International Journal for Research in Applied Science & Engineering Technology (IJRASET)

# Implementing user-defined Integrity Constraint in MYSQL

Deepika[#1], Mr. Anil Arora[#2]
[1]*M. Tech. Scholar,* [2]*Assistant Professor, Department of Computer Science & Engineering*
*Gateway Institute of Engineering & Technology (GIET), Sonepat*

**Abstract—** ***The methods with which we can maintain correctness of any specific table in data base management system are Integrity constraints. The general idea of this paper is to investigate and observe user-defined integrity constraint handling or integrity control in relational database management systems. Every business applications run among the rules which are defined previously, these rules are also valid and appropriate to business data and they should not be violated. MySQL presented a unique and special feature called data constraint or integrity constraint that was applied with the formation data structure. Constraints are business rules, which are imposed on data being stored in a table. Due to their carelessness or lack of knowledge Integrity control deals with the avoidance of semantic errors made by users; Integrity rules used by the integrity control subsystem so that it verify the database and operations on the database. In today's database systems and functions for implementing user-defined integrity constraints triggers are being applying in a variety of important ways. The most important use of triggers is still to implement a variety of integrity constraints taken by the application, and we have disagreed that additional uses of triggers than one might think are in fact maintaining constraints of one sort or another.***
***Keywords— Trigger, Trigger Signal, Event, Business rule Constraint.***

## I. INTRODUCTION

Data constraints allowed by MySQL to be join with table column level via sql syntax before data will be check for integrity. Once upon a time data constraints which are element of a table column construct, then the data being entered into a table column against the data constraints checks by the oracle database engine. If the data surpass this check, data is stored in the table column, or else the data will be discarded. The entire record will be discarded if a single column of the record which is being entered into table be unsuccessful to satisfy a constraint and it will not be stored in the table.

There are numerous mechanisms given by SQL-92 (and subsequent SQL-99) for identifying integrity constraints standard. K*eys*, *Not-null constraints*, and *Referential integrity* —these all are the most common types of constraints, each one have their own syntax and enforcement mechanisms. In association to triggers is the reality that referential integrity constraints can be individual with particular actions to be taken upon violations, such as *cascaded delete* or *set null*. The more common *check* constraints are combined with a given database table. To classify conditions, SQL-like syntax is applied that must embrace for each tupple of the table, and on inserts and updates to the table the conditions will be checked.

Although trigger was not included because it is not supported like element of the SQL-92 standard, in the early to mid- 1990's, triggers were already supported by some products. The SQL-99 standard has widespread exposure of triggers, and today triggers are supported by all most important relational DBMS vendors. Whenever an identified *event* arises, a trigger is activated, frequently an *insert*, *delete*, or *update* on a particular table. Once activated, a non-compulsory identified *condition* is verified and, if the condition is true (or omitted), an *action* is executed. There are a many of essential details to the specification and execution semantics of triggers, only a small number of of which are enclosed here.

## II. CONSTRAINTS OR TRIGGERS

Constraints as well as event-driven application logic should be support by an active database system (DBMS). Conversely, whenever possible declarative constraints should be used in lieu of triggers. Initial, to implement one declarative constraint there is need of several triggers; then, in all cases the system has no technique to assurance the validity of the constraint. Consider utilities loaded by the database in which before it can be accessed, the database ensures the declarative constraints against the loaded data. While triggers are also applied for transitional constraints and also for event-driven application logic, there is no method to establish which triggers should be checked. When constraints and triggers are added to a database with pre-existing data then these actions also applies.

Knowledge of declarative specifications can be used by the database engine to optimally estimate constraints. For example, during referential integrity enforcement by reducing the isolation level to cursor stability, concurrency control hot spots can be

avoided. Tool vendors may also take benefit of declarative constraints for previously-checking entries at the client (e.g. a mobile laptop) prior to sending data to the server.

The most systems provides declarative constructs and described in SQL92 only carried a little, although useful, The acceptable states of the value of the database classified by the set of static constraints, e.g. salary > commission + hourly Wage * hours Worked. The transitional constraints do not hold by them that control the technique in which the database value can transition from one state to the next e.g. salary increases must be less than 10%. Event-driven invocation of application and business logic also do not support by them. Therefore, to enhance the declarative constraint constructs and to capture application specific business rules we need triggers. During database modifications, triggers offered a procedural ways for describing implicit activity. To support event-driven invocation of application logic we need to use triggers, which can be strongly integrated with a modification and executed by identifying a trigger on the base table of the modification in the database engine.

For declarative constraints triggers should not be used as a substitute. Therefore, the constraint logic with transitional constraints, data conditioning capabilities, exception handling, and user defined repairing actions extended by them.

In short, there are benefits to use both declarative constraints and procedural triggers and both types of constructs are available in many profitable systems. Therefore, it is very important to describe and realize the relations of constraints and triggers.

## III. DECLARATIVE CONSTRAINTS

Declarative constraints in SQL, check constraints and referential constraints are the two forms of static constraints. Even if declaratively specified, every constraint involves a set of events for which the constraint is checked and if the constraint is not satisfied (or, more accurately, evaluates to false) an action to be taken. The system verifies it against the existing data, when a static constraint is identified and once identified; it assured that the constraint is always satisfied. The system confirmed all declarative static constraints earlier than the loaded data can be accessed and following load utilities are used. To validate Input data, we typically used a check constraint; in a given table this condition is true for every row.

Generally, including multi-table assertions, a check constraint can be any SQL condition. Since such statements are extremely costly to support, this feature is practically restricted to the special case key constraints and one variable aggregate-free check constraints, i.e., value constraints, in most existing products. These constraints allow the definition of uniqueness constraints, value restrictions and intra row value checks on a single table.

In object-oriented extensions of SQL, check constraints can contain functions which are deterministic that applied to the values and they don't have side effect. When the constraint is described then value constraints need only be evaluated, when any table on which the constraint is defined or after special load utilities are run, is modified by an UPDATE or INSERT statement. In addition, when any table referenced in the condition is modified then multi-table assertions may also require to be checked. For any updated or inserted rows, if the constraint estimates to be false then the statement that caused the modification2 is discarded and any changes made by the statement are undone.

A referential integrity (RI) constraint establishes a contact between a set of columns allocated as a foreign key and a unique key such that the non-null values of the foreign key must also appear as values in the unique key. The constraint's child table is foreign key's table and the unique key's table is the constraint's parent table. Note that the parent table and child table both can be the same, referred to as a self-referencing RI constraint. Like check constraints, when the constraint is defined, RI constraints must also be checked and then special load utilities are run.

Conversely, there are two tables involved and four modifications that cause constraint estimation: (a) insertions into the child table, (b) updates a foreign key column, (c) deletions from the parent table, and (d) updates of a unique key column, these are four modifications. The statement that caused the modification is rejected when modifications to the child table violate the constraint, and several changes made by the statement are undone. When the parent is modified by a DELETE or UPDATE statement, then the action to be taken is one of a set of previously-defined actions that is denoted with a delete rule or update rule, respectively, when the constraint is defined. This previously-defined set which reject the violating statement3 includes: NO ACTION and RESTRICT; Which sets the nullable columns of the foreign key of any child rows is SET NULL that match the modified parent rows to null; Which sets the foreign key columns for any child rows is SET DEFAULT that match the modified parent to their default values; and which deletes any matching child rows is CASCADE.

## IV. MYSQL TRIGGERS

Basically triggers are procedural, in contrast to declarative constraints. The event managing the execution of a trigger is explicitly specified in its definition. UPDATE, DELETE or INSERT statement are the triggering events applied to a base table. An optional column list can be specified in the case of updates to further restrict the set of update events which activate the trigger. If the trigger is executed before or after its event, the trigger also has an activation time that specifies and it has a granularity that describes how many times the trigger is executed for the event. Before-triggers execute before their event and

are extremely useful for conditioning of the input data before modifications are applied to the database and the relevant constraints evaluated. After-triggers execute after their event and are typically used to embed application logic, which typically runs after the modification completes.

The granularity of a trigger can be indicated as either FOR EACH ROW or FOR EACH STATEMENT, referred to as row-level and statement-level triggers respectively. The trigger is executed once for each row affected by the event when the event of a row-level trigger occurs. If no rows are affected, then the trigger is never be evaluated. However, the trigger is executed once for each row affected by the event a statement-level, trigger is executed exactly once whenever its event occurs, even if the event does not modify any rows. Like to constraints, both row-level and statement-level after triggers must (appear to) execute only after the triggering event completes executing; before-triggers must appear to execute entirely before the triggering event's modifications are applied. Note that when the trigger executes the granularity does not dictate.

However, if they or any constraints that must be evaluated that need access to either the base table of the triggering event or any table that is modified by the trigger or the constraints, then row-level triggers can only be processed in a set-oriented fashion. E.g., if a row-level update trigger on table T accesses the average of a column in T, it must either be computed entirely before or after the application of any modifications to T the average must not be computed in the middle of the triggering update; then each trigger has access to the before-transition values and after-transition values of the event through the declaration of transition variables and transition tables. If both OLD and NEW transition variables are declared as above, then the new and old values of a given row can be compared. For example, if the trigger's table is EMP, and EMP has column salary, then we can check if the modification was a raise by comparing old salary with new salary. Using the keywords NEW-TABLE and OLD-TABLE Transition tables are similarly declared. If NT is declared as a new transition table, it is a virtual table containing the values of all modified rows immediately after the modification is applied. Similarly, if OT is declared as an old transition table, it is a virtual table containing the values of all modified rows before the modification is applied.

For each type of trigger, not all types of transition variables and transition table references are valid. In particular, triggers defined on insert events can only see new values and triggers defined on delete events can only see old values while triggers defined on update events can see both old and new values Furthermore, transition variables are only accessible at the granularity of one row, and hence, can only be referenced by row-level triggers. Both statement-level and row-level before triggers participate in a fix point computation of the transition tables; during the computation of this fix point, inherently, the entire content of the transition tables is not yet computed. Hence, before-triggers cannot reference transition tables. However, both row level and statement-level after-triggers can reference transition tables. Such references allow row-level after-triggers to compare a single transition row value with aggregations on the transition tables, e.g., how the new salary in a given row compares with total impact of the salary increase.

## V. TRIGGER EXAMPLES

Triggers are basically database objects that are *attached* to a table, and are only *fired* when an *INSERT*, *UPDATE* or *DELETE* occurs. This means that it specifies a particular action to take place whenever a given event takes place on a particular object.

A trigger can be created by using following syntax:
Create or Replace Trigger <Trigger-name>
{Before/After} {Insert/Delete/Update} on <Table-nm>
For Each Row [When <Condition>]
Declare
<Declaration of Variables & Constants>;
Begin
<SQL & PL/SQL Statements>;
End;

**Example1:** In our first example we create a trigger that will create backup copy of all the records which are deleted or updated from the product table in the new prodbackup table. The example results are shown in figure 1, 2 & 3 below.

Volume 3 Issue VI, June 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

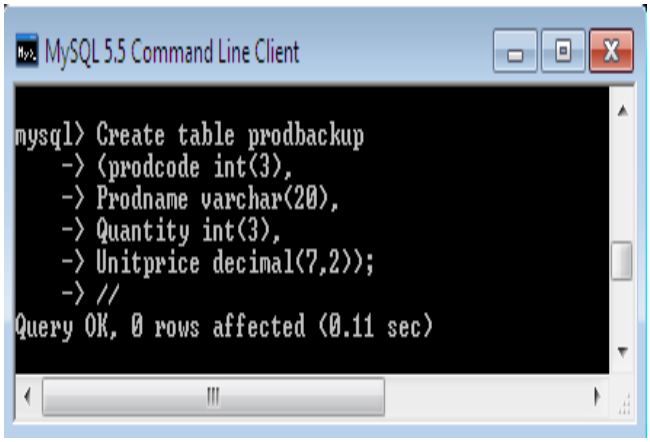


Figure 1: Creating prodbackup table

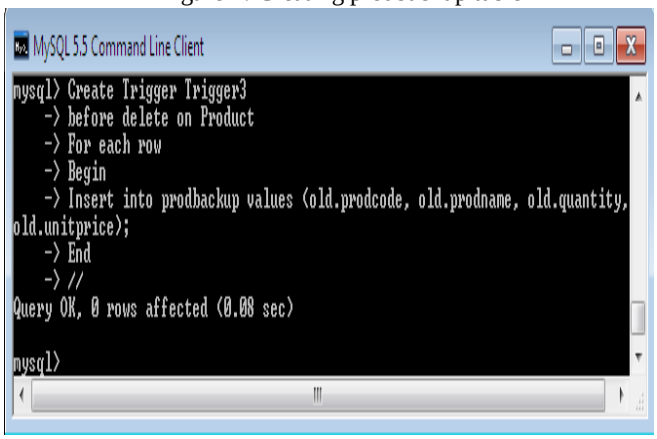


Figure 2: Creating trigger for backup the product table

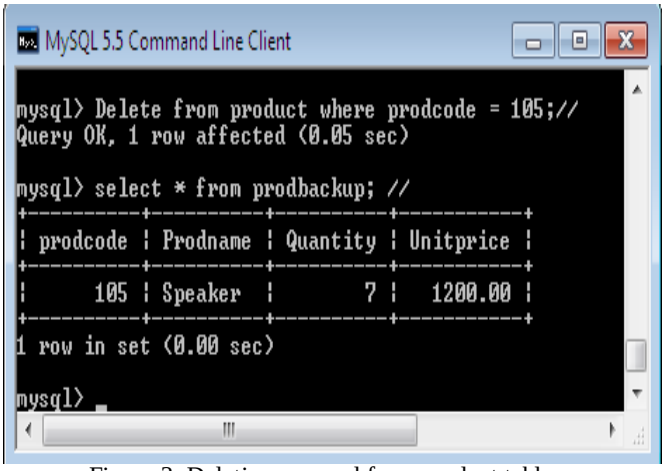


Figure 3: Deleting a record from product table

The product table is updated with 4 records remaining & product with prodcode 105 is permanently deleted from the table. The deleted record is now transferring to the prodbackup table due to effect of trigger. We can view the deleted record from prodbackup table as shown in figure 3 above.

**Example2:** In our next example we create a trigger that will abort any insert operation on product table where quantity entered is negative. By default negative quantity is inserted in product table as shown in figure 4 below.

Volume 3 Issue VI, June 2015
ISSN: 2321-9653
International Journal for Research in Applied Science & Engineering Technology (IJRASET)

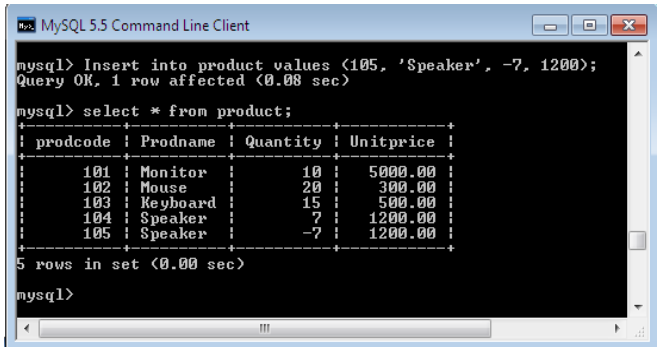


Figure 4: Listing of product table with negative quantity entered

The example results after creating trigger are shown in figure 5 & 6 below.

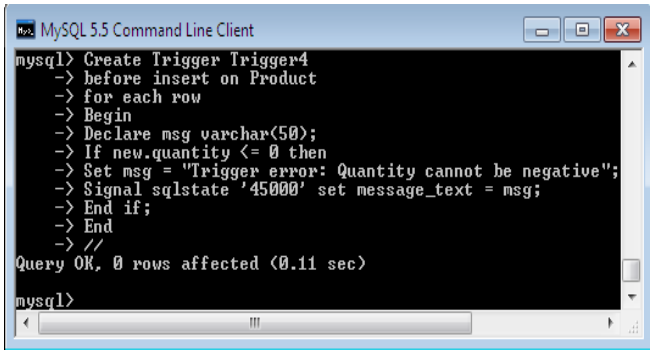


Figure 5: Creating trigger for preventing negative quantity

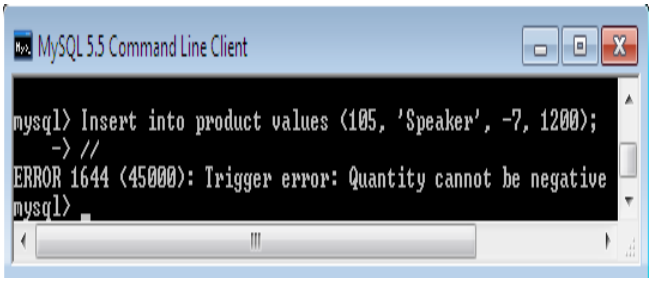


Figure 6: Result of trigger 5

Example3: In our next example we create a trigger that will abort any update operation on sales table where quantity of ordered product is decreased. The example results are shown in figure 7 & 8 below.

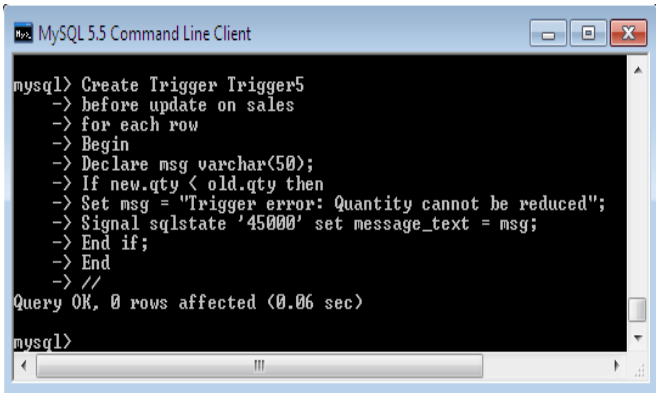


Figure 7: Creating a trigger to abort update operation

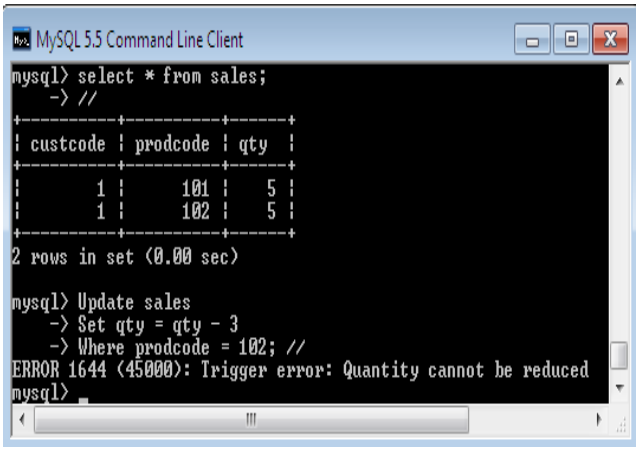


Figure 8: Trigger error message for decreasing quantity

## VI. CONCLUSIONS

Declarative constraints and triggers are two essential features that have been introduced to support user requirements in relational DBMSs. Given the differing expressive powers of declarative constraints and triggers, support for both is required for today's applications. The semantics of the interaction of triggers and declarative constraints must be carefully defined to avoid inconsistent execution and to provide users a comprehensive model for understanding such interaction. From the above we can conclude that **MYSQL Data Constraints** are wide conceptualized. Only data, which satisfies the conditions (rules) set, should be stored for future analysis. If the data gathered fails to satisfy the conditions (rules) set, it must be rejected. This ensures that the data stored in a table will be valid and have integrity. Thus, we can say that data constraints are provides integrity and data level security.

## REFERENCES

[1] R. Agrawal, R. Cochrane, and B. Lindsay. On maintaining priorities in a production rule system. In Proc. of the 17th hat. Conf. on Very Large Data Bases, pages 479-487, Barcelona (Catalonia, Spain), September 1991
[2] L. Brownston, R. Farrell, E. Kant, and N. Martin. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley, Reading, Massachusetts, 1985.
[3] S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In Proc. of the 16th ht. Conf. on Very Large Data Bases, pages 566-577, Brisbane, Australia, August 1990.
[4] R. Cochrane. Issues in Integrating Active Rules Into Database Systems. Ph.D. &sserta%on, University of Maryland, Department of'computer Science, College Park, MD, 1992.
[5] R. Cochrane and N. Mattos. ISO-ANSI SQL3 Change Proposal, ISO/IEC JTCl/SC21/WG3 DBL LHR-89, X3H2-95-458, An Execution Model for After Triggers, November 1995.
[6] Database Programming and Design. The 1996 Business Rules Summit, February 1996.
[7] M. Lenzerini. Data integration: A theoretical perspective. In Proc. of PODS 2002, pages 233–246, 2002.
[8] A. Cal`ı, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. In Proc. of CAiSE 2002, volume 2348 of LNCS, pages 262–279. Springer, 2002.
[9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. Artificial Intelligence, 2012.
[10] Ivan Bayross (2009). The programming language of Oracle. New Delhi: BPB publications
[11] Oracle Corporation. Oracle Workflow Technical Configuration in Oracle Applications Release 11, 2000. Available at http://www.oracle.com/support/library/supportnews/html/workflow.html.
[12] Integrating Triggers and Declarative Constraints in SQL Database Systems Roberta Cochrane Hamid Pirahesh Nelson Mattos Proceedings of the 22nd VLDB Conference Mumbai(Bombay), India, 1996
[13] A Study on Oracle Data Constraints Kalyani M. Raval, Shivkumar H. Trivedi
Volume 1, Issue 1, June 2013 International Journal of Advance Research in Computer Science and Management Studies
[14] Data Integrity [On-Line] Available at: http://docs.oracle.com/cd/E11882_01/server.112/e25789/datain e.htm
[15] Microsoft SQL server, version 7.0. Microsoft Corporation, 1999.
[16] Webopedia: 2005, Data integrity http://www.pcwebopaedia.com/TERM/d/data_integrity.html