more current events, whereas the input to our proposed model is one current event along with one or more past events. Although, CEP's semantic window in [59], or sliding and batch windows [75] are used for backtracking purpose, they still are limited to the real-time or incoming data stream, and do not focus on the past events that might be previously recorded into the database.
2. AD and CEP do not judge or question the legitimacy of occurring events; rather, they accept all the incoming events passively. Our work scrutinizes the incoming simple events, and decide whether to prohibit it or not. Although semantically similar, CEP's sequence constraint and our proposed sequence constraint differ in the way they deal with events. That is, after the events have occurred, CEP checks whether they had appeared in sequence or not (post processing); our approach on the other hand stores the events only if their occurrence was in sequence at the first place (preprocessing).

3. AD and CEP do the pattern matching or condition monitoring by taking into account occurring (lower-level) events and see whether or not a complex event or action can be generated out of them. Our work considers past events to see whether or not the incoming event can be preempted. Unlike AD and CEP our focus is not to generate complex (higher level) events or actions (alarms), but to judge the incoming simple event in the light of past events.
4. We focus of the same type of event occurring at different points in time, whereas CEP and AD focus on different types of events occurring at almost same point in time to generate complex events or actions. Our research is limited to simple events, which is not the case with CEP.
5. We also differ significantly on the type of constraints or operators being used in the work. CEP deals with: sequences, conjunctions, disjunctions, and negation operators. Our research proposes: aggregate, threshold, sequence, range, window, interval, penalty, interval, and compulsory constraints.
6. In AD and CEP, operators are used to make the system more *reactive* by generating either the actions or complex events, whereas our constraints make the system more *vigilant* by letting it know whether to accept or reject the incoming event.

10.9. Summarized

To summarize the state of art and our work, we distinguish each of them based on the perspective or approach they took towards time. Techniques that introduced dynamism (to some extent) into the database are listed in table 10.7.

Table 10.1 Unique Approaches towards Time

Databases	**Approach**
AD	Reactive/Responsive w.r.t. time
CEP	Generative w.r.t. time
Real time Databases	Proactive w.r.t. time (meets the deadline)
Apache Cassandra’s TTL, PL/SQL	Automatic w.r.t. time
Temporal Databases	Distinctive w.r.t time (non-overlapping data version)
DW	Calculative w.r.t. time
DATA-MINING, Paper [7] and [8]	Predictive w.r.t. time
Our Research	Preemptive w.r.t. time

Note: In the above able, ‘w.r.t.’ stands for: ‘with respect to’.

Chapter 11

Conclusion and Future Work

We conclude by summarizing our work and discuss the existing limitations of our work, along with the possible extensions that we look forward to in near future. At the end, references and appendixes are provided for supplemental purpose.

11.1. Conclusion

As the time progresses, values in a database table keep on entering and exiting the cells. Based on the temporal-requirements, we identified patterns or relationships among these values, and then proposed cellular-constraints to ensure that these relationships are not violated by the new incoming event. The subset of proposed constraints ensures that contextual dependency between the values that are being overwritten onto the same cell at different intervals of time is preserved. These constraints ascertain that a change to the cell is done by taking into account previous events from the time window. If the occurring event is not in harmony with past events on the timeline, then the event is forbidden.

In addition, this work ensures that if the occurring event is not on-time, then the transaction associated with the insert/update statement is prohibited. One cannot go back in time and update the existing information, or in advance insert values ahead of time. In this way the data are not vulnerable to untimely inserts and updates. The arrival and departure of the values in a cell (manipulation) becomes time-dictated.

This paper talks about how the proposed constraints are imposed on the cells of column in a table, and how they differ from the traditional constraints that are applied at the column level. Just as ‘functional dependency’ helps eliminate anomalies, we proposed ‘temporal and contextual dependencies' to overcome the temporal and contextual anomalies. Through the proposed temporal and contextual dependencies, this work provides consistency of data with respect to time.

Overall, our research ascertains that data should get populated and manipulated only at the right time to achieve temporal consistency, which indirectly also brings more discipline in the execution of insert and update statements. This research also establishes the fact that past acts (events) of a given object determine the fate of its next incoming event. All this we achieve by integrating the changing or dynamic metadata (captured by the proposed cellular-constraints) with the existing static one.

11.2. Limitations and Future Work:

We have proposed a framework where the cellular constraints take into account either 1, some, or all past events from the current window to judge the incoming event and make a decision of whether or not to prohibit it. But these cellular constraints may or may not be the complete set. As listed in table 4.3, we managed to propose only nine constraints from the overall sixteen combinations. We plan to explore and see whether or not there is any possibility of adding more constraints to the proposed list. Our goal is to make this list of cellular constraints as comprehensive as possible.

We plan to extend the capabilities of the aggregate constraint so that it can handle complex operations as well. Currently the abilities of aggregate-constraint are limited, as the user can specify only a simple equation composed of one arithmetic operator, a relational operator and a constant. We plan to make it more flexible, where complex equations can be passed as parameters to this aggregate constraint. Along similar lines, we plan to enhance the sequence constraint, which is currently limited to linear ordering. In the future, we want to ensure that the sequence constraint can incorporate or support partial ordering apart from the existing linear ordering.

Currently, our research has scrutinized the insert/update DML statements. In future we would like to see if such preemption or prohibition procedures can be applied on events representing ‘select’ and ‘delete’ statements. That is no selection or deletion of data is possible until certain legitimate timespan has not arrived. We also plan to see the difference in performance when these constraints, existing at backend on server, are implemented or ran as scripts at user/client end instead.

Finally, as the event tables store the past event information, there is data redundancy. For a given object’s incoming event, a particular subset of past events from the event table are to be considered to make a decision on whether or not to prohibit the incoming event. The extra storage, lookup, and retrieval associated with the event table is an overhead and might not scale efficiently for a larger data set. Since our current experiments and results were on much smaller dataset, we plan to see the impact of high volume data on the proposed approach. In future, we also plan to work on a procedure that identifies and then removes the expired data from the event table on a regular basis.

References:

1. R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger, "The QUEST Data Mining System", International Conference on Knowledge Discovery in Databases and Data Mining, August 1996, pp. 244-249.
2. I. Ahn, and R. Snodgrass, "Performance evaluation of a temporal database management system", ACM SIGMOD international conference on Management of data (SIGMOD '86), 1986, pp. 96-107.
3. J. F. Allen, "Maintaining knowledge about temporal intervals", Communications of the ACM, Volume 26, Issue 11, November 1983, pp. 832-843.
4. H. Banieqbal, and H. Barringer, "Temporal Logic in Specification: Altrincham, Uk, April 8-10, 1987 : Proceedings", 1989, Springer-Verlag Publication.
5. A. Barros, G. Decker, and A. Grosskopf, "Complex events in business processes", 10th International Conference on Business Information Systems (BIS), 2007, pp. 29–40.
6. C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration", ACM Computing Surveys (CSUR), Volume 18 Issue 4, Dec. 1986, pp. 323-364.
7. M. Berndtsson, and J. Hansson, "Issues in Active Real-Time Databases", ARTDB, 1995, pp. 142-157.
8. A. Bestavros, "Advances in real-time database systems research", ACM SIGMOD Record, Volume 25, Issue 1, March 1996, pp. 3-7.
9. P. Bizarro, "BiCEP - Benchmarking Complex Event Processing Systems", Dagstuhl Seminar Proceedings 07191, Event Processing, Germany, November 2007.
10. S. Chakravarthy, "Early active database efforts: a capsule summary", IEEE Transactions on Knowledge and Data Engineering, Volume 7, Issue 6, December 1995, pp. 1008-1010.
11. S. Chakravarthy, and D. Mishra, "Snoop: An expressive event specification language for active databases", Data & Knowledge Engineering, Volume 14, Issue 1, pp. 1-26.
12. S Chaudhuri, and U Dayal, "An Overview of Data Warehousing and OLAP Technology", ACM SIGMOD Record, Volume 26, Issue 1, March 1997, pp. 65–74.
13. M.S. Chen, J. Han, P. S. Yu, "Data mining: an overview from a database perspective", IEEE Transactions on Knowledge and Data Engineering, Volume 8, Issue 6, 1996, pp. 866-883.
14. Y.H. Chiang, A. Doan, and J. F. Naughton, "modeling entity evolution for temporal record matching", ACM SIGMOD International Conference on Management of Data (SIGMOD '14), 2014, pp. 1175-1186.
15. E. F. Codd, "A relational model of data for large shared data banks", Communications of the ACM, Volume 13 Issue 6, June 1970, Pages 377-387.
16. C.J. Date, " An introduction to Database Systems", 8th Edition, Addison Wesley Publication.
17. U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin. D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny and R. Jauhari, "The HiPAC Project: Combining Active Databases and Timing Constraints", ACM SIGMOD Record - Special Issue on Real-Time Database Systems, Volume 17, Issue 1, March 1988, pp. 51-70.

18. C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio, and G. Wiederhold, "A consensus glossary of temporal database concepts", ACM SIGMOD Record, Volume 23, Issue 1, March 1994, pp. 52-64.
19. M. Eckert, and F. Bry, "Complex Event Processing (CEP)", Informatik Spektrum 32(2), 2009, pp. 163-167.
20. N. Edelweiss, P.N. Hubler, M.M. Moro, and G. Demartini, "A Temporal Database Management System Implemented on Top of a Conventional Database", Computer Science Society, 2000. SCCC '00. Proceedings, 2000, pp. 58-67.
21. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases", AI Magazine, Volume 17, Issue 3, 1996, pp. 37-54.
22. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Knowledge Discovery and Data Mining: Towards a Unifying Framework", 2nd International Conference on Knowledge Discovery and Data Mining, 1996, pp. 82-88.
23. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data", Communications of the ACM, Volume 39, Issue 11, November 1996, pp. 27-34.
24. H. R. Firoozy-Najafabadi, and A. H. Navin, "Rule Scheduling Methods in Active Database Systems: A Brief Survey", 6th International Conference on Application of Information and Communication Technologies (AICT), 2012, pp. 1-5.
25. P. Fraternali, P. D. Milano, and L. Tanca, "A Structured Approach for the Definition of the Semantics of Active Databases", ACM Transactions on Database Systems (TODS), Volume 20, Issue 4, December 1995, pp. 414-471.
26. C. Glymour, D. Madigan, D. Pregibon, and P. Smyth, "Statistical inference and data mining", Communications of the ACM, Volume 39, Issue 11, November 1996, pp. 35-41.
27. M. Golfarelli, and S. Rizzi, "A methodological framework for data warehouse design", 1st ACM international workshop on Data warehousing and OLAP (DOLAP '98), 1998, pp. 3-9.
28. P. Gray, "Present and future directions in data warehousing", ACM SIGMIS Database, Volume 29, Issue 3, 1998, pp. 83-90.
29. D. Gyllstrom, E. Wu, H. Chae, Y. Diao, P. Stahlberg, and G. Anderson, "SASE: Complex Event Processing over Streams", Conference on Innovative Data Systems Research (CIDR), 2007.
30. H. Gregersen, and C. S. Jensen, "Temporal Entity-Relationship Models—A Survey", IEEE Transactions on Knowledge and Data Engineering, Volume 11, Issue 3, May/June 1999, pp. 464 - 497.
31. J. Han, M. Kamber, and J. Pei, "Data Mining: Concepts and Techniques", Second Edition, Morgan Kaufmann Publication
32. D.J. Hand, "Data mining: statistics and more?", The American Statistician, Volume 52, Issue 2, May 1998, pp. 112-118.
33. W. Hu, W. Ye, Y. Huang, and S. Zhang, “Complex Event Processing in RFID Middleware: A Three Layer Perspective”, Third International Conference on Convergence and Hybrid Information Technology (ICCIT '08), 2008, pp. 1121- 1125.

34. B. Husemann, J. Lechtenborger, and G. Vossen, "Conceptual data warehouse design", International Workshop on Design and Management of Data Warehouses (DMDW), 2000.
35. N. Idoudi, C. Duvallet, B. Sadeg, R. Bouaziz, and F. Gargouri, "How to model a real-time database?", IEEE International Symposium on Object/Component/Service-Oriented Real-time distributed Computing (ISORC '09), 2009, pp. 321–325.
36. J. Jackson, "Data mining: A conceptual overview", Communications of the Association for Information Systems, Volume 8, 2002, pp. 267–296.
37. C. S. Jensen, and R. Snodgrass, "Temporal Specialization and Generalization", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, Volume 6, Issue 6, December 1994, pp. 954-974.
38. C.S. Jensen and R.T. Snodgrass, "Temporal Data Management", IEEE Transactions on Knowledge and Data Engineering, Volume 11 Issue 1, January/February 1999, pp. 36-44.
39. K. Jerrigan, “Oracle Total Recall with Oracle Database 11g Release 2”, White paper, Oracle, September 2009.
40. T. Johnston, and R. Weis, "Managing Time in Relational Databases: How to Design, Update and Query Temporal Data", 1st Edition, Morgan Kaufmann Publication.
41. J.R. Haritsa, M.J. Carey and M. Livny, “Earliest Deadline Scheduling for Real-Time Database Systems”, Real-Time Systems Symposium, 1991, pp. 232-242.
42. B. Kao and H. Garcia-Molina, “An Overview of Real-time Database Systems”, In Real Time-computing, W. A. Halang and A. D. Stoyenko, Eds.: SpringerVerlag, 1994.
43. B. Kao, K. Y. Lam, B. Adelberg, R. Cheng, and T. Lee, "Updates and View Maintenance in Soft Real-Time Database Systems", eighth international conference on Information and knowledge management (CIKM '99), pp. 300-307.
44. K. Kulkarni, and J.-E. Michels, “Temporal features in SQL: 2011”, ACM SIGMOD Record, Volume 41, Issue 3, September 2012, pp. 34-43.
45. V. S. Lai, J.-P. Kuilboer, and J. L. Guynes, "Temporal databases: model design and commercialization prospects", ACM SIGMIS Database, Volume 25 Issue 3, August 1994, pp. 6-18.
46. W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan, "Muppet: MapReduceStyle Processing of Fast Data", VLDB Endowment, Volume 5, Issue 12, August 2012, pp. 1814-1825.
47. J. Lawyer, and and S. Chowdhury, "Best practices in data warehousing to support business initiatives and needs", 37th Annual Hawaii International Conference on System Sciences, 2004, pp. 1-9.
48. P. Li, X. L. Dong, A. Maurino, and D. Srivastava, "Linking temporal records", VLDB Endowment, Volume 4, Issue 11, 2011, pp. 956–967.
49. K. J. Lin, and S.H. Son, "Real-Time Databases: Characteristics and Issues", First IEEE Workshop on Object-Oriented Real-Time Dependable Systems, 1994, pp. 113-116.
50. J. Lindstrom, "Real Time Database Systems", Wiley Encyclopedia of Computer Science and Engineering, 2008.
51. D.R. McCarthy, and U. Dayal, "The Architecture Of An Active Data Base Management System", ACM SIGMOD international conference on Management of data (SIGMOD '89), Volume 18, Issue 2, June 1989, pp. 215-224.

52. R.A. Morris, and L. Khatib, "Entities and Relations for Historical Relational Databases", Temporal Representation and Reasoning (TIME '97), 1997, pp. 180-186.
53. S. B. Navathe, "Evolution of data modeling for databases", Communications of the ACM - Special issue on analysis and modeling in software development, Volume 35 Issue 9, Sept. 1992, pp. 112-123.
54. G. Ozsoyoglu, and R. T. Snodgrass, "Temporal and real-time databases: a survey", IEEE Transactions of Knowledge and Data Engineering, Volume 7, Issue 4, 1995, pp. 513– 532.
55. N. W. Paton and O. Diaz, "Active Database Systems", ACM Computing Surveys (CSUR), Volume 31, Issue 1, March 1999, pp. 63-103.
56. P. Picouet, and V. Vianu, "Semantics and expressiveness issues in active databases", fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS '95), May 1995, pp. 126-138.
57. K. Ramamritham, R. Sivasankaran, J. A. Stankovic, D. T. Towsley, and M. Xiong, "Integrating Temporal, Real-Time, and Active Databases", ACM SIGMOD Record, Volume 25, Issue 1, March 1996, pp. 8-12.
58. N. Rescher, "Temporal Logic", 1971, Springer-Verlag publications.
59. S. Rizvi, "Complex Event Processing Beyond Active Databases: Streams and Uncertainties", Master's thesis, EECS Department, University of California, Berkeley, December 2005.
60. D. B. Robins, "Complex Event Processing", CSEP 504, 2010.
61. G. Sannik, and F. Daniels, "Enabling the Temporal Data Warehouse", White paper, Teradata, September 2010.
62. C. Saracco, M. Nicola, and L. Gandhi, "A matter of time: Temporal data management in DB2 10", Technical report, IBM, April 2012.
63. P. Pin-Shan, "The entity-relationship model—toward a unified view of data", ACM Transactions on Database Systems (TODS), Volume 1 Issue 1, March 1976, pp. 9-36.
64. H. A. Schmid, J. R. Swenson, "On the semantics of the relational data model", ACM SIGMOD international conference on Management of data, 1975, pp. 211-223.
65. A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database System Concepts", 5th Edition, McGraw-Hill Publication, August 2005.
66. M. Singhal, "Issues and Approaches to Design of Real-Time Database Systems", ACM SIGMOD Record - Special Issue on Real-Time Database Systems, Volume 17, Issue 1, March 1988, pp. 19-33.
67. A. P. Sistla and O. Wolfson, "Temporal triggers in active databases", IEEE Transactions on Knowledge and Data Engineering (TKDE), Volume 7, Issue 3, 1995, pp. 471-486.
68. P. Sleat, and P. Osmon, "A methodology for real-time database system construction", Third International Conference on Software Engineering for Real Time Systems, 1991, pp. 233-238.
69. R. Snodgrass, "The temporal query language TQuel", ACM Transactions on Database Systems (TODS), Volume 12, Issue 2, June 1987, pp. 247-298.
70. R. Snodgrass, "Temporal databases status and research directions", ACM SIGMOD Record - Directions for future database research & development, Volume 19, Issue 4, December 1990, pp. 83-89.

71. R.T. Snodgrass, M.H. Böhlen, C.S. Jensen, and A. Steiner, "Adding Valid Time to SQL/Temporal", ANSI X3H2-96-501r2, ISO/IEC JTC1/SC21/WG3 DBL-MAD-146r2, Nov. 1996.
72. R.T. Snodgrass, M.H. Böhlen, C.S. Jensen, and A. Steiner, "Adding Transaction Time to SQL/Temporal", ISO/IEC JTC1/SC21/WG3 DBL MCI-143, ANSI X3H2-96-152r, 1996 (b)
73. S.H. Son, "Real-Time Database Systems: Present and Future", Second International Workshop on Real-Time Computing Systems and Applications, 1995, pp. 50-52.
74. M. Stonebraker, AND G. Held, "Networks, hierarchies, and relations in database management systems," ACM Pacific, April 1975, pp. 1-9
75. S. Suhothayan, K. Gajasinghe, I. L. Narangoda, S. Chaturanga, S. Perera, and V. Nanayakkara, "Siddhi: a second look at complex event processing architectures", ACM workshop on Gateway computing environments (GCE '11), November 2011, pp. 43-50.
76. N. Tryfona, F. Busborg, and J.G.B. Christiansen, "starER: A Conceptual Model for Data Warehouse Design", 2nd ACM international workshop on Data warehousing and OLAP (DOLAP '99), 1999, pp. 3-8.
77. O. Ulusoy, "Current research on real-time databases", ACM SIGMOD Record, Volume 21, Issue 4, December 1992 , pp. 16-21.
78. O. Ulusoy, "Research issues in real-time database systems", Information Sciences 87, 1995, pp. 123-151.
79. S. Urman, "Oracle9i PL/SQL Programming", 2nd Edition, McGraw-Hill Osborne Media Publication.
80. P. Vassiliadis, and T. Sellis, "A survey of logical models for OLAP databases", ACM SIGMOD Record, Volume 28, Issue 4, December 1999, pp. 64-69.
81. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for ETL processes", 5th ACM international workshop on Data Warehousing and OLAP (DOLAP '02), 2002, pp. 14–21.
82. G. M. Weiss, and B. Davison, "Data Mining", In H. Bidgoli (ed.), Handbook of Technology Management, John Wiley and Sons, Volume 2, 2010, pp. 542-555.
83. J. Widom, "Research problems in data warehousing", fourth international conference on Information and knowledge management (CIKM '95), 1995, pp. 25-30.
84. R. Wijaya, and B. Pudjoatmodjo, "An overview and implementation of extraction-transformation-loading (ETL) process in data warehouse (Case study: Department of agriculture)", 3rd International Conference on Information and Communication Technology (ICoICT), 2015, pp. 70-74.
85. M. C. Wu, A. P. Buchmann, "Research Issues in Data Warehousing", BTW '97, 1997.
86. E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams", ACM SIGMOD international conference on Management of data (SIGMOD '06), 2006, pp. 407-418.
87. C. Zang,and Y. Fan, "Complex event processing in enterprise information systems based on RFID", Enterprise Information Systems, Volume 1, Issue 1, February 2007, pp. 3-23.
88. ACT-NET Consortium, "The active database management system manifesto: a rulebase of ADBMS features", ACM SIGMOD Record, Volume 25, Issue 3, September 1996, pp. 40-49.

89. Oracle Database Online Documentation 11g Release 2 (11.2) - Database Data Warehousing Guide, Part V Data Warehouse Performance, 22 SQL for Analysis and Reporting.
(https://docs.oracle.com/cd/E11882_01/server.112/e25554/analysis.htm#DWHSG8659)

APPENDIX-A

Database Terminologies:

Primary key Constraint: This constraint ensures that the values appearing in a column of the table are all unique and not null.

Foreign Key Constraint: This constraint ensures that the values appearing in a column have already appeared in some other foreign column.

Check Constraint: This constraint ensures that the values appearing in a column of the table are part of some predefined group, list or range of values.

Not-Null Constraint: This constraint ensures that the values appearing in a column of the table are not null.

Unique Constraint: This constraint ensures that the values appearing in a column of the table are all unique; i.e., no duplication allowed.

Domain Integrity: Ensures that all the values in a column fall within a defined or existing set of values, termed as valid values.

Referential Integrity: Ensures that the values in a column should already be referenced but in a foreign column.

Entity Integrity: Ensures that each row in table is uniquely identified.

Normalized forms: A process where the tables are decomposed to eliminate redundancy, that too without loss of any data.

Functional Dependency: Column Y is functionally dependent on column X ($X \rightarrow Y$), if and only if each value in X is associated with one value in Y.

Insert Anomaly: An insert anomaly occurs when certain column information cannot be inserted into the table without the presence of other column information.

Update Anomaly: An update anomaly occurs when only partial instances of the same data are being updated, and not all.

Delete Anomaly: A delete anomaly occurs when certain column(s) are lost because of the deletion of other column.

APPENDIX-B

DDL for tables with Primary Keys:
In this paper, the ‘create-statements’ from Example-1-to-9 did not have primary keys as we wanted to showcase the prohibition of insert statements by the cellular constraints. Below, we enumerate the schemas that were used to simulate the preemption of update statements. The results were pretty much identical to the output in Experiment and Results section, so we avoid repeating them here again.

Example-11:

Window Constraint:

```
create table class_roster (
student_id varchar(10),
course varchar(10),
grade varchar(2),
semester varchar(10),
primary key (student_id),
foreign key (semester) references semester_info(semester)
/*object(student_id),
window(grade)    :    [semester_info.end_date,    0000-01-07
00:00:00]*/
);
```

Example-12:

Aggregate Constraint:

```
create table transactions (
customer_id varchar(10),
branch varchar(10),
withdrawal_amount double,
primary key (customer_id)
/*
object(customer_id),
aggregate(withdrawal_amount) : [+, <=, 400],
window(withdrawal_amount) : [0000-01-01 00:00:00, now()]
*/
);
```

Example-13:

Compulsory Constraint:

```
create table commuters (
user_name varchar(20),
buspass_id varchar(20),
busstop_location varchar(50),
balance double,
primary key (user_name),
foreign key (buspass_id) references bus_pass(pass_id)
/*
object(user_name),
compulsory(busstop_location) : [1],
window(busstop_location)  :  [bus_pass.valid_from,0000-01-31
00:00:00]
*/
);
```

Example-14:

Interval Constraint:

```
Create table H1_applicants (
applicant_id varchar(10),
decision varchar(20),
primary key (applicant_id)
/*
object(applicant_id),
frequency(decision) : [3],
interval(decision) :[1,0000-12-31 00:00:00],
window(decision) : [object, 0002-05-31 00:00:00]
*/
);
```

Example-15:

Frequency Constraint:

```
Create table GRE_attempts (
student_id varchar(10),
location varchar(20),
score double,
primary key (student_id)
/*
object(student_id),
interval(score) :[1,0000-1-21 00:00:00],
frequency(score) : [5],
Window(score) : [0000-12-31 00:00:00, now()]
*/
);
```

Example-16:

Penalty Constraint:

```
create table signin_info (
user_id varchar(20),
pass_word varchar(20),
last_loggedin datetime,
primary key (user_id),
foreign key (user_id) references credentials(users)
/*
object(user_id),
penalty(pass_word) : [credentials.passphrase, =, 5 , 0000-00-
00 23:59:59],
window(pass_word) : [0000-00-00 12:00:00, now()]
 */
);
```

Example-17:

Range Constraint:

```
create table card_users (
customer_id varchar(20),
expenditure_amount double,
shop varchar(20),
card_id bigint,
primary key(customer_id),
foreign key (card_id) references cards(card_number)
/*
object(customer_id),
Range(expenditure_amount) : [("0-300", 0000-12-31 00:00:00),
("0-2000",  0002-12-31  00:00:00),  ("0-5000",  0004-12-31
00:00:00), ("0-15000", 0099-12-31 00:00:00)],
window(expenditure_amount) : [cards.issue_date, now()]
*/
);
```

Example-18:

Sequence Constraint:

```
Create table progress_report (
student_id varchar(10),
graduation_year varchar(10),
gpa double,
primary key (student_id),
foreign key (student_id) references student_info(id)
/*
object(student_id),
sequence(graduation_year)  :  [Freshman,  Sophomore,  Junior,
Senior],
interval(graduation_year) :[1,0000-12-31 00:00:00],
Window(graduation_year) : [object, 0005-12-31 00:00:00]
*/
);
```

Example-19:

Threshold Constraint:

```
Create table transcripts (
student_id varchar(10),
course_number varchar(10),
grade varchar(2),
primary key (student_id)
/*
object(student_id),
Threshold(grade) : [("C|C+|C-|D|D+|D-", 2), ("F", 1)],
Window(grade) : [0002-12-31 00:00:00, now()]
*/
);
```