Volume 5, Issue 6, June 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Techniques for Performing User-Defined Integrity Constraint

Ritu Raj*, **Kirti bhatia**
SKITM, Ladrawan,
India

Abstract—Integrity constraints are the techniques with which we can maintain correctness of any specific table in data base management system. This paper gives an overview of research regarding user-defined integrity control or integrity constraint handling in relational database management systems. All business applications run with predefined rules, these rules are also applicable to business data and they should not be violated. Oracle provides a special feature call data constraint or integrity constraint that was applied at the time of creation of data structure. Business rules, which are enforced on data being stored in a table, are called constraints. Integrity control deals with the prevention of semantic errors made by users due to their carelessness or lack of knowledge; the integrity control subsystem uses integrity rules to verify the database and operations on the database. Triggers are being used in a variety of significant ways in today's database systems and applications for implementing user-defined integrity constraints. The primary use of triggers is still to enforce various integrity constraints driven by the application, and we have argued that more uses of triggers than one might think are in fact maintaining constraints of one sort or another.

Keywords— Data Constraint, Trigger, Business rule Constraint.

I. INTRODUCTION

Oracle permits data constraints to be attached to table column level via sql syntax that check data for integrity earlier stage. Once data constraints are part of a table column construct, the oracle database engine checks the data being entered into a table column against the data constraints. If the data pass this check, it stored in the table column, else the data are rejected. Even if a single column of the record being entered into table fails a constraint and the entire record is rejected and not stored in the table.

The SQL-92 (and subsequent SQL-99) standard provides several mechanisms for specifying integrity constraints. The most common kinds of constraints—keys, non-null constraints, and referential integrity—each have their own syntax and enforcement mechanisms. Of interest in relationship to triggers is the fact that referential integrity constraints can be specified with particular actions to be taken upon violations, such as cascaded delete or set null. The more general check constraints are associated with a given database table. SQL-like syntax is used to specify conditions that must hold for each tuple of the table, and the conditions are checked on inserts and updates to the table.

Although trigger support was not included as part of the SQL-92 standard, triggers were supported by some products already in the early to mid-1990's. The SQL-99 standard has extensive coverage of triggers, and today all major relational DBMS vendors have some support for triggers. A trigger is activated whenever a specified event occurs, usually an insert, delete, or update on a particular table. Once activated, an optional specified condition is checked and, if the condition is true (or omitted), an action is executed. There are a number of important details to the specification and execution semantics of triggers, only a few of which are covered here.

II. CONSTRAINTS OR TRIGGERS

An active database system (DBMS) should support constraints as well as event-driven application logic. However, declarative constraints should be used in lieu of triggers whenever possible. First, several triggers are required to enforce one declarative constraint; even then, the system has no way of guaranteeing the validity of the constraint in all cases. Consider the database load utilities in which the database checks the declarative constraints against the loaded data before it can be accessed. There is no way to determine which triggers should be checked since triggers are also used for transitional constraints and for event-driven application logic. This behavior also applies when constraints and triggers are added to a database with pre-existing data.

The database engine can use knowledge of declarative specifications to optimally evaluate constraints. For example, concurrency control hot spots can be avoided during referential integrity enforcement by reducing the isolation level to cursor stability. Guaranteeing the presumed semantic intent of triggers, in general, requires an isolation level of repeatable read. Furthermore, semantic query optimization is not possible if the declarative semantics are hidden in triggers. Tool vendors may also take advantage of declarative constraints for pre-checking entries at the client (e.g. a mobile laptop) before sending data to the server.

The declarative constructs provided by most systems and defined in SQL92 only support a small, albeit useful, set of static constraints that define the acceptable states of the value of the database, e.g. salary > commission + hourlyWage * hoursWorked. They do not support transitional constraints that restrict the way in which the database value can transition from one state to the next e.g. salary increases must be less than 10%. They also do not support event-driven invocation of application and business logic. Hence, triggers are required to enhance the declarative constraint constructs and to capture application specific business rules. Triggers provide a procedural means for defining implicit activity during database modifications. They are used to support event-driven invocation of application logic, which can be tightly integrated with a modification and executed in the database engine by specifying a trigger on the base table of the modification.

Triggers should not be used as a replacement for declarative constraints. However, they extend the constraint logic with transitional constraints; data conditioning capabilities, exception handling, and user defined repairing actions.

In summary, there are advantages to using both declarative constraints and procedural triggers, and both types of constructs are available in many commercial systems. It is not feasible to expect applications providers to either migrate their existing applications to use only triggers or partition the tables in their database according to the type of constraints and triggers that are required. It is therefore imperative to define and understand the interaction of constraints and triggers.

III. DECLARATIVE CONSTRAINTS

Declarative constraints in SQL include two forms of static constraints: check constraints and referential constraints. Although declaratively specified, each constraint implies a set of events for which the constraint is checked and>n action to be taken if the constraint is not satisfied (or, more accurately, evaluates to false). When a static constraint is defined, the system verifies it against the existing data, and once defined, guarantees that the constraint is always satisfied. After load utilities are used, the system verifies all declarative static constraints before the loaded data can be accessed. A check constraint, typically used to validate input data, is a condition that is true for every row in a given table. In general, a check constraint can be any SQL condition, including multi-table assertions. Since such assertions are extremely expensive 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 applied to the values as long as the functions are deterministic and do not have side-effects. Value constraints need only be evaluated when the constraint is defined, after special load utilities are run, or when any table on which the constraint is defined is modified by an UPDATE or INSERT statement. In addition, multi-table assertions may also need to be checked when any table referenced in the condition is modified. If the constraint evaluates to false for any updated or inserted rows then the statement that caused the modification2 is rejected and any changes made by the statement are undone.

A referential integrity (RI) constraint establishes a relationship between a set of columns designated 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 foreign key's table is the constraint's child table and the unique key's table is the constraint's parent table. Note that the parent table and child table can be the same, referred to as a self-referencing RI constraint. Like check constraints, RI constraints must also be checked when the constraint is defined and after special load utilities are run. However, since there are two tables involved in the constraint, there are four modifications that cause constraint evaluation: (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. Whenever modifications to the child table violate the constraint, the statement that caused the modification is rejected and any changes made by the statement are undone. However, the action to be taken when the parent is modified by a DELETE or UPDATE statement is one of a set of pre-defined actions that is specified with a delete rule or update rule, respectively, when the constraint is defined. This pre-defined set includes: NO ACTION and RESTRICT, which reject the violating statement3; SET NULL, which sets the nullable columns of the foreign key of any child rows that match the modified parent rows to null; SET DEFAULT, which sets the foreign key columns for any child rows that match the modified parent to their default values; and CASCADE, which deletes any matching child rows.

IV. SQL TRIGGERS

In contrast to declarative constraints, triggers are procedural. The event governing the execution of a trigger is explicitly specified in its definition. This triggering event is an UPDATE, DELETE or INSERT statement 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 that activate the trigger. The trigger also has an activation time that specifies if the trigger is executed before or after its event and a granularity that defines 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 specified as either FOR EACH ROW or FOR EACH STATEMENT, referred to as row-level and statement-level triggers respectively. When the event of a row-level trigger occurs, the trigger is executed once for each row affected by the event. If no rows are affected, then the trigger is never evaluated. However, 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 finishes executing; before-triggers must appear to execute entirely before the triggering event's modifications are applied. Note that the granularity does not dictate when the trigger executes. Obviously, there are many cases when row-level triggers can be safely and optimally processed in-flight. However, if they or any constraints that must be evaluated require 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, then the average must not be computed in the middle of the triggering update; it must either be computed entirely before or after the application of any modifications to T. 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. The declaration "referencing NEW as N" defines N as a single row correlation variable that contains the value of a row in the database immediately after the modification is applied. Similarly, "referencing OLD as 0" declares 0 as a single row correlation variable containing the value of the same row in the database before the modification is applied. 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 0. salary with N. salary. Transition tables are similarly declared using keywords NEW-TABLE and OLD-TABLE. 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.

Not all types of transition variables and transition table references are valid for each type of trigger. 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.

Each trigger has an action that is optionally guarded by a condition; the action is only executed when the triggering event occurs and, if specified, the condition is satisfied. The condition is a single, unrestricted search condition and the action is a procedure containing a sequence of SQL statements. Both the action and the condition can query the transition variables and tables as well as the current value of the database, and these queries can invoke user-defined functions. Since before-triggers appear to execute entirely before the event occurs, any queries of the database in their conditions or actions must appear to read the database state prior to any modifications made by the event. Similarly, the conditions and actions of all after-triggers must appear to read the database state after all of the event's modifications are applied. If the after-trigger needs to query the state of the database prior to the event, then it can reconstruct it using the transition tables. For example, if T is modified by the triggering event, NT is the new transition table and OT is the old transition table, then the state of T prior to the event can be approximated by (T/NT) U OT.

V. TRIGGER EXAMPLES & RESULTS

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.

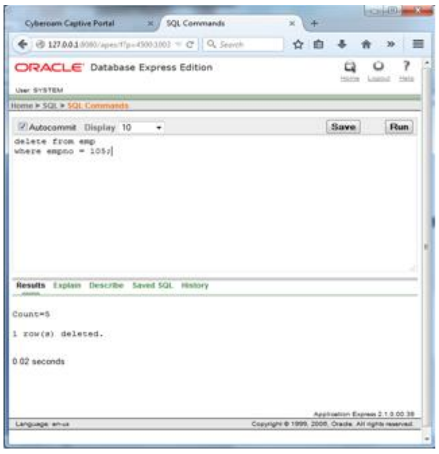
A trigger can be created by using following syntax:

```
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: The following trigger creates a back up emp table empbackup table after each delete or update operation on emp table

```
Create or Replace Trigger Backup
  After delete or update on Emp
  For each row
  Begin
  Insert into Empbackup values
  (:old.empno, :old.ename, :old.job, :old.sal);
  End;
```

Figure 1 & 2 below shows the result of above trigger

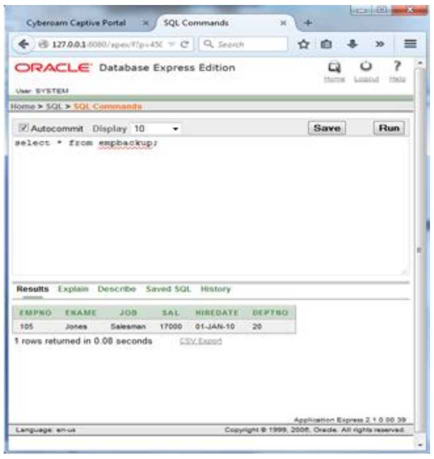


```
delete from emp
where empno = 105;
```

```
Count=5

1 row(s) deleted.

0.02 seconds
```



```
select * from empbackup;
```

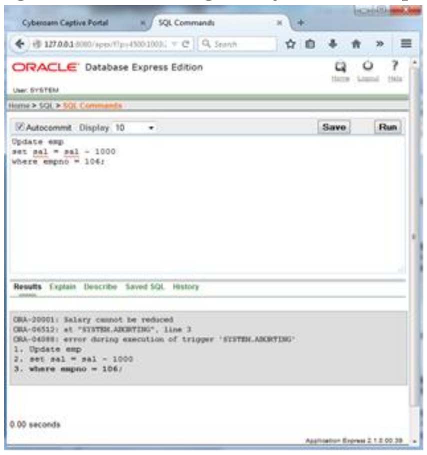
EMPNO	ENAME	JOB	SAL	HIREDATE	DEPTNO
105	Jones	Salesman	17000	01-JAN-10	20

1 rows returned in 0.08 seconds

Example2: Define a trigger to abort each update statement on emp table if salary is decreased.

```
Create or Replace trigger Aborting
  Before update on emp
  For each row
  Begin
  If  :new.sal < :old.sal Then
Raise_application_error(-20001, ‘salary can’t be reduced’);
  End If;
  End;
```

Figure 3 below shows the error message after decreasing salary for the employee.



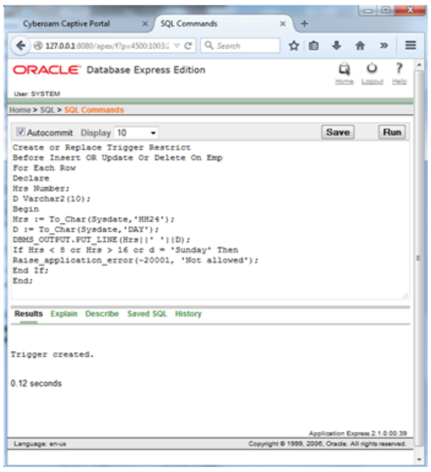
```
Update emp
set sal = sal - 1000
where empno = 106;
```

```
ORA-20001: Salary cannot be reduced
ORA-06512: at "SYSTEM.ABORTING", line 3
ORA-04088: error during execution of trigger 'SYSTEM.ABORTING'
1. Update emp
2. set sal = sal - 1000
3. where empno = 106;

0.00 seconds
```

Figure 3: Trigger error message for decreasing salary

Example3: In our last example we create a trigger Restrict that will abort any update, delete or insert operations on emp table when it is Sunday or for Monday to Saturday it is before 8:0 AM or after 5:00 PM. The example results are shown in figure 4 & 5 below. There are no restrictions on viewing the record

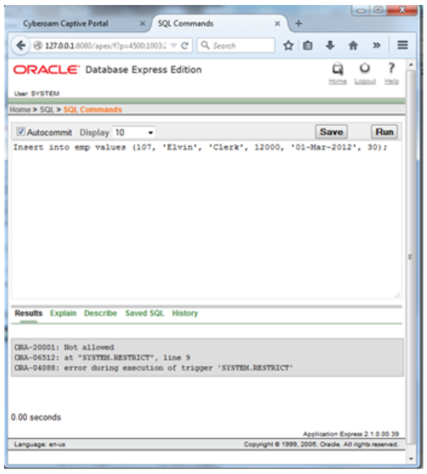


```
Create or Replace Trigger Restrict
Before Insert OR Update Or Delete On Emp
For Each Row
Declare
Hrs Number;
D Varchar2(10);
Begin
Hrs := To_Char(Sysdate,'HH24');
D := To_Char(Sysdate,'DAY');
DBMS_OUTPUT.PUT_LINE(Hrs||' '||D);
If Hrs < 8 or Hrs > 16 or d = 'Sunday' Then
Raise_application_error(-20001, 'Not allowed');
End If;
End;
```

```
Trigger created.

0.12 seconds
```

Cyberoam Captive Portal
SQL Commands
127.0.0.1:8080/apex/f?p=4500:1003:
ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.



```
Insert into emp values (107, 'Elvin', 'Clerk', 12000, '01-Mar-2012', 30);
```

```
ORA-20001: Not allowed
ORA-06512: at "SYSTEM.RESTRICT", line 9
ORA-04088: error during execution of trigger 'SYSTEM.RESTRICT'

0.00 seconds
```

Cyberoam Captive Portal
SQL Commands
127.0.0.1:8080/apex/f?p=4500:1003:
ORACLE Database Express Edition
User: SYSTEM
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History
Application Express 2.1.0.00.39
Language: en-us
Copyright © 1999, 2006, Oracle. All rights reserved.

VI. CONCLUSIONS

Declarative constraints andtriggers are two essential features that have been introducedto support user requirements in relationalDBMSs. Given the differing expressive powers ofdeclarative constraints and triggers, support for bothis required for today's applications.The semantics of the interaction of triggers anddeclarative constraints must be carefully defined toavoid inconsistent execution and to provide users acomprehensive model for understanding such interaction.From the above we can conclude that **Oracle 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] Oracle [1]: 2005, Oracle Company Profile Page http://company.monsterindia.com/oraclein/

[16] Webopedia: 2005, Data integrity http://www.pcwebopaedia.com/TERM/d/data_integrity.html