Making SQL Assertions Work

Toon Koppelaars

Database Development, Oracle Corporation
toon.koppelaars@oracle.com

SQL assertions (SQL-92) promised a generic way to express data-integrity constraints, but remain absent from commercial RDBMSs because efficient, production-grade implementations are hard to achieve. We present a practical, general approach that makes SQL assertions feasible in real systems.

An implementation of SQL assertions must satisfy the following properties:

1. Validation of a SQL assertion should only occur when a DML statement could potentially violate it.
2. Validation should be performed incrementally whenever possible, to avoid checking all rows of all involved tables.
3. Concurrently running transactions may cooperate to violate a SQL assertion; this must be detected, and a serial execution order enforced.

We introduce the concept of a *potentially violating DML statement with respect to a SQL assertion*. For each table reference, we compute at compile-time a polarity and a guard query. The polarity specifies whether inserts or deletes can potentially violate the assertion. The guard query further refines this at run-time: it searches the inserted, deleted, or updated rows for assertion-specific properties that could make the DML statement violating.

When a DML statement is identified as potentially violating, we re-use the guard query and inject it into the SQL assertion's Boolean expression. Several injection strategies are available, depending on the expression's structure and the table reference's position. Through these injections we achieve incremental revalidation.

Our implementation supports the read-committed isolation level, which is the default and most widely used level in Oracle databases. In this isolation level, transactions cannot see uncommitted changes made by others, which can allow two transactions to cooperate in violating a SQL assertion. For example, consider a foreign key constraint where a parent row initially has no child rows:

- Transaction 1 deletes the parent row; validation allows this since no child exists.
- Transaction 2, still able to see the parent, inserts a child row under it.
- Transaction 1 commits.
- Transaction 2 commits.

The result is a child row referencing a non-existent parent. Here, Transactions 1 and 2 cooperated to violate the foreign key. (To prevent this in our foreign key implementation, the insert is blocked until Transaction 1 commits.) Similar scenarios arise for SQL assertions. Our novel, generic locking algorithm detects these situations precisely and enforces a serial execution order only when required for correctness. Unlike traditional approaches, the algorithm never locks rows or tables; instead, it uses *assertion-locking memory objects*.