Chapter 3 Active Databases

- Passive Database Systems

  only execute queries or transactions explicitly submitted by a user or an application program.

- Active database systems

  monitor situations of interest and, when they occur, triggers an appropriate response in a timely manner.

3.1 Rule Models and Languages

The desired behavior of active databases is expressed in production rules (also called event-condition-action rules).

on event

if condition

then action

- Execution

  The rules are triggered by events such as database operations, and when the triggering event occurs the condition is evaluated against the database; if the condition is satisfied the action is executed.

- storage

  The rules are defined and stored in the database, and evaluated by the database, subject to authorization, concurrency control, and recovery.

Limited production rule capabilities are now appearing in commercial database productions, such as Ingres, InterBase, Oracle, Rdb, and Sybase, and in the SQL92 and SQL99 standards.

3.2 Rules specification

- Event Specification

  The most common triggering events in active database rule languages are modifications to the data in the database, such as **insert, delete**, and **update**.

define rule MonitorNewEmps **on insert to** employee **if** ... **then** ...

define rule MonitorSalAccess **on retrieve** employee.salary **if** ... **then** ...

  – SQL92: Assertion

  before commit, after insert, after delete, after update

  – SQL99: Trigger

  before (after) insertion, deletion, update

- Condition Specification

  In all database production rule languages, the condition part of a rule specifies either

  – a predicate: satisfied if the predicate is true, or

  – a query: satisfied if the query returns a nonempty answer.

```
define rule MonitorRaise
on update to employee.salary
if employee.salary > 1.1 * old employee.salary
then ...
```

- Action Specification

  The action part of a rule specifies the operations to be performed when the rule is triggered and its condition is satisfied.

  In SQL99, rule actions can be arbitrary sequences of retrieve and modification commands over any data in the database. Rule actions may also specify rollback to abort the current transaction.

```
define rule FavorNewEmps
on insert to employee
then delete employee e
where e.name = employee.name
```

```
define rule AvgTooBig
on update to employee.salary
if (select avg(salary)
from new-updated > 100)
then rollback
```

```
create trigger DeptDel
before delete on department
when department.budget < 100,000
delete employee
where employee.dno = department.dno
```

Consider a database consisting of the following tables.

```
employee( employee_name, position, salary )
supervision( employee_name, supervisor_name )
```

The first table indicates the job title and salary for each employee and the second indicates who is whose boss.
Write a trigger (or two) to enforce the constraint that every employee has a supervisor, except the president (i.e., the position is "president").

```
create trigger employee_supervisor
before insert or update on employee
for each row
when ( new.position != ‘‘president’’ )
declare
   dummy integer;
begin
   select count(*) into dummy
   from supervision
   where new.employee_name = supervision.employee.name and
         supervision.supervisor_name <> NULL;

   if ( dummy < 1 )
     then raise_application_error( -2000,
                 ’employee must has a supervisor’);
   end if
end
```

```
create trigger supervision_supervisor
before delete on supervision
for each row
declare emp, supervisor integer;
begin
  select count(*) into emp
  from employee e
  where e.employee_name = old.employee_name and
        e.position <> ‘‘president’’;
  select count(*) into supervisor
  from supervision s
  where s.employee_name = old.employee_name and
        s.supervisor <> old.supervisor and
        s.supervisor <> NULL;
  if ( emp > 0 && supervisor < 1 )
    then raise_application_error( -2001,
                    ’employee must has a supervisor’);
    end if
end
```

(A similar trigger for update on supervision is also needed.)

3.3 Rule Ordering

- conflict free

  SQL92 and SQL99 do not allow more than one rule to be defined with the same trigger event, hence conflict resolution is never needed

- conflict resolution

  What if more than one rule is triggered by an event?

  priority, exception hierarchies, partial order

- Rule Organization

  In most relational database systems, rules are defined in the scheme and treated in the same way as other meta-data objects.

  Enable and disable options

3.4 Rule Execution Semantics

The semantics of a database production rule language determines how rule processing will take place at run-time.

- how rules will interact with database operations and transactions.

Rule execution: there are a number of alternatives.

- firing the rule after each tuple is modified
- firing the rules once for the entire set of modifications
- firing the rules at the end of an entire transaction

Execution sequential: what if more than one rule can be triggered by the same event.

- utilize some form of conflict resolution to select one rule at a time, or
- execute all the rules concurrently
- nested triggering

[1] SQL92 and SQL99

An example form Oracle

```
create trigger check_retail
   /* triggering event */
 BEFORE INSERT OR UPDATE OF cost, suggrtl, sellrtl ON product
    FOR EACH ROW
 BEGIN
    /* trigger action - you can reference new and old values */
   IF :new.cost > :new.sellrtl THEN
      raise_application_error(-20225, 'Selling BELOW Cost!');
   ELSIF :new.sellrtl > :new.suggrtl THEN
      raise_application_error(-20230, 'Selling Retail TOO High');
   END IF;
 END;
```

- Rule firing

  Both SQL92 and SQL99 permit tuple-level and set-level processing of assertions and triggers. The choice is made at rule definition time by specifying a **FOR EACH ROW** option.

- Rule processing is strictly sequential.

  No conflict resolution is necessary for no two rules can be defined to have the same triggering event.

- The same table cannot be modified multiple times in a sequence of rule firings.

3.5 Design of Active Databases

- Triggers are very powerful, and thus must be used with caution
- Execution semantics of triggers is too complicated
- Users of triggers
  - to maintain database consistency

    constraints vs. triggers
  - to alert users to unusual events
  - to support auditing and security checks
  - to generate a log of events

3.6 Examples

Example 1 Consider the following relational database schema:

```
emp( Eid, Ename, Age, Salary)
works(Eid, Did, Pct_time )
dept(Did, Budget, Manager )
```

Write triggers to enforce each of the following constraints

(1) Each department has at most one manager.

```
create trigger dept_manager_insert
before insert on dept for each row
when new.Manager not in ( select Manager
                          from dept
                          where Did = new.Did )
     and (select count(*)
          from dept
          where Did = new.Did) > 0
then abort
```

Note that similar triggers are also needed for updates.

(2) A manager of any department must be an employee, i.e. an Eid in emp.

```
create trigger dept_reference_constraint
before insert on dept for each row
when new.Manager not in ( select Eid
                          from emp )
then abort
```

Note that similar triggers are also needed for updates.

(3) Employee must make a minimum salary of $2000.

```
create trigger salary_bottom
before insert on emp for each row
when new.Salary < 2000
then abort
```

(4) The total percentage of all appointments for an employee must be under 100%.

```
create trigger working_load_cap
before insert on works
for each row
when (100 - new.Pct_time) < ( select sum(Pct_time)
                              from works
                              where works.Eid = new.Eid )
```

Example 2 Suppose there are two relations r and s such that the foreign key B of r references the primary key A of s. Describe how the trigger mechanism can be used to implement the ON DELETE CASCADE option, when a tuple is deleted from s.

Note that if a referential constraint specifies ON DELETE CASCADE then all the matching rows of foreign keys will be deleted if a row is deleted from the referenced table.

```
create trigger  simulating_on_delete_cascade
before delete on s
for each row
begin
     delete from r where B = old.A
end
```

Example 3 Write a trigger to count the number of inserted tuples with age < 18 and record the information in the statistics table.

```
create triggers set_count
after insert on students
referencing new table as InsertedTuples
for each statement
   insert into statistics_table( ModifiedTable, Type, Count )
   select 'Students', 'Insert', Count(*)
   from InsertedTuples I
   where I.age < 18
```

Example 4 Consider the account relation below:

account(account_number, branch_name, balance)

with the obvious meaning.

Write triggers, in the Oracle style, to create an audit trail, logging the information into a relation called *account_trail*. The logged information should include the user-id (assume an SQL function user_id() provides this information) and a timestamp, in addition to OLD and NEW values if available. You must also provide the CREATE TABLE statement for the account_trail relation.

You may assume that the account number and branch name cannot be altered. Please also note the differences among INSERT, DELETE, and UPDATE operations.

The create table statement is given below.

```
  CREATE TABLE account_trail(
      user_id INTEGER,
      timestamp DATE,
      operation CHAR(1),
      account_number INTEGER,
      old_balance INTEGER,
      new_balance INTEGER
);
```

```
CREATE TRIGGER log_account_trail
BEFORE insert or delete or update on ACCOUNT
FOR EACH ROW
BEGIN
  IF inserting THEN
    INSERT INTO  account_trail VALUES (user_id(),sysdate,'I',
        :new.account_number,NULL,:new.balance);
  END IF;
  IF updating THEN
    INSERT INTO  account_trail VALUES (user_id(),sysdate,'U',
         :new.account_number,:old.balance,:new.balance);
 END IF;
 IF deleting THEN
   INSERT INTO  account_trail VALUES (user_id(),sysdate,'D',
        :new.account_number,:old.balance,:NULL);
  END IF;
```

Example 5 Consider a database consisting of the following tables with obvious meanings:

```
employee( employee_name, street, city )
works( employee_name, company_name, salary )
company( company_name, city )
manager( employee_name, manager_name )
```

Write triggers to enforce the following constraints.

1. Every employee works for a company located in the same city as the city in which the employee lives.

2. No employee earns a salary higher than that of his/her manager.

The following is a trigger defined for employee, and similar triggers for works and company are also needed.

```
create trigger employee_location
before insert or update on employee
for each row
when (exists (select c.city
              from work, company
              where work.company_name = company.company_name and
                    new.employee_name = works.employee_name and
                    new.city != company.city
             )
      )
abort
```

For Oracle, the trigger can be defined as below

```
create trigger employee_position
before insert or update on employee
for each row
declare
  dummy integer;
begin
  select count(*)  into dummy
  from works, company
  where works.company_name = company.company_name and
        :new.employee_name = works.employee_name  and
        :new.city != company.city;

  if ( dummy > 0 )
    then raise_application_error( -20502,
         'employee must live in the same city as the employer');
    end if;
end;
```

The following is an Oracle trigger defined on works. Similar one is also needed for manager when the supervision relation changes.

```
create trigger salary_cap
before insert or update on works
for each row
declare  dummy integer;
begin
   select count(*) into dummy
   from works, manager
   where :new.employee_name = manager.employee_name and
         works.employee_name = manager.manager_name and
         :new.salary > works.salary;
   if ( dummy > 0 )
     then raise_application_error( -20503,
          'an employee cannot make more than his/her manager');
     end if;
end;
```

Can we enforce the same constraint using FOR-EACH-STATEMENT triggers?

- the difference between FOR EACH ROW and FOR EACH STATEMENT
- REFERENCING NEW TABLE can not be used if BEFORE is specified
- triggers for enforcing constraints usually use BEFORE, not AFTER

3.7 Appendix The following is from the SQL99:

4.23 Triggers

A trigger is defined by a <trigger definition>.

A <trigger definition> specifies a trigger that is described by a trigger descriptor.

A trigger descriptor includes:

The name of the trigger;

The name of the table that is the subject table of the trigger;

The trigger action time (BEFORE, INSTEAD OF, or AFTER);

The trigger event (INSERT, DELETE, or UPDATE) of the trigger;

The old values correlation name, if any, of the trigger;

The new values correlation name, if any, of the trigger;

All of the triggered actions of the trigger;

If the trigger event is UPDATE, then the trigger column list for the trigger event of the trigger, as well as an indication of whether the trigger column list was explicit or implicit;

4.23.1 Triggered action

The definition of a triggered action specifying SQL-statements that are to be executed (either once for each row or once for the whole triggering INSERT, DELETE, or UPDATE statement) before, instead of, or after rows are inserted into a table, rows are deleted from a table, or one or more columns are updated in rows of a table.

The execution of such a triggered action resulting from the insertion, deletion, or updating of a row in a table may cause the triggering of further triggered actions in other affected tables.

```
create table triggers (
       trigger_catalog                   information_schema.sql_identifier,
       trigger_schema                    information_schema.sql_identifier,
       trigger_name                      information_schema.sql_identifier,
       event_manipulation                information_schema.character_data,
       event_object_catalog              information_schema.sql_identifier
         constraint triggers_event_object_catalog_not_null not null,
       event_object_schema               information_schema.sql_identifier
         constraint triggers_event_object_schema_not_null not null,
       event_object_table                information_schema.sql_identifier
         constraint triggers_event_object_table_not_null not null,
       action_order             information_schema.cardinal_number
      not null,
       action_condition         information_schema.character_data,
       action_statement_list    information_schema.character_data
     not null,
       condition_reference_old_table     information_schema.sql_identifier,
       condition_reference_new_table     information_schema.sql_identifier,
       column_list_is_implicit           information_schema.sql_identifier,
       constraint triggers_primary_key
         primary key ( trigger_catalog, trigger_schema, trigger_name ),
       constraint triggers_foreign_key_schemata
         foreign key ( trigger_catalog, trigger_schema )
           references schemata  )
```