Database Integrity Mechanism between OLTP and Offline Data

Muhammad Salman[1], Nafees Ur Rehman[2], and Muhammad Shahid[3]

[1] Graduate School of Engineering Sciences & Information Technology,
Hamdard University Karachi-75400, Pakistan
[2] Department of Computer & Information Sciences,
University of Konstanz, P.O. Box. D-188, 78457 Germany
[3] Department of Computer & Information Technology,
Pakistan International Airline Karachi-75200, Pakistan
{msalman_74,muhammadshahid}@hotmail.com,
nafees.rehman@uni-konstanz.de

Abstract. This paper describes integrity mechanism between OLTPs and offline data. Normally every RDBMS supports five Integrity Constraints (ICs) namely primary key or composite key, unique key, foreign key, not null and check constraints. Online database integrity is achieved through these five ICs. However, as per the retention period data is backed up and removed from the OLTPs for space and performance efficiency. But there is no standardized protocol on keeping integrity between offline data and data present in the OLTPs. Therefore, we present a solution to address the problem of offline data integrity by keeping a representative set of purged data & ICs in the online database to ensure data integrity between OLTPs and offline data. We further support our proposed solution with the help of two types of integrity tests i.e., sufficient and complete test.

Keywords: Integrity Constraints, OLTPs Performance, Offline Data.

1 Introduction

The word integrity is derived from Latin adjective integer (whole, complete). In this context, integrity is the sense of wholeness. Integrity is a concept of consistency of actions, values, methods, measures, principles, expectation and outcomes [1]. Similarly database integrity means keeping your data accurate, consistent and valid. It is achieved by preventing accidental or deliberate but unauthorized insertion, modification or destruction of data in a database. Online database integrity is achieved by integrity constraints. So in order to improve OLTPs performance we have to implement certain retention policy and offline backup will be taken as per the retention policy. And then that particular set of data will be removed from OLTPs. Unfortunately, not that much work has been done to maintain offline data integrity in such a scenario.

To explain the problem, consider a database schema containing tables namely Customer, Product, Order and etc. The size of data in Order table, let's assume, is 50GB. Therefore, we have to implement certain retention policy and offline backup will be taken as per the retention policy. Finally, this particular set of data will be

J.-S. Pan, S.-M. Chen, N.T. Nguyen (Eds.): ACIIDS 2012, Part II, LNAI 7197, pp. 371–380, 2012.

deleted from online Order table. However, any DML operation on Order table would challenge the data integrity of the offline data as there is no communication with the running OLTPs and the backed up offline data.

2 Related Work

Let us present a summary of few related works in this section.

In [2], an integrity subsystem for RDBMS has been discussed, and it shows how integrity is different from the other important area of security, consistency, and reliability. The integrity subsystem reacts as guards that prevent database against unauthorized insertion, modification or destruction of data in a database, and data accurately stored in database.

[3] Introduces schema integration process. Schema integration is an important step in the database integration, and deal with integrity constraints problems. The knowledge about the correspondences between integrity constraints and extensional assertions must be known in the schema integration process, and handle all issues of database integrity at schema level in the distributed environment.

[4] Explains the active integrity constraints (AICs) with consistent database. An active integrity constraint is a special constraint which is maintained database integrity for all DML operations. DML operation would be generated constraints violation while data already available in database. Otherwise, DML operations are allowed.

[5] Maintains database integrity during updating process has been explained. It means that, multiple users access online database concurrently. If one user performs update query on particular record set during this process other users cannot perform update query on that particular record set until first user performs commit or rollback transaction on that particular record set.

[6] This paper describes database integrity patterns. Actually vendors of RDBMS (Relational Database Management System) are quite high, but number of available solutions to active database integrity is limited. The solutions to avoid RDBMS integrity problems can be formalized as a pattern language. Integrity patterns were defined in the form of constraints, locking, and transactions.

[7] Explains the process of database integrity for synthetic vision system (SVS). Synthetic vision system provides information to cockpit crew with whether a heads down display and heads up display. HDD and HUP containing aircraft state guidance, navigation information, and weather condition. So therefore all information depends upon database integrity. It is very essential for the safety.

In [8], a solution for maintenance of database integrity in the replication environment is presented. The previous version of this paper presented a method for replication through 1-copy serialability. The new version of this paper is suggested replication process through snapshot isolation (SI). SI controls all issues of database integrity.SI takes copy of data from active database and only updates that data which is changed in the active database. This method is improved performance of replication process and maintained database integrity. Most of modern all RDBMSs support SI through materialized view (stored snapshot of data).

In [9], authors discussed schema integration in the entity relationship model (ERM). Schema integration is achieved through three steps. That is, first, finds out ambiguities of integrity constraints in the integration process and solved; second, schemas are merged; third restructure integrated schema according to specific goal.

The above mentioned research papers describes solution of various online database integrity problems and contributed further enhancement in this direction. None of these directly deal with the problem of online database integrity with offline data. Now we present a solution to this problem in the following sections.

3 Method to Achieve Integrity

To achieve offline data integrity, the two following approaches have been proposed:

- Offline data integrity at record level
- Offline data integrity on batch mode

3.1 Offline Data Integrity at Record Level

Offline data integrity at record level is achieved through RDBMS trigger. In this technique, as per retention policy (i.e., when there are large amounts of data)) we took the online database backup and stored representative subset of backup data that consists of primary key or composite key, unique key and foreign key values and finally removed backup data from the OLTPs. Afterwards, before a new record is inserted or unique values of the existing records are updated, this is first verified for any integrity violation against the representative subset of backed up data stored in the online database system. In case of a violation, insertion or updation is rejected and integrity validation exception is raised. Otherwise, DML operations in the online database are allowed. Similarly in the foreign key case, when we insert new record into tables then first it is checked against online database subsets (foreign key) for integrity violations. And similarly if there is a violation it raises an exception and DML operation is disallowed. Otherwise, the change takes effect and DML operations are performed in the online database system.

3.2 Offline Data Integrity on Batch Mode

Integrity verification for each tiny change results in comparatively huge process and effect the performance of OLTPs. To control this problem, we suggest a method where we temporarily allow data in the OLTPs and ignore integrity violation just to provide for better performance in the OLTPs. Instead of checking integrity for each row, we do it in batch mode. It can be done on the off-hours of the day or at a particular time as deemed appropriate. The logic goes into a stored procedure to check integrity in batch mode. In this method, as per retention policy (i.e., when there are large amounts of data) we took the online database backup and stored representative subset of backup data that consists of primary key or composite key, unique key and foreign key values and finally removed backup data from the database. In this case, insertion of new

records into tables or updating of the existing values is allowed. Later, at the appropriate time, the stored procedure is invoked and is executed in a manner ignoring values with no conflicts but reporting records that are in integrity violation.

3.3 Logical Design of Integrity Mechanism between Online System and Offline Data

The Fig 1 explains the process of our solution. As per retention policy we take the backup of online database and store representative subset of backup data in the online database and finally remove backup data from the database. When a user performs insert or update query on the online database then on first step the query is validated by representative subset of data that is offline data. If a record or tuple primary or composite key, unique key values are matched with a representative subset of data then a transaction is aborted from the database. Otherwise, query is allowed on the online database. Similarly if the new updated value for foreign key in OLTP matches a foreign key value in the representative subset of data then the transaction is allowed. Otherwise, constraint violation exception is raised.

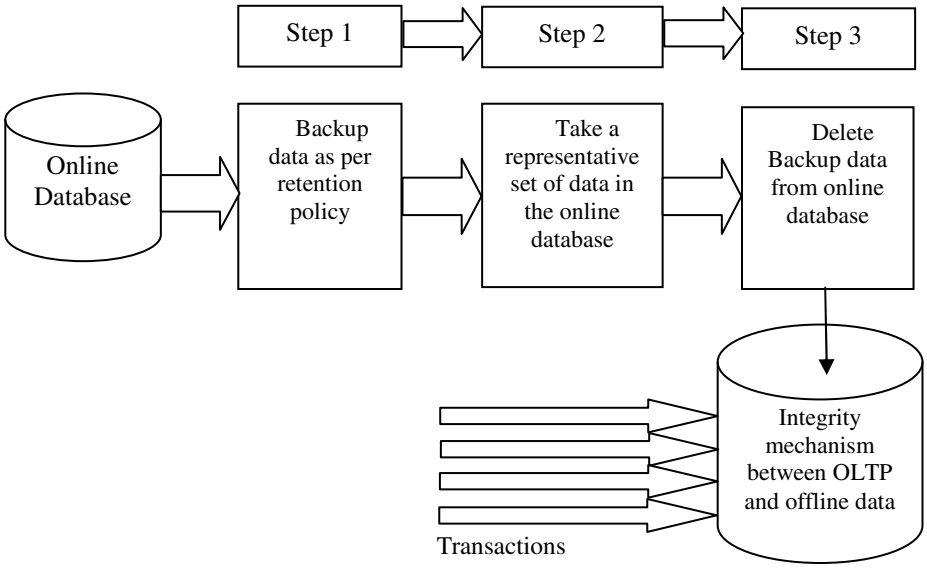

Fig. 1. Integrity mechanism between OLTPs & offline data

3.4 Program Code

3.4.1 Trigger Level Checking Integrity Constraints and Tests

In the trigger code, first we select all keys values (primary key, unique Key, foreign key) from offline data set which is available in the online database. When new transaction is inserted or updated then at time check value against the selected primary and unique keys values. If value is already available then exception is

generated otherwise transaction is performed. Similarly in the foreign key case, new transaction foreign key value is checked against the selected foreign key values. If current transaction value is matched then transaction is performed otherwise, transaction generated the exception.

```
Input: List of integrity constraints primary key PK,
unique key UK, foreign key FK from Representative set of
data (Offline Data)
Output: Valid transaction performed or aborts transaction
(if available in the offline data)
Begin
 Store the begin time in the Tri_Result table
 For each j in Off-Transaction loop
  Begin
  Invoke sufficient test
  If current_value.PK is equal to j.PK and
  Current_value.UK is equal to j.UK then
  Action: Abort current transaction from the database;
  Else
  Action: Complete test is performed & DML 'T' Done;
  End.
 For each k in Off-Transaction loop
  Begin
  Invoke sufficient test
  If current_value.FK is equal to k.FK then
  Action: DML 'T' Done;
  Else
  Action: Complete test is performed & Abort Current
  Transaction;
  End.
  Action: Store the end time in the Tri_Result table;
End.
```

3.4.2 Procedure Level Checking Integrity Constraints and Tests

In the procedure code, first we selected complete all keys values (primary, unique and foreign) on daily base from the online database. Second we selected complete all keys values from the offline dataset which is available in the online database. If daily base online database values matched with offline represented set of data then it report to an integrity violation.

```
Input: List of integrity constraints primary key PK,
unique key UK, foreign key FK from Representative set of
data (Offline Data)
Output: Valid transaction performed or aborts transaction
(if available in the offline data)
```

```
Begin
 Store the begin time in the Proc_Result table
 For each i in online-transaction loop
  For each j in off-transaction loop
  Begin
  Invoke sufficient test
  If i.PK is equal to j.PK and i.Uk is equal to j.UK then
  Action: Abort Current transaction from the database;
  Else
  Action: Complete test is performed & DML transaction
  Done;
  End.
 End.
 For each i in online-transaction loop
  For each j in off-transaction loop
  Begin
  Invoke sufficient test
  If i.FK is equal to j.FK then
  Action: DML performed on database;
  Else
  Action: Complete test is performed & Abort
  Transaction, T;
  End.
 End.
  Action: Store the end time in the Proc_Result table;
End.
```

4 Experimental Result Proof through McCarroll

Our approach has been developed to manage RDBMS integrity between OLTPs and offline data McCarroll introduces different level of integrity tests namely sufficient test, necessary test and complete test [10]. In the sufficient test, when update or insert operation is satisfied with respect to the integrity constraint and thus DML operation is performed on the table. In the necessary test, when insert or update operation is not satisfied with respect to the integrity constraint and thus DML operation abort the transaction from the database. Complete test have both sufficient and necessary tests properties.

When a user issues an update or insert query again the online database, the integrity constraints for the offline data are tested for any violation. If there is a violation, i.e. the update or inserted field value is in conflict with integrity constraints of offline data, then an exception is raised and transaction is aborted. Otherwise, update or insert query is executed and is allowed to make changes in the database. Referring to the below mention Table-2, ($\forall i \exists u \exists v \exists w \exists x \exists y \exists z$order ($i$, u, v, w, x, y, z)) is an example of sufficient test, which verify the existence of *Ord_id* values in the *Order_offline* table that is representative subset of data whereas i, u, v, w, x, y and z are constants.

When insert or update query is executed on an *Order* table. If *Ord_id* attribute value exists in an *Order_offline* table, then we conclude that the sufficient test and initial constraint, *I1*, is satisfied. If there is no record available with respect to an *Ord_id* attribute of *Order_offline* table, then further verification is performed by complete test and record is inserted or updated into *Order* table. Similarly we performed same test for *Card_no* attribute (which is unique key) in an *Order* table.

In the case of ($\forall u \exists i \exists v \exists w \exists x \exists y \exists z$ order (*u, i, v, w, x, y, z)*) is an example of sufficient test, which verified an existence of *Cid* attribute value in an *Order_offline* table, that is representative subset of data. When insert or update query is executed on an *Order* table. If *Cid attribute* value exists in an Order_*offline* table, then we conclude that the sufficient test property or initial constraint, *I2*, is satisfied. If there is no record available with respect to *Cid* attribute of *Order_offline* table, then further verification performed by complete test. Similarly we performed same test for *Pid* attribute (which is foreign key) in an Order table.

Every record is verified against the integrity constraint values between online database (*Order* table) and offline data (*Order_offline* table).

Table 1. Schematic Algebra

Schema: *Customer* (cid, cname, add, loc); *Product*(pid, pname, price, unit_price); *Order* (ord_id, cid, pid, card_no, ord_date, sup_date, tot_price); *Order_offline* (ord_id, cid, pid, card_no) Integrity Constraints:
Every record of *Ord_id* attribute in an *Order_offline* table exists in an *Order* table
I_1: $(\forall i \forall b \forall c \forall d \exists u \exists v \exists w \exists x \exists y \exists z)(Order\_offline(i, b, c, d) \rightarrow Order(i, \text{u}, \text{v}, \text{w}, x, y, z))$
Every record of *Cid* attribute in an *Order_offline* table exists in an *Order* table
I_2: $(\forall a \forall i \forall c \forall d \exists u \exists v \exists w \exists x \exists y \exists z)(Order\_offline(a, i, c, d) \rightarrow Order(\text{u}, \text{i}, \text{v}, \text{w}, x, y, z))$
Every record of *Pid* attribute in an *Order_offline* table exists in an *order* table
I_3: $(\forall a \forall b \forall i \forall d \exists u \exists v \exists w \exists x \exists y \exists z)(Order\_offline(a, b, i, d) \rightarrow Order(\text{u}, \text{v}, \text{i}, \text{w}, x, y, z))$
Every record of *Card_no* attribute in an *Order_offline* table exists in an *order* table
I_4: $(\forall a \forall b \forall c \forall i \exists u \exists v \exists w \exists x \exists y \exists z)(Order\_offline(a, b, c, i) \rightarrow Order(\text{u}, \text{v}, \text{w}, \text{i}, x, y, z))$

Table 2. Integrity Association

I	Insert and Update Template	Integrity Test
I_1	insert(*order*(*i, u, v, w, x ,y ,z*))	1. $(\forall i \exists x \exists y \exists z)(order\_offline(i, b, c, d))$ [1]
		2. $(\forall i \exists u \exists v \exists w \exists x \exists y \exists z)(order(i, u, v, w, x, y, z))$[2]
	update(*order*(*i, u, v, w ,x, y, z*))	3. $(\forall b \forall c \forall d)(\neg order\_offline(i, b, c, d))$ [1]
I_2	insert(*order*(*u, i, v, w, x, y, z*))	1. $(\forall x \exists i \exists y \exists z)(order\_offline(a, i, c, d))$ [1]
		2. $(\forall u \exists i \exists v \exists w \exists x \exists y \exists z)(order(u, i, v, w, x, y ,z))$[2]
	update(*order*(*u, i, v, w, x ,y, z*))	3. $(\forall a \forall c \forall d)(\neg order\_offline(a, i, c, d))$ [1]
I_3	insert(*order*(*u*, v, *i, w, x, y, z*))	1. $(\forall x \exists y \exists i \exists z)(order\_offline(a, b, i, d))$ [1]
		2. $(\forall u \exists v \exists i \exists w \exists x \exists y \exists z)(order(u, v, i, w, x, y, z))$[2]
	update(*order*(*u, v, i, w, x, y, z*))	3. $(\forall a \forall b \forall d)(\neg order\_offline(a, b, i, d))$ [1]
I_4	insert(order(u, v, w, i, y, z))	1. $(\forall w \exists x \exists y \exists i)(order\_offline(a, b, c, i))$ [1]
		2. $(\forall u \exists v \exists w \exists i \exists x \exists y \exists z)(order(u, v, w, i, x, y, z))$[2]
	update(*order*(*u, v, w, i, x, y, z*))	3. $(\forall a \forall b \forall c)(\neg order\_offline(a, b, c, i))$ [1]

5 Experimental Results

The system on which we ran our experiment is running Window XP, with 2GB RAM and 2GHz processor. This experimental result is done between online database (Order table) and offline data (*Order_offline table*).

5.1 Offline Data Integrity at Record Level

In Fig 2, shows experimental result for offline data integrity using Oracle 10g, SQL Server 2000 and IBM DB2 9.7 trigger. In this experiment, online database stored the representative set of data that is five million, seven million and up to 10 million and each valid transaction took process time is 1.600, 2.100 and up to 3.500 seconds for Oracle 10g in the online database. Similarly in the SQL Server 2000 case, each valid transaction took process time is two, three, four and five seconds. In the IBM DB2, each valid transaction took process time is 1.600, 2.200 and up to 3.400 seconds.

Table 3. Record level RDBMS Time Statistics

S.No	Offline Data Set	Oracle Time (Second)	SQL Server Time (Second)	IBM DB2 Time (Second)
1	5000000	1.600	2	1.600
2	7000000	2.100	3	2.200
3	9000000	2.400	4	2.500
4	11000000	3.500	5	3.400

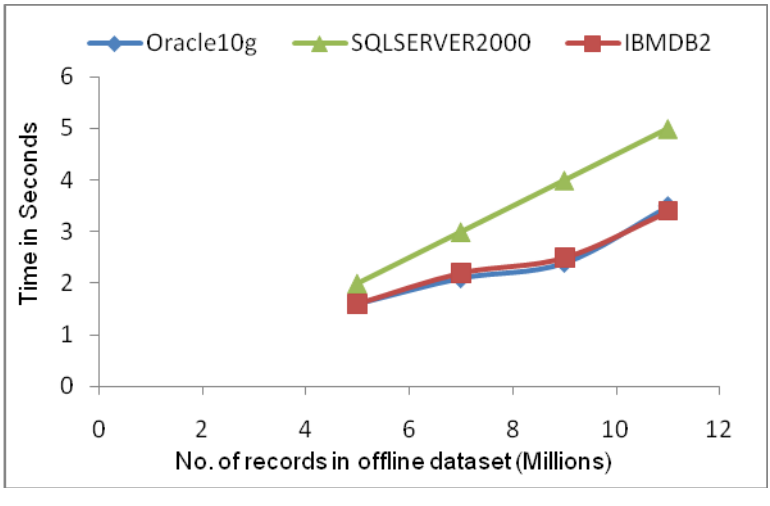

Fig. 2. Offline Data Integrity through Trigger

5.2 Offline Data Integrity on Batch Mode

This method improved OLTP performance as compare to record level approach because it is execute on batch mode. In each experiment, we stored a representative subset of data in the online database that is four millions, six millions, eight millions and 10millions and daily OLTPs (online transaction processing system) transactions on a table that is two thousand, four thousands, six thousand and ten thousand. Therefore, we calculated the stored procedure execution timing in different tools (Oracle10g, SQL Server 2000 and IBM DB2 9.7).

Table 4. Batch mode RDBMS Time Statistics

S.No	Daily OLTP Transaction	Oracle Time (Second)	SQL Server Time (Second)	IBM DB2 Time (Second)
1	2000	4	5	3
2	4000	4.100	6	4
3	6000	4.200	8	4.500
4	8000	4.400	12	5.100
5	10000	4.500	18	6.500

In Fig 3, shows experimental result for offline data integrity using Oracle 10g, SQL Server 2000 and IBM DB2 procedure. In this experiment, Oracle stored procedure execution took 4, 4.2 and up to 4.5 seconds against an OLTP daily transaction on the database that is 2000, 6000 and up to 10000 and reporting records that are in integrity violation. Similarly in the SQL Server 2000 case, stored procedure execution took five, six and up to eighteen seconds and reporting records that are in integrity violation. In the IBM DB2 case, stored procedure took execution three, four and up to 6 seconds and reporting records that are in integrity violation.

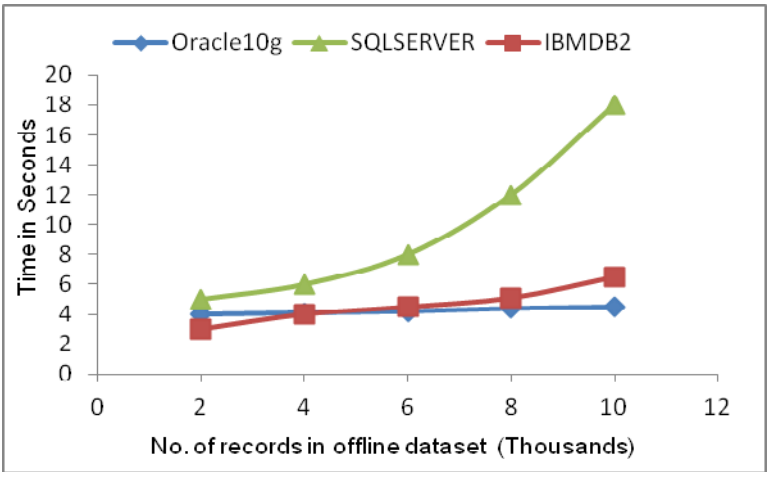

Fig. 3. Offline Data Integrity through procedure

6 Conclusions

We were able to introduce an integrity checking mechanisms between two disconnected systems, i.e. online database systems (OLTPs) and Offline data. We justified the offline data integrity through *sufficient* and *complete* tests. The paper presented results of various experiments conducted to support our work. These experiments were carried out on the major market products available like Oracle (10g) Microsoft SQL Server (2000) and IBM DB2 (9.7). The integrity mechanism presented was tested at two levels in the database systems i.e., at record level and in batch level. The integrity verification at level is implemented using a trigger and checks the integrity of the new record with the offline data. In case of a violation, the record insertion is rejected. The integrity verification in batch mode tolerates any integrity inconsistencies of the updated or newly inserted data for temporary time period to avoid performance degradation of OLTP while it is running the business. However, at an appropriate time, all updations and insertions can be checked for any integrity violation with the offline data.

References

1. The American Heritage Dictionary of the English Language, El-shaddai, p. 2000 (2009)
2. Eswaran, K., Chamberlin, D.: Functional Specifications of a Subsystem for Database Integrity. ACM (1975)
3. Türker, C.: Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration. In: Eder, J., Rozman, I., Welzer, T. (eds.) ADBIS 1999. LNCS, vol. 1691, pp. 31–45. Springer, Heidelberg (1999)
4. Caroprese, L., Greco, S., Zumpano, E.: Active Integrity Constraints for Database Consistency Maintenance. IEEE Transactions on Knowledge and Data Engineering, 1042–1058 (2009)
5. Mayol, E., Teniente, E.: A Survey of Current Methods for Integrity Constraint Maintenance and View Updating. Advances in Conceptual Modeling, 67–73 (1999)
6. Mircea Petrescu, A., Rotaru, O.P.: A Database Integrity Pattern Language. ACM (2008)
7. Uut de Haag, M., Sayre, J.: Terrain Database Integrity Monitoring for Synthetic Vision Systems. IEEE Transactions on Aerospace and Electronic Systems, 386–406 (2005)
8. Lin, Y., Kemme, B.: Snapshot Isolation and Integrity Constraints in replicated Databases. ACM (2009)
9. Batini, Carlo.: A Methodology for Data Schema Integration in the Entity Relationship Model. IEEE Transactions on Software Engineering, 650–664 (2009)
10. McCarroll, N.F.: Semantic Integrity Enforcement in Parallel Database Machines PhD Thesis, University of Sheffield, UK (1995)