Proprietăți relaționale avansate în standardul SQL

prof. dr. ing. Mircea Petrescu

Standardul SQL – defineşte o bază comună pentru definirea, accesul şi folosirea datelor în modelul relațional. De fapt, standardul SQL este un limbaj cuprinzător, care include sublimbaje pentru: definirea datelor, interogarea şi manipularea datelor, restricțiile de integritate, definirea vederilor, definirea funcțiilor şi procedurilor, proceduri stocate, controlul tranzacțiilor, trigger-e, structura programelor SQL, autorizarea utilizatorilor.

Trebuie reținut faptul că în procesul definirii datelor, formularea restricțiilor asupra acestora are o însemnătate înaltă. Privind semnificația unor caracteristici mai avansate ale SQL, vom remarca următoarele:

1. Vederile şi aserțiunile se folosesc în transportul schemelor conceptuale din modelul EEA şi din UML în modelul relațional;
2. Procedurile SQL şi modulele memorate persistente (PSM) sunt importante în dezvoltarea aspectelor de comportament ale aplicațiilor de baze de date şi, din acest punct de vedere, pe ele se sprijină caracteristicile obiectual-relaționale ale SQL;
3. Folosirea trigger-elor pentru „dinamizarea" prelucrării este, de asemenea, un suport al proprietăților obiectual-relaționale ale SQL;
4. Variantele SQL încorporat şi SQL dinamic se pot alătura temei privind funcționarea bazelor de date pe „web"

Unele elemente privind definirea şi manipularea datelor

Rememorăm, mai întâi, caracteristicile limbajului de definire a datelor – în cele ce urmează fiind descrisă sintaxa de nivel înalt pentru *definirea datelor în SQL*. Se observă că definirea de bază a datelor reclamă specificarea tabelelor, coloanelor şi restricțiilor. Mențione specială: fiecare coloană într-o tabelă are un tip de date, aşa cum se arată în tabloul ce urmează sintaxei, precum şi o posibilă specificare a unei valori „lipsă".

Restricții de integritate în standardul SQL

Sunt folosite în definirea schemelor de relație, pentru explicarea înțelesului semantic al datelor din baza de date. Pentru a asigura corectitudinea informației în orice stare dată a bazei de date, toate restricțiile de integritate trebuie respectate. Sintaxa restricțiilor pentru definirea datelor în SQL sunt:

a) sintaxa construirii tabelelor

create table <table_name> (<table_element_list>)
<table_element_list> ::= <table_element> [{, <table_element>} …]
<table_element> ::= <column_definition> | <table_constraint>
<column_definition> ::= <column_name> <data_type> [default <default_value>] [<column_contraint> …]

b) sintaxa restricțiilor privind coloanele

<column_contraint> ::= [constraint <constraint_name>] not null | primary key | unique | <check_constraint> | <column_reference>
<check_constraint> ::= check (<boolean_valued_expression>)
<column_reference> ::= references <referenced_table_name> [(referenced_column_name)] [<update>] [<delete>]

c) sintaxa restricțiilor privind tabelele

<table_constraint> ::= [constraint <constraint_name>] <primary_key_constraint> | <unique_constraint> | <referential_constraint> | <check_constraint>
<primary_key_constraint> ::= primary key (<column_name> [{, <column_name>} …])
<unique_constraint> ::= unique (<column_name> [{, <column_name>} …])

<referential_constraint> ::= foreign key (<referencing_column_name> [{, <referencing_column_name>} …]) <references_specification>
<referencing_specification> ::= references <referenced_table_name> [(<referenced_column_name> [{, <referenced_column_name>} …])] [<update>] [<delete>]
<update> ::= on update <action>
<delete> ::= on delete <action>
<action> ::= no action | cascade | set null | set default

Mai sus s-au folosit următoarele convenţii de notaţie:
< > reprezintă numele elementelor sintactice
[ ] semnifică o sintaxă opţională
{ } semnifică o sintaxă obligatorie
| indică o alegere a sintaxei
... indică o sintaxă ce se repetă
::= separă un element sintactic de definiţia sa

În SQL, restricţiile pot fi formulate fie ca o parte a definiţiei, fie ca restricţii separate asupra definiţiilor de tabele. Restricţiile din cadrul unei definiţii de tabelă pot fi exprimate ca restricţii de coloană sau ca restricţii de tabelă (vezi sintaxa de mai sus).

Tipuri de date în SQL

Categoria	Tipuri de date
şir de caractere	char, varchar, nchar, nchar varying, clob (character large object), nclob
număr	int, integer, smallint, numeric decimal, float, real, double precision
temporal	date, time, timestamp, interval
boolean	boolean (true, false or unknown)
construite	user defined type (UDT) reference type row type collection type
şir binar	blob (binary large object)
şir de biţi	bit, bit varying

Unele menţiuni suplimentare fată de conţinutul sintaxei pentru definirea datelor: restricţia foreign key defineşte noţiunea de integritate referenţială între o cheie străină din o tabelă (relaţie) şi o cheie primară din altă tabelă (sau din aceeaşi). Integritatea referenţiată prin restricţia de cheie străină este fundamentală în definirea schemei unei baze de date relaţionale. Această restricţie obligă valoarea cheii străine (care este compusă din una sau mai multe coloane atribute) să fie sau „null” sau egală cu valoarea unei chei primare la care se referă. Cheia străină trebuie să aibă acelaşi tip de date ca şi cheia primară cu care este legată.

Pentru *actualizarea definiţiilor tabelelor* se folosesc următoarele tipuri de sintaxă:
Sintaxa modificării „alterării” tabelelor: alter table <table_name> etc.
Sintaxa eliminării tabelelor: drop <table_name> <drop_behavior> etc.

Sintaxa limbajului de *manipulare a datelor*:
Sintaxa introducerii de valori: insert into <table_name> [<column_name> …]
Sintaxa eliminării de cod: delete from <table_name> [where <search_condition>]
Sintaxa pentru axtualizare: update <table_name> set <set_clause> …

Sintaxa *vederilor*:

În SQL vederile sunt tabele derivate. Tabelele de acest tip nu sunt memorate fizic în baza de date, ci sunt calculate dinamic prin acțiunea frazelor de interogare. Sintaxa:
create [recursive] view <view_name> [(<column_name> [{, <column_name>} …])] as <SQL_query>;

Aserțiuni

Pentru formularea unor restricții mai generale decât cele corespunzând tabelelor şi coloanelor, SQL dă posibilitatea folosirii unei instrucțiiuni specifice, cu sintaxa:
create assertion <constraint_name> check (search_condition);

Spre deosebire de restricțiile de tabele şi de coloane, aserțiunile nu sunt asociate cu nicio definiție de tabelă particulară. Aserțiunile sunt folosite de obicei pentru a formula restricții asupra mai multor tabele. Remarcăm că pentru a satisface restricția, condiția din clauza check trebuie să furnizeze o valoare fie true, fie unknown.

Sintaxa rutinelor menționate în codul SQL (proceduri stocate)

Cu ajutorul rutinelor menționate în cadrul SQL – sau proceduri stocate – se crează proceduri sau funcții, care apoi pot fi menționate (sau „invocate") din cadrul SQL. Sunt similare cu procedurile şi funcțiile din alte limbaje de programare. Un tip de rutină SQL – invocată – este „metoda" din contextul tipurilor definite de utilizator (UDT).

Distingem două tipuri de rutine invocate din SQL: rutine SQL – scrise în limbajul SQL; rutine externe – scrise într-un limbaj extern, ca C++, Java, Fortran. Decizia privind tipul de rutină este luată de programatorul de aplicație. Rutinele externe pot conduce la „neadaptare de impedanță", dacă tipurile de date din SQL nu se potrivesc cu tipurile de date convenabile limbajului extern.

Sintaxa unei proceduri SQL:
create procedure <routine_name> ([<parameter> …]) <routine_body>
<parameter> ::= [{in | out | inout}] [<parameter_name>] <data_type>
<routine_body> := <SQL_statement>

Sintaxa unei funcții SQL:
create function <routine_name> ([<parameter> …]) returns <datatype> <routine_body>
<parameter> ::= [<parameter_name>] <data_type>
<routine_body> ::= return <value_expression> | null

Sintaxa pentru apelarea (din cod SQL) a procedurilor şi funcțiilor este:
call <routine_name> ([<SQL_argument_name> …])

Crearea şi apelarea rutinelor externe – în documentația de detaliu a standardului SQL.

Module memorate persistente (PSM)

PSM este o opțiune a standardului SQL. Prin PSM sunt îmbogățite rutinele SQL cu concepte tipice pentru limbajele de programare de nivel înalt. PSM sunt implementate complet de puține sisteme comerciale de baze de date. Exemplu de limbaj care amintește de PSM este PL/SQL din Oracle. Standardul SQL actual permite creşterea „completitudinii" de prelucrare prin folosirea următoarelor facilități:

Instrucțiuni compuse:

În PSM, corpurile rutinelor pot fi complicate până la „aserțiuni compuse”, prin folosirea comenzii: begin ... end. Prin folosirea aserțiunilor compuse, se pot introduce specializarea totală şi restricția ISA.

Variabile:
Prin PSM, rutinele SQL pot fi completate cu declarații de variabile şi cu instrucțiuni (comenzi) de atribuire. De exemplu:
declare deptCode varchar(3);
declare deptName varchar(50);
...
set deptCode = ‘CSE’;

Mulțimi de rezultate şi cursoare

În SQL, rezultatul unei interogări poartă numele de „mulțime rezultat”. Cursorul este o proprietate a PSM pe care se sprijină procesul de iterații pe liniile unei mulțimi rezultat. Cursoarele pot fi folosite pentru a controla evoluția execuției comenzilor SQL. În fond, un cursor este un „pointer” folosit pentru a examina fiecare rând al unei mulțimi rezultat.

Sintaxa asociată cu folosirea cursoarelor:
declare <cursor_name> cursor for <SQL_query>
open <cursor_name> [cascade <on|off>]
fetch [[first | last | prior | next] from] <cursor_name> into <variable _name> [{, <variable_name>} …]
close <cursor_name>

Instrucțiuni (comenzi) pentru controlul fluxului prelucrării

Cursoarele sunt foarte utile în efectuarea diferitelor tipuri de comenzi de „buclare” („looping”) folosite de PSM. Dintre acestea, se menționează: for, while, loop, repeat. Folosirea acestora este similată cu modul de utilizare în limbajele de nivel înalt.

Dinamizarea prelucrării cu ajutorul trigger-elor

Trigger-ele sunt destinate dinamizării prelucrării sistemelor relaționale de baze de date. Se bazează pe noțiunea de eveniment – condiție – acțiune, respectiv pe reguli ce țin de cele trei componente ale principiului de mai sus. Acestea sunt „reguli active”, definite inițial în domeniul *bazelor de date active*.

Un sistem activ de bază de date oferă proiectantului unei aplicații posibilitatea de a monitoriza apariția unor tipuri specifice de evenimente care se porduc în baza de date. Aceste evenimente sunt de obicei modificări ale datelor, dar pot fi şi mențiónări (invocări) de proceduri şi funcții, sau chiar producerea de evenimente generate de tactul sistemului. Când se produce un eveniment, o regulă activă determină evaluarea unei condiții. Dacă rezultatul este „adevărat”, se execută acțiunea regulei.

Regulile active sunt pentru corectarea încălcării restricțiilor, gestiunea fişierelor-jurnal, etc.

Sintaxa trigger-elor – specifică:
1) apariția în timp a trigger-ului, prin before ... sau after ...
2) tipul de trigger – row sau statement;
3) clauza referencing, pentru folosirea tabelelor de tranzacție.

Sintaxa debutează ca mai jos, pentru trigger-ul cu numele statement:

create trigger <trigger_name> {before | after} {insert | delete | update [of <column_name> [{, <column_name>} …]]} on <table_name> [referencing <old_or_new_values_alias_list>] [for each {row | statement}] [when (search_condition)] <trigger_SQL_statement>

<old_or_new_values_alias_list> ::=
    [old [row] [as] old values <correlation_name>]
    [new [row] [as] new values <correlation_name>]
    [old table [as] <old_values_table_name>]
    [new table [as] <new_values_table_name>]
<trigger_SQL_statement> ::= <SQL_statement> | begin atomic <SQL_statement> … end

SQL incorporat

Limbaje care acceptă folosirea SQL incorporat sunt Ada, C, Cobol, Fortran, MUMPS, Pascal, PL/I şi SQLJ (pentru Java). Într-un limbaj care acceptă SQL incorporat, fiecare instrucțiune SQL trebuie precedată de: exec sql ... ca mai jos:
Exemplu – cu limbaj gazdă C:
exec sql begin declare section;
    char code[3];
    char name[40];
exec sql end declare section;
strcpy(code, "ASC");
strcpy(name, "Automatica si Stiinta Calculatoarelor");
exec sql insert into department values (:code, :name);

SQL dinamic

SQL incorporat este static; aici toate instrucțiunile SQL ce urmează să fie executate dintr-un program scris în limbaj gazdă, sunt cunoscute dinainte. Avantaj al SQL static (incorporat): instrucțiunile SQL sunt compilate şi optimizate în timpul unui proces de pre-prelucrare, ceea ce îmbunătățeşte execuția aplicației.

În mod diferit, SQL dinamic este o parte a standardului SQL actual care foloseşte SQL în mod direct în codul aplicației. Cu alte cuvinte, utilizatorii introduc direct interogări SQL, în mod dinamic, sau construiesc interogări în interiorul codului aplicației. Pentru a compila, optimiza şi executa o frază de interogare în acest regim, se foloseşte comanda: execute immediate.

Interfața de nivel de apelare

Această interfață (CLI – Call-Level Interface) este o manieră mai dinamică de a comunica cu o bază de date, decât SQL incorporat sau SQL dinamic. *Avantajul CLI este că nu necesită un preprocesor* SQL dinamic. Pentru interacțiunea cu bazele de date relaționale este folosită „interfața de programare pentru aplicație" (API – application programming interface) – amplasată între limbajul de programare folosit şi baza de date.

Prin urmare, un limbaj care utilizează CLI poate comunica dinamic cu câteva sisteme de baze de date. Deci, programul nu este compilat în mod specific pentru comunicare cu un singur sistem de bază de date, ca în cazul SQL incorporat. Un alt avantaj al CLI este că permite interogarea metadatelor unei baze de date. Originea CLI – în ODBC – Open Database Connectivity.