DATABASDESIGN FÖR INGENJÖRER - 1DL124

Sommar 2005

En introduktionskurs i databassystem

http://user.it.uu.se/~udbl/dbt-sommar05/
alt. http://www.it.uu.se/edu/course/homepage/dbdesign/st05/

Kjell Orsborn
Uppsala Database Laboratory
Department of Information Technology, Uppsala University,
Uppsala, Sweden

UPPSALA
UNIVERSITET

Database Integrity Constraints

(Elmasri/Navathe ch. 5.2 and 9.1)

Kjell Orsborn

Department of Information Technology
Uppsala University, Uppsala, Sweden

UPPSALA
UNIVERSITET

Domain constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.

Domain constraints cont…

- The **check** clause in SQL-92 permits domains to be restricted:
  - Use check clause to ensure that an hourly-wage domain allows only values greater than a specified value.
    create domain *hourly-wage* **numeric**(5,2) **constraint** *value-test* **check**(**value** >= 4.00)
  - The domain hourly-wage is declared to be a decimal number with 5 digits, 2 of which are after the decimal point
  - The domain has a constraint that ensures that the hourly-wage is greater than 4.00.
  - The clause constraint value-test is optional; useful to indicate which constraint an update violated.

Referential integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: if "Perryridge" is a branch name appearing in one of the tuples in the account relation, then there exists a tuple in the branch relation for branch "Perryridge".
- Formal definition
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively.
  - The subset α of R_2 is a *foreign key* referencing K_1 in relation r_1 , if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1]$ = t2[α].
  - Referential integrity constraint: $\Pi_\alpha(r_2) \subseteq \Pi_K(r_1)$

Referential integrity in the E-R model

- Consider relationship set R between entity sets E_1 and E_2.
- The relational schema for R includes the primary keys K_1 of E_1 and K_2 of E_2.
- Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively.
- Weak entity sets are also a source of referential integrity constraints. The relation schema for a weak entity set must include the primary key of the entity set on which it depends.

Database modification

- The following tests must be made in order to preserve the referential integrity constraint: $\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$
- **Insert**. If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$. That is $t_2[\alpha] \in \Pi_{K1}(r_1)$
- **Delete**. If a tuple t_1 is deleted from r_1, the system must compute the set of tuples in r_2 that reference t_1:

  $$\sigma_{\alpha = t1[K1]}(r_2)$$

  If this set is not empty, either the delete command is rejected as an error, or the tuples that reference t_1 must themselves be deleted (*cascading deletions* are possible).

Database modification cont'd

- Update. There are two cases:
  - **Case 1**: If a tuple t_2 is updated in relation r_2 and the update modifies values for the foreign key α, then a test similar to the insert case is made. Let t_2' denote the new value of tuple t_2. The system must ensure that: $t_2'[\alpha] \in \Pi_{K1}(r_1)$
  - **Case 2**: If a tuple t_1 is updated in r_1, and the update modifies values for the primary key (K_1), then a test similar to the delete case is made. The system must compute $\sigma_{\alpha = t1[K1]}(r_2)$ using the old value of t_1 (the value before the update is applied). If this set is not empty, the update may be rejected as an error, or the update may be cascaded to the tuples in the set (*cascading update*), or the tuples in the set may be deleted (*cascading delete*).

Referential integrity in SQL

- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
  – The **primary key** clause of the create table statement includes a list of the attributes that comprise the primary key.
  – The **unique key** clause of the create table statement includes a list of the attributes that comprise a candidate key.
  – The **foreign key** clause of the create table statement includes both a list of the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.

Referential integrity in SQL - example

create table *customer*
(*customer-name* **char**(20) **not null**,
customer-street **char**(30),
customer-city **char**(30),
primary key (*customer-name*))

create table *branch*
(*branch-name* **char**(15) **not null**,
branch-city **char**(30),
assets **integer**,
primary key (*branch-name*))

Referential integrity in SQL - example cont’d

```
create table account
        (account-number char(10) not null,
        branch-name char(15),
        balance integer,
        primary key (account-number),
        foreign key (branch-name) references branch)

create table depositor
        (customer-name char(20) not null,
        account-number char(10) not null,
        primary key (customer-name,account-number),
        foreign key (account-number) references account,
        foreign key (customer-name) references customer)
```

Cascading actions in SQL

```
create table account
        ...
        foreign key (branch-name) references branch
                on delete cascade
                on update cascade,
        ...)
```

- If a tuple in *branch* is deleted (updated), there is a tuple in *account* that will also be deleted (updated), i.e. the delete (update) cascades.

Cascading actions in SQL cont’d

- If there is a chain of foreign-key dependencies across multiple relations, with **on delete cascade** specified for each dependency, a deletion or update at one end of the chain can propagate across the entire chain.
- If a cascading update or delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the *transaction*. As a result, all the changes caused by the transaction and its cascading actions are undone.

Assertions

- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL-92 takes the form:

  create assertion <assertion-name> **check** <predicate>
- When an modification (insert/delete/update)of the db is made, the system tests it for validity. This testing may introduce a significant amount of overhead; hence assertions should be used with great care.

Assertion example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
        (not exists
        (select *
        from branch
        where (select sum( amount)
                     from loan
                     where loan.branch-name =
                                  branch.branch-name) >=
                     (select sum( amount)
                                  from account
                                  where loan.branch-name =
                                               branch.branch-name)))
```

Another assertion example

- Every loan has at least one borrower who maintains an account with a minimum balance of $1000.00.

```
create assertion balance-constraint check
      (not exists
      (select *
      from loan
      where not exists
               (select *
               from borrower, depositor, account
               where loan.loan-number  = borrower.loan-number and
                borrower.customer-name = depositor.customer-name and
                depositor.account-number = account.account-number and
                account.balance >= 1000)))
```

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- The SQL-92 standard does not include triggers, but many implementations support triggers. SQL:99 has specified triggers.

Trigger example

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - setting the account balance to zero
  - creating a loan in the amount of the overdraft
  - giving this loan a loan number identical to the account number of the overdrawn account
- New values after an update are represented by the keyword **new** and old values by the keyword **old**.
- The condition for executing the trigger is a check to see if the *balance* after an update to the *account* relation (represented by **new**.*balance*) results in a negative *balance*.

Trigger example cont'd

```
create trigger overdraft for account before update as
    begin
      if (new.balance < 0) then
        begin
          insert into loan values
            (branch-name, old.account-number, new.balance);
          insert into borrower
            (select customer-name, account-number
            from depositor
            where old.account-number = depositor.account-number);
          update account S set S.balance = 0
            where S.account-number = old.account-number;
        end
    end
```

Stored procedures

- A **stored procedure** makes it possible to store procedural code for applications in the database.
- A stored procedure can perfor complex operations and act as an interface to an application.
- Procedures are executed in the database and can more easily and efficiently retrieve data since they can include SQL queries directly.
- Stored procedures can perform certain error handling using exceptions (comp. exeptions i programming languages)

Stored procedure example

- The following procedure, SUB_TOT_BUDGET, takes a department number as its input parameter, and returns the total, average, minimum, and maximum budgets of departments with the specified HEAD_DEPT. It computes total, average, smallest, and largest department budget.

```
CREATE PROCEDURE SUB_TOT_BUDGET (head_dept CHAR(25))
  RETURNS (tot_budget DECIMAL(12, 2), avg_budget DECIMAL(12, 2),
                min_budget DECIMAL(12, 2), max_budget DECIMAL(12, 2)) AS
  BEGIN
        SELECT SUM(BUDGET), AVG(BUDGET), MIN(BUDGET), MAX(BUDGET)
        FROM DEPARTMENT
        WHERE HEAD_DEPT = :head_dept
        INTO :tot_budget, :avg_budget, :min_budget, :max_budget;
  END;
END
```

- The procedure is executed by:

```
EXECUTE PROCEDURE SUB_TOT_BUDGET(“Sales and Marketing”);
```

Stored proc. vs. triggers

- Triggers are used when one should monitor updates of tables where one do not who, or how, the table will be updated. Should be used with carefulness!!!
- Stored procedures are used to update tables where one knows that the table update will allways be done through the procedure instead of through a direct update.
- Stored procedures are precompiled and can be used as efficient views (views are normally not optimized before they are refererenced), but are usually used for updates.

UPPSALA UNIVERSITET