Principles of Distributed Database Systems

M. Tamer Özsu

Patrick Valduriez

Outline

- Distributed Data Control
  - View management
  - Data security
  - Semantic integrity control

Semantic Integrity Control

Maintain database consistency by enforcing a set of constraints defined on the database.

- Structural constraints
  - Basic semantic properties inherent to a data model e.g., unique key constraint in relational model
- Behavioral constraints
  - Regulate application behavior, e.g., dependencies in the relational model
- Two components
  - Integrity constraint specification
  - Integrity constraint enforcement

Semantic Integrity Control

- Procedural
  - Control embedded in each application program
- Declarative
  - Assertions in predicate calculus
  - Easy to define constraints
  - Definition of database consistency clear
  - But inefficient to check assertions for each update
    - Limit the search space
    - Decrease the number of data accesses/assertion
    - Preventive strategies
    - Checking at compile time

Constraint Specification Language

Predefined constraints

specify the more common constraints of the relational model

- Not-null attribute

  ENO **NOT NULL IN** EMP

- Unique key

  (ENO, PNO) **UNIQUE IN** ASG

- Foreign key

  A key in a relation *R* is a foreign key if it is a primary key of another relation *S* and the existence of any of its values in *R* is dependent upon the existence of the same value in *S*

  PNO **IN** ASG **REFERENCES** PNO **IN** PROJ

- Functional dependency

  ENO **IN** EMP **DETERMINES** ENAME

Constraint Specification Language

Precompiled constraints

Express preconditions that must be satisfied by all tuples in a relation for a given update type

(INSERT, DELETE, MODIFY)

NEW - ranges over new tuples to be inserted

OLD - ranges over old tuples to be deleted

General Form

CHECK ON <relation> [**WHEN** <update type>] <qualification>

Constraint Specification Language

Precompiled constraints

- Domain constraint

  CHECK ON PROJ (BUDGET≥500000 **AND** BUDGET≤1000000)

- Domain constraint on deletion

  CHECK ON PROJ **WHEN DELETE** (BUDGET = 0)

- Transition constraint

  CHECK ON PROJ (**NEW**.BUDGET > **OLD**.BUDGET **AND**
  NEW.PNO = **OLD**.PNO)

Constraint Specification Language

1. General constraints

Constraints that must always be true. Formulae of tuple relational calculus where all variables are quantified.

General Form

CHECK ON <variable>:<relation>,(<qualification>)

variable: A variable representing a tuple (record) within a relation (table).

relation: The name of the table or relation the variable belongs to.

qualification: The condition that must always be true.

Constraint Specification Language

2. Functional dependency

CHECK ON e1:EMP, e2:EMP

(e1.ENAME = e2.ENAME **IF** e1.ENO = e2.ENO)

This constraint applies to the EMP table, where e1 and e2 represent two records from the same table. It states that **if two records have the same Employee Number (ENO), then their Employee Name (ENAME) must also be the same**.

This enforces a **functional dependency**: ENO → ENAME, meaning an employee number must uniquely determine an employee name.

In a **distributed database**, enforcing this constraint requires ensuring data consistency even when employee records are stored across multiple nodes.

Constraint Specification Language

3. Constraint with aggregate function

```
CHECK ON g:ASG, j:PROJ
    (SUM(g.DUR WHERE g.PNO = j.PNO) < 100 IF
        j.PNAME = "CAD/CAM")
```

a.Data Distribution:

The ASG and PROJ tables might be stored on different nodes in a distributed database system.

Querying both tables together (WHERE g.PNO = j.PNO) requires efficient **distributed joins**.

b.Aggregation Across Nodes:

Computing SUM(g.DUR) means that **partial results may need to be computed on different nodes** before being aggregated into a final sum.

A distributed database may use **MapReduce-style operations** to compute partial sums locally before combining them globally.

c. Consistency and Transaction Control:

If updates to ASG.DUR occur frequently, the system must ensure that constraint checks remain **consistent across nodes**.

Some distributed databases might use **eventual consistency**, while others may enforce strict constraints using **global transactions**.

Constraint Specification Language

Challenges in Distributed Databases

- **Efficient Query Execution:** Ensuring that the sum calculation does not require excessive data shuffling between nodes.
- **Constraint Enforcement at Scale:** Enforcing the constraint in real-time as new ASG.DUR values are inserted or updated.
- **Concurrency Control:** Multiple transactions updating ASG.DUR simultaneously might cause violations if not handled correctly.

Integrity Enforcement

Two methods

- Detection

  Execute update u: $D \rightarrow D_u$

  If D_u is inconsistent then

  if possible: compensate $D_u \rightarrow D_u'$

  else

  undo $D_u \rightarrow D$

- Preventive

  Execute u: $D \rightarrow D_u$ only if D_u will be consistent

  - Determine valid programs
  - Determine valid states

Query Modification

- Preventive
- Add the assertion qualification to the update query
- Only applicable to tuple calculus formulae with universally quantified variables

```
UPDATE  PROJ
SET     BUDGET = BUDGET*1.1
WHERE   PNAME = "CAD/CAM"
```

↓

```
UPDATE  PROJ
SET     BUDGET = BUDGET*1.1
WHERE   PNAME = "CAD/CAM"
AND     NEW.BUDGET ≥ 500000
AND     NEW.BUDGET ≤ 1000000
```

Compiled Assertions

What are Compiled Assertions?

- Compiled assertions define **constraints** that must be enforced whenever a **relation (R) is updated in a certain way (T)**. They are written as **triples (R, T, C)**:
- **R** → The relation (table) affected by the update.
- **T** → The type of update (INSERT, DELETE, MODIFY).
- **C** → The assertion that must hold true based on **differential relations** (changes caused by the update).

Compiled Assertions

Example: Foreign key assertion

$\forall g \in ASG, \exists j \in PROJ : g.PNO = j.PNO$

The basic **foreign key constraint** is:

> **"Every assignment (`g`) in `ASG` must reference an existing project (`j`) in `PROJ`."**

This is formally written as:

$$\forall g \in ASG, \exists j \in PROJ : g.PNO = j.PNO$$

This means that for every row `g` in `ASG`, there must exist a row `j` in `PROJ` where their **project numbers (PNO) match**.

Compiled Assertions

Compiled assertions:

(ASG, **INSERT**, C1), (PROJ, **DELETE**, C2), (PROJ, **MODIFY**, C3)

where

C1: $\forall$**NEW** $\in$ ASG+ $\exists$j $\in$ PROJ: NEW.PNO = j.PNO

C2: $\forall$g $\in$ ASG, $\forall$**OLD** $\in$ $PROJ^-$: g.PNO ≠ **OLD**.PNO

C3: $\forall$g $\in$ ASG, $\forall$**OLD** $\in$ $PROJ^-$ $\exists$**NEW** $\in$ $PROJ^+$:

g.PNO ≠**OLD**.PNO OR **OLD**.PNO = **NEW**.PNO

Compiled Assertions

1. Insert into `ASG` (C1)

Assertion:

$$\forall NEW \in ASG^{+}, \exists j \in PROJ : NEW.PNO = j.PNO$$

🔷 **Explanation:**

- When a new row (`NEW`) is inserted into `ASG`, its `PNO` must already exist in `PROJ`.
- `ASG+` represents the newly inserted records in `ASG`.
- The system must check that for every new assignment, a matching project exists.

Compiled Assertions

2. Delete from `PROJ` (C2)

Assertion:

$$\forall g \in ASG, \forall OLD \in PROJ^{-} : g.PNO \neq OLD.PNO$$

🔷 **Explanation:**

- `PROJ-` represents **deleted records** from `PROJ`.
- When a project is deleted, we must check if there are any assignments (`g`) referencing it in `ASG`.
- If such an assignment exists, deleting the project would leave an orphaned assignment, violating the foreign key constraint.
- The system must **prevent the deletion** or take corrective action (e.g., cascade delete or restrict delete).

Compiled Assertions

3. Modify `PROJ` (C3)

Assertion:

$$\forall g \in ASG, \forall OLD \in PROJ^{-}, \exists NEW \in PROJ^{+} : g.PNO \neq OLD.PNO \text{ OR } OLD.PNO = NEW.PNO$$

◆ **Explanation:**

- `PROJ-` represents **old values** before modification.
- `PROJ+` represents **new values** after modification.
- If a project's `PNO` is being modified, we must ensure that either:
  - No assignments in `ASG` reference the old `PNO`, **or**
  - The old `PNO` still exists in the modified `PROJ`.
- This prevents breaking existing references in `ASG`.

Compiled Assertions

Compiled assertions define **foreign key constraints** across **INSERT, DELETE, and MODIFY** operations.
They ensure **referential integrity** in a database.
Enforcing them in **distributed systems** requires efficient constraint checking, distributed transactions, and possible use of eventual consistency mechanisms.

Differential Relations

Given relation R and update u

R^+ contains tuples inserted by u

R^- contains tuples deleted by u

Type of u

insert R^- empty

delete R^+ empty

modify $R^+ \cup (R - R^-)$

Differential Relations

- **R+ stores inserted tuples**
- **R- stores deleted tuples**
- **Insert:** Only R+ changes (new rows added)
- **Delete:** Only R- changes (rows removed)
- **Modify:** Both R+ and R- change (old rows removed, new rows inserted)

Differential Relations

Algorithm:

Input: Relation *R*, update *u*, compiled assertion C_i

R → A relation (table) in the database.

u → An update operation (INSERT, DELETE, or MODIFY).

Ci → A compiled assertion that needs to be checked.

Steps of the Algorithm

1 **Generate Differential Relations:**

- Identify **R+ (inserted tuples)** and **R– (deleted tuples)** based on the update `u`.

2 **Check for Constraint Violations:**

- Retrieve all tuples from `R+` and `R-` that **do not satisfy** the assertion `Ci`.

3 **Validate the Assertion:**

- If no such violating tuples are found, the assertion holds **(i.e., the database remains consistent)**.

Differential Relations

Example :

u is delete on J. Enforcing (EMP, DELETE, C2) :

retrieve all tuples of EMP^-
into RESULT
where not(C2)

If RESULT = { }, the assertion is verified

💡 **Scenario:**

- An **EMPLOYEE (EMP)** table exists.
- An update `u` **deletes tuples from another table J**.
- A compiled assertion **(EMP, DELETE, C2)** must be enforced.

🛠 **Execution Steps:**

1. Retrieve all tuples in `EMP-` (the deleted rows).
2. Check if any of these tuples violate `C2`.
3. If no tuples violate `C2` (`RESULT = {}`), then the assertion is **valid and enforced**.

Distributed Integrity Control

- Problems:
  - Definition of constraints
    - Consideration for fragments
  - Where to store
    - Replication
    - Non-replicated : fragments
  - Enforcement
    - Minimize costs

Types of Distributed Assertions

- Individual assertions
  - Single relation, single variable
  - Domain constraint
- Set oriented assertions
  - Single relation, multi-variable
    - functional dependency
  - Multi-relation, multi-variable
    - foreign key
- Assertions involving aggregates

Distributed Integrity Control

- Assertion Definition
  - Similar to the centralized techniques
  - Transform the assertions to compiled assertions
- Assertion Storage
  - Individual assertions
    - One relation, only fragments
    - At each fragment site, check for compatibility
    - If compatible, store; otherwise reject
    - If all the sites reject, globally reject
  - Set-oriented assertions
    - Involves joins (between fragments or relations)
    - May be necessary to perform joins to check for compatibility
    - Store if compatible

Distributed Integrity Control

- Assertion Enforcement
  - Where to enforce each assertion depends on
    - Type of assertion
    - Type of update and where update is issued
  - Individual Assertions
    - If update = insert
      - Enforce at the site where the update is issued
    - If update = qualified
      - Send the assertions to all the sites involved
      - Execute the qualification to obtain R^+ and R^-
      - Each site enforces its own assertion
  - Set-oriented Assertions
    - Single relation
      - Similar to individual assertions with qualified updates
    - Multi-relation
      - Move data to perform joins; then send the result to query master site

Conclusion

- Solutions initially designed for centralized systems have been significantly extended for distributed systems
  - Materialized views and group-based discretionary access control
- Semantic integrity control has received less attention and is generally not well supported by distributed DBMS products
- Full data control is more complex and costly in distributed systems
  - Definition and storage of the rules (site selection)
  - Design of enforcement algorithms which minimize communication costs