Prilog

C

UGRAĐENI PROGRAMSKI SQL

Sve SQL naredbe koje smo obradili u poglavlju 6 mogu se koristiti u interaktivnom režimu rada sa bazom podataka. U takvom režimu rada, korisnik zadaje jednu po jednu SQL naredbu, a sistem za upravljanje bazom podataka je izvršava ili odbacuje ako je po bilo kom osnovu neispravna. Pri tome, korisnik može da uradi sve što je u granicama njegovih prava.

Ono što odmah možemo da uočimo u vezi interaktivnog režima rada jeste nemogućnost formiranja procedura koje bi automatizovale kompleksne manipulacije nad bazom podataka.

Uz prethodno, ozbiljna mana interaktivnog rada sa bazom podataka jeste i to što od svakog korisnika traži poznavanje SQL jezika. U tradicionalnim informacionim sistemima to nije bio slučaj: korisnik je pristupao podacima posredstvom posebno izrađenih programa, i njegovo je bilo samo da vrši izbor opcija, unos podataka i uvid u razne preglede, bez ulaženja u programirane detalje manipulacije podacima.

Navedene okolnosti uslovile su da SQL jezik od početka sadrži naredbe koje omogućavaju njegovo korišćenje u programima pisanim na standardnim proceduralnim programskim jezicima. Način na koji se to ostvaruje već smo opisali u poglavlju 2:

- u izvornom programu na nekom standardnom programskom jeziku na određenim mestima se ubacuju standardni SQL delovi teksta;
- SQL preprocesor prevodi SQL delove teksta u pozive posebnih funkcija i procedura, što daje izvorni program po standardu za određeni programski jezik;
- standardni izvorni program se prevodi i povezuje u izvršnu celinu, prilikom čega se uz standardnu koristi i SQL biblioteka posebnih funkcija i procedura.

Na opisani način dobijaju se izvršni programi koji kombinuju sve prednosti tradicionalnih programskih jezika i SQL jezika.

U narednim odeljcima izložene su osnove upotrebe SQL jezika u programima pisanim na standardnim programskim jezicima, pri čemu se ograničavamo na programski jezik C. Radi preglednosti, sve programske SQL konstrukcije u C programskom tekstu koji sledi naglašene su tamnijim slovima i znakovima, a nebitni delovi programa kao što su unošenje i ispisivanje podataka navedeni su kao <opis>.

C.1 Deklaracija programskih varijabli

Prilikom sastavljanja programa na nekom standardnom programskom jeziku za rad sa bazom podataka, prvo što moramo definisati jesu varijable koje služe kao sprega sa podacima u bazi podataka. Te varijable se koriste i u standardnim i u SQL delovima programa, a deklaracija koju prepoznaje SQL preprocesor je sledeće forme:

```
DeklaracijaSpreznihVarijabli ::=
   EXEC SQL BEGIN DECLARE SECTION ;
   DeklaracijeVarijabliProgramskogJezika
   EXEC SQL END DECLARE SECTION ;
```

Pri tome, korespondencija između tipova podataka SQL i C jezika je sledeća:

```
   SQL tip                C tip
   ---------------------  ---------------------------------
   INTEGER                int
   REAL                   float
   CHARACTER              char
   CHARACTER [n]          char [n]
   CHARACTER VARYING [n]  char [n+1] organizovan kao string
```

Primer

U C programu za rad sa tabelom Pozajmica deklaracija bi glasila:

```
EXEC SQL BEGIN DECLARE SECTION ;
         int  SifP ;
         char SifC [3] ;
         char SifK [3] ;
         char SifN [4] ;
         int  Dana ;
EXEC SQL END DECLARE SECTION ;
```

C.2 Programski SELECT i INSERT za jedan red

Pod programskim **SELECT** i **INSERT** naredbama za jedan red podrazumevamo naredbe koje se odnose isključivo na jedan red tabele, odnosno:

- **SELECT** koji kao rezultat daje jedan red;
- **INSERT** koji ubacuje jedan red u tabelu.

Navedena **SELECT** naredba se po izvesnim dodacima razlikuje od one koja se koristi u interaktivnom režimu rada. Njena sintaksna definicija glasi:

```
ProgramskiSELECTJedanRed ::=
   EXEC SQL SELECT    { R-Izraz | G-Izraz } ,..
             INTO     { :Varijabla } ,..
             FROM     { Tabela [ Nadimak ] } ,..
          [ WHERE     R-Predikat ]
          [ GROUP BY Kolona ,..
          [ HAVING    G-Predikat ] ] ;
```

Jedino ograničenje koje važi jeste to da se uvek dobija rezultat u formi jednog i samo jednog reda. Unutar tog ograničenja, dozvoljeno je sve što smo do sada naveli za **SELECT** naredbu: svođenje, spajanje, skupovne operacije, podupiti, pogledi. U **R-Predikat** se mogu koristiti i programske varijable, sa dvotačkom ispred naziva. U **INTO** klauzuli navode se, sa dvotačkom ispred, jedna ili više programskih varijabli odvojenih zarezima. Te varijable moraju odgovarati po broju, redosledu i tipu elementima iza **SELECT** klauzule.

Navedena naredba **INSERT** za programski režim rada je forme:

```
ProgramskiINSERTJedanRed ::=
   EXEC SQL INSERT INTO Tabela [ ( Kolona ,.. ) ]
            VALUES ( { { :Varijabla } | Konstanta } ,.. ) ;
```

Razlika u odnosu na interaktivnu formu **INSERT** naredbe jeste ta što se kao vrednosti pored konstanti mogu navesti i programske varijable, a sve to mora po broju, redosledu i tipu biti saglasno sa deklaracijom kolona, ili ako ona nije data sa **CREATE TABLE** naredbom za tabelu **Tabela**.

Primeri

Sledeći programski segment na C jeziku za datu postojeću šifru naslova prikazuje naziv naslova i naziv oblasti:

```
...
EXEC SQL BEGIN DECLARE SECTION ;
         char [4]  SifN ;
         char [20] NazivN ;
         char [20] NazivO ;
EXEC SQL END DECLARE SECTION ;
...
{
 < ucitavanje SifN > ;
 EXEC SQL SELECT N.Naziv, O.Naziv
           INTO  :NazivN, :NazivO
           FROM  Naslov N, Oblast O
           WHERE N.SifO = O.SifO
             AND N.SifN = :SifN ;
 < ispisivanje NazivN i NazivO > ;
 }
```

Programski segment na C jeziku kojim se unosi podatak o novom članu glasi:

```
...
EXEC SQL BEGIN DECLARE SECTION ;
         char [3]  SifC ;
         char [15] Ime ;
EXEC SQL END DECLARE SECTION ;
...
{
 < ucitavanje SifC i Ime > ;
 EXEC SQL INSERT INTO Clan
                 VALUES ( :SifC, :Ime ) ;
}
```

C.3 Programski DELETE i UPDATE

Za naredbe programskog ažuriranja **UPDATE** i **DELETE** ne postoje ograničenja na jedan red tabele, i njima se može postići ažuriranje ni jednog, jednog ili više redova, uključujući i sve.

Sintaksna definicija programske naredbe uklanjanja **DELETE** u neposrednoj formi (posredna će biti izložena kasnije) glasi:

```
ProgramskiDELETE ::=
   EXEC SQL DELETE FROM  Tabela [ Nadimak ]
                 [ WHERE R-Predikat ] ;
```

gde za **R-Predikat** važi sve ranije rečeno, uz dodatne napomene:

- mogu se koristiti i programske varijable, sa dvotačkom ispred naziva;
- nije dozvoljen podupit nad tabelom iz koje uklanjamo redove.

Izvršenjem ove naredbe uklanjaju se iz tabele **Tabela** oni redovi za koje je zadovoljen **R-Predikat** ili svi redovi ako **R-Predikat** nije zadat.

Za programsku naredbu izmene **UPDATE** u neposrednoj formi imamo sledeću definiciju

```
ProgramskiUPDATE ::=
   EXEC SQL UPDATE Tabela [ Nadimak ]
            SET   { Kolona = R-Izraz } ,..
          [ WHERE R-Predikat ] ;
```

uz sledeće napomene:

- u **R-Izraz** i **R-Predikat** se mogu koristiti i programske varijable, sa dvotačkom ispred naziva;
- u **R-Predikat** nije dozvoljen podupit nad tabelom u kojoj menjamo redove.

Ovakva naredba vrši naznačene izmene u onim redovima tabele **Tabela** za koje je zadovoljen **R-Predikat** ili u svim ako **R-Predikat** nije zadat.

I pored jednostavnosti navedenih formi naredbi, one se retko koriste u praksi. Umesto toga, koristi se posredna forma koja se kao i druga forma **SELECT** naredbe zasniva na prethodnom definisanju skupa redova koji su predmet manipulacije i potom iterativnoj obradi tog skupa "red po red".

Primeri

Sledeći programski segment uklanja podatke o pozajmicama zadate knjige:

```
...
EXEC SQL BEGIN DECLARE SECTION ;
         char [3] SifK ;
EXEC SQL END DECLARE SECTION ;
...
{
 < ucitavanje SifK > ;
 EXEC SQL DELETE FROM  Pozajmica
                 WHERE SifK = :SifK ;
}
```

Programski segment za ispravku pogrešno unete šifre oblasti naslova glasi

```
...
EXEC SQL BEGIN DECLARE SECTION ;
         char [4] SifN ;
         char [2] SifO ;
EXEC SQL END DECLARE SECTION ;
...
{
 < ucitavanje SifN i SifO > ;
 EXEC SQL UPDATE Naslov
          SET   SifO = :SifO
          WHERE SifN = :SifN ;
}
```

C.4 Kursor i operacije nad njim

Navedimo dve bitne razlike između interaktivnog jezika SQL i standardnih proceduralnih programskih jezika:

- interaktivni SQL jezik ne raspolaže sa konstrukcijama grananja i ponavljanja.
- proceduralni jezici su u radu sa datotekama slogovno orijentisani, u tom smislu da se podaci obrađuju "slog po slog" i da uvek postoji tekući slog kome se pristupa; nasuprot tome, SQL jezik je tabelarno orijentisan - predmet manipulacija su cele tabele i rezultati su u opštem slučaju cele tabele;

Prva razlika se premoštava kombinacijom standardnih i programskog SQL jezika, dok nam druga do sada nije predstavljala problem s obzirom na jednostavnost naših primera. Sve programske `SELECT` naredbe koje smo do sada naveli odnosile su se na samo jedan red rezultatnih podataka i takav rezultat se mogao preuzeti `INTO` klauzulom u programske varijable.

Za slučajeve upita koji kao rezultat daju više redova programski SQL predviđa sledeće dodatne konstrukcije:

- deklaraciju skupa rezultantnih redova preko definicionog upita; takav skup se u SQL terminologiji naziva kursor
- otvaranje kursora, što podrazumeva izvršenje definicionog upita i nastanak rezultatnog skupa redova; nakon toga, nad njim se mogu obavljati operacije "red po red", što odgovara slogovnoj orijentaciji proceduralnih programskih jezika;
- zatvaranje kursora, kada više nije potreban.

Razmotrimo prvo naredbu deklaracije kursora, čija je sintaksna definicija:

```
DeklaracijaKursora ::=
   EXEC SQL DECLARE CURSOR Kursor
                          FOR R-Upit ;
```

Ovde `R-Upit` može biti bilo koja od formi koje smo obradili u odeljcima o `SELECT` naredbi u poglavlju 6. U slučaju da nad redovima rezultata želimo da vršimo ažuriranje, na snazi su ograničenja koja smo definisali za ažurabilnost pogleda. Unutar istog programa možemo deklarisati više kursora sa različitim nazivima `Kursor` koji se formiraju po pravilima za nazive programskih varijabli.

Prethodna naredba predstavlja samo deklaraciju. Da bi skup redova koji čine kursor bio dostupan za obradu, kursor treba otvoriti naredbom čija je forma:

```
OtvaranjeKursora ::=
   EXEC SQL OPEN CURSOR Kursor ;
```

Po izvršenju naredbe otvaranja kursora, prvi rezultantni red (ako ga ima) u skupu nastalom izvršenjem definicionog upita postaje tekući red.

Na kraju programa ili kada nam kursor više ne treba, kursor zatvaramo naredbom sledeće sintakse:

```
ZatvaranjeKursora ::=
   EXEC SQL CLOSE CURSOR Kursor ;
```

Naredba učitavanje podataka iz tekućeg reda kursora je forme:

```
CitanjeRedaKursora ::=
   EXEC SQL FETCH Kursor
                 INTO :Varijabla ,.. ;
```

Navedena lista programskih varijabli mora biti saglasna po broju, redosledu i tipu sa listom kolona iz definicionog upita kursora. Posle izvršenja ove naredbe, naredni red u kursoru postaje tekući red.

Naredba izmene tekućeg reda ažurabilnog kursora je nešto složenija Njena sintaksna definicija glasi:

```
IzmenaRedaKursora ::=
   EXEC SQL UPDATE Tabela
                 SET  { Kolona = R-Izraz } ,..
                 WHERE CURRENT OF Kursor ;
```

Ovde su neophodne sledeće napomene:

- svakom **UPDATE** treba da prethodi **FETCH** iz istog kursora;
- **Tabela** je naziv tabele navedene u definicionom upitu kursora;
- **R-Izraz** može da sadrži i programske varijable sa dvotačkom ispred naziva;
- **WHERE CURRENT OF** klauzula znači da se izmena vrši u redu koji je prethodno pročitan sa **FETCH**.

Izvršenje ove naredbe ne pomera indikator tekućeg reda.

Preostaje još da razmotrimo naredbu uklanjanja tekućeg reda kursora, za koju važi sledeća sintaksna definicija:

```
UklanjanjeRedaKursora ::=
   EXEC SQL DELETE FROM  Tabela
                  WHERE CURRENT OF Kursor ;
```

pri čemu važe napomene kao i za **UPDATE** naredbu. Nakon izvršenja ove naredbe, tekući red kursora postaje red koji se nalazio iza uklonjenog reda

Na kraju, napomenimo da se uslov za izmenu ili uklanjanje tekućeg reda kursora ili mora ugraditi u definicioni upit kursora ili razrešiti u programu uslovljavanjem izvršenja odgovarajuće SQL naredbe.

Primeri

Navešćemo primere za upit i operacije izmene i uklanjanja nad kursorom. Neka za sve navedene primere važe sledeće deklaracije varijabli i kursora:

Varijable:

```
...
EXEC SQL BEGIN DECLARE SECTION ;
         char [4]  SifN ;
         char [20] Naziv ;
         char [2]  SifO ;
         char [2]  SifOZad ;
         char [2]  SifOSta ;
         char [2]  SifONov ;
EXEC SQL END DECLARE SECTION ;
...
```

Deklaracija kursora:

```
...
EXEC SQL DECLARE CURSOR SviNaslovi
                        FOR SELECT *
                        FROM Naslov ;
...
```

Upit

Programski segment za prikaz naziva svih naslova zadate šifre oblasti:

```
...
{
 EXEC SQL OPEN CURSOR SviNaslovi ;
 < ucitavanje  SifOZad > ;
 < ispisivanje SifOZad > ;
 do while (1)                        // Objasnjenje naknadno
    {
     EXEC SQL FETCH SviNaslovi
                   INTO :SifN, :Naziv, :SifO ;
     if ( < NistaNijeUcitano > )   // Objasnjenje naknadno
        break ;
     if ( SifO == SifOZad )
        < ispisivanje Naziv > ;
    }
 EXEC SQL CLOSE CURSOR Svi Naslovi ;
}
```

Izmena

Programski segment kojim se stara šifra oblasti naslova menja u novu:

```
...
{
 EXEC SQL OPEN CURSOR SviNaslovi ;
 < ucitavanje SifOSta i SifONov > ;
 do while (1)
    {
     EXEC SQL FETCH SviNaslovi
                  INTO :SifN, :Naziv, :SifO ;
     if ( < NistaNijeUcitano > )
        break ;
     if ( SifO == SifOSta )
        EXEC SQL  UPDATE Naslov
                        SET SifO = :SifONov
                        WHERE CURRENT OF SviNaslovi ;
    }
 EXEC SQL CLOSE CURSOR SviNaslovi
}
```

Brisanje

Programski segment za uklanjanje naslova zadane šifre oblasti.

```
...
{
 EXEC SQL OPEN CURSOR SviNaslovi ;
 < ucitavanje SifOZad > ;
 do while (1)
    {
     EXEC SQL FETCH SviNaslovi
                  INTO :SifN, :Naziv, :SifO ;
     if ( < NistaNijeUcitano > )
        break ;
     if ( SifO == SifOZad )
        EXEC SQL DELETE FROM Naslov
                 WHERE CURRENT OF SviNaslovi ;
    }
 EXEC SQL CLOSE CURSOR SviNaslovi ;
}
```

Uz prethodna tri primera neophodna su određena objašnjenja. U sva tri slučaja korišćena je bezuslovna programska petlja `do while (1)` iz koje se iskače kada SQL naredba čitanja reda iz kursora ne vrati ništa pošto nema više redova za čitanje. To podrazumeva da programski SQL podržava ispitivanje ishoda operacija.

C.5 Varijabla SQLCODE i naredba WHENEVER

U prethodnim primerima ostalo je nerešeno pitanje prepoznavanja kraja kursora, odnosno situacije kada **FETCH** naredba više nema šta da čita. Isto tako, u ranijim jednostavnim primerima ostavili smo nerešene situacije traženja reda koji ne postoji, ubacivanja reda sa vrednošću primarnog ključa koja već postoji u tabeli, i slično. Sve te situacije razrešene su u programskom SQL jeziku preko posebne sistemske varijable **SQLCODE** koja se u programu mora deklarisati kao varijabla tipa **INTEGER** i koja se posle izvršenja svake programske SQL naredbe postavlja na vrednost koja govori o ishodu, u smislu uspešnog izvršenja ili neke greške.

Vrednosti na koje se postavlja **SQLCODE** posle izvršenja svake programske SQL naredbe pokrivaju sve moguće ishode, odnosno sledeće situacije:

- komanda je izvršena bez greške i imala je efekta, odnosno nije nastupila ni jedna od niže navedenih situacija:
  SQLCODE se postavlja na vrednost 0 ;
- komanda je izvršena bez greške ali nije imala nikakvog efekta, odnosno: **SELECT** nije vratio ni jedan red rezultata, **FETCH** nema šta da učita (već je učitan poslednji red kursora ili njegov definicioni upit nije dao ni jedan red), za neposredni **UPDATE** ili **DELETE** ni jedan red tabele ne zadovoljava **R-Predikat** u **WHERE** klauzuli:
  SQLCODE se postavlja na 100;
- prilikom izvršenja komande nastupila je neka greška; svi dotadašnji efekti komande se poništavaju:
  SQLCODE se postavlja na neku negativnu vrednost koja bliže identifikuje nastalu grešku; konkretna vrednost zavisi od implementacije.

Varijablu ishoda **SQLCODE** možemo u programu koristiti u uslovnim grananjima i petljama i tako obraditi svaki mogući ishod, ali i pored toga u SQL jeziku postoji posebna naredba grananja (u stvari obrade izuzetka) čija je forma:

```
DefinicijaObradeIzuzetka ::=
   EXEC SQL WHENEVER Ishod CONTINUE | { GOTO Labela } ;

Ishod :=
   Konstanta | { { SQLWARNING | NOT FOUND } | SQLERROR }
```

čije je značenje je sledeće: ako posle neke operacije **SQLCODE** vrati vrednost **Ishod**, ili se izvršava sledeća programska naredba ako je navedena akcija **CONTINUE** ili se vrši skok na programsku naredbu sa labelom **Labela** ako je navedena **GOTO** akcija. Ishod se osim brojnom konstantom može specificirati i navođenjem jedne od standardnih simboličkih konstanti za **SQLCODE**:

- **SQLWARNING** ili **NOT FOUND** kao ekvivalemt vrednosti 100;
- **SQLERROR** za slučaj neke greške, odnosno kao ekvivalent negativne vrednosti.

C.6 Primer složenog održavanja baze podataka

Kao primer složenog održavanja baze podataka poslužiće nam situacija kada član vraća pozajmljenu knjigu, o čemu je već bilo reči na kraju poglavlja 4. Neka nam kao polazna osnova za to posluži ranije navedena algoritamska specifikacija postupka **VracanjeKnjige**, i to u varijanti koraka 2:

```
VracanjeKnjige (  > pSifK )
  :  Inicijalizacija
  :  Obrada
  :  :  Uvid u neophodne podatke
  :  :  :  Uvid koji je clan i kada uzeo knjigu
  :  :  :  Uvid kojeg je naslova knjiga
  :  :  Evidentiranje vracanja knjige
  :  :  Evidentiranje obavljene pozajmice
  :  :  :  Odredjivanje nove vrednosti primarnog kljuca
  :  :  :  Odredjivanje koliko je trajala pozajmica
  :  :  :  Evidentiranje podataka o pozajmici
  :  :  Obrada eventualne rezervacije
  :  :  :  Provera da li ima rezervacije za vracenu knjige
  :  :  :  Opsluzivanje rezervacije (ako treba)
  :  :  :  :  Nalazenje prioritetne rezervacije
  :  :  :  :  Evidenriranje da je rezervacija opsluzena
  :  :  :  :  Evidentiranje da je knjiga rezervisana
  :  Finalizacija
```

Sledi odgovarajuća funkcija na programskom jeziku C sa ubačenim programskim SQL naredbama. Prethodno je potrebno da definišemo određene pomoćne funkcije koje su neophodne s obzirom da se u bazi podataka BIBLIOTEKA kao tip većine atributa koristi niz znakova umesto stringa:

```
void KopijaNiza ( char *NizU, char *NizIz, int n ) ;
```

Kopira n znakova iz znakovog niza **NizIz** u znakovni niz **NizU**;

```
int BrojDana ( char *Datum ) ;
```

za datum u obliku znakovnog niza **Datum** dužine 8 vraća broj dana između tog datuma i tekućeg datuma;

```
void TekuciDatum ( char *Datum ) ;
```

vraća tekući datum u obliku znakovnog niza **Datum** dužine 8'.

Sama funkcija **VracanjeKnjige** glasi:

```
#include "sql.h" ;
int VracanjeKnjige ( char *pSifK )
    {
/*   Inicijalizacija */
       {
        enum ( NE_REZERVISANA, JE_REZERVISANA ) ;
        int vIshod = NE_REZERVISANA ;
        EXEC SQL BEGIN DECLARE SECTION ;
                 char vSifC [3] ;
                 char vSifN [4] ;
                 char vSifK [3] ;
                 char vDatum [8] ;
                 char vDatumVreme [14] ;
                 int  vSifP ;
                 int  vDana ;
        EXEC SQL END DECLARE SECTION ;

        KopijaNiz ( vSifK, pSifK, 3 ) ;
       }

/*   Obrada */
       {
/*      Uvid u neophodne podatke */
         {
          EXEC SQL SELECT SIFC, DATE_CHAR ( DATUM )
                   INTO  :vSifC, :vDatum
                   FROM  DRZI
                   WHERE SIFK = :vSifK ;

          EXEC SQL SELECT SIFN
                   INTO  :vSifN
                   FROM  KNJIGA
                   WHERE SIFK = :vSifk ;
         }
/*      Evidentiranje vracanja knjige */
         {
          EXEC SQL DELETE FROM DRZI
                   WHERE SIFK = :vSifK ;
         }
```

nastavlja se

nastavak

```
/*      Evidentiranje obavljene pozajmice
          {
           EXEC SQL SELECT MAX ( SIFP )
                    INTO  :vSifp
                    FROM  POZAJMICA ;

           vSifP ++ ;
           vDana = BrojDana ( vDatum ) ;

           EXEC SQL INSERT INTO POZAJMICA
                          VALUES ( :vSifP, :vSifC,
                                   :vSifK, :vSifN, :vDana ) ;
          }
/*      Obrada eventualne rezervacije * /
          {
           EXEC SQL SELECT MIN ( DATETIME_CHAR ( DATUM ) )
                    INTO  :vDatumVreme
                    FROM  REZERVACIJA
                    WHERE SIFN = :vSifN ;

           if ( !SQLCODE )
              {

               vIshod = JE_REZERVISANA ;

               EXEC SQL SELECT SIFC
                        INTO  :vSifC
                        FROM  REZERVACIJA
                        WHERE DATUM = CHAR_DATETIME ( :vDatumVreme ) ;

               EXEC SQL DELETE FROM REZERVACIJA
                        WHERE DATUM = CHAR_DATETIME ( :vDatumVreme ) ;

               TekuciDatum ( vDatum ) ;

               EXEC SQL INSERT INTO JE_REZERVISANA
                              VALUES ( :vSifK,:vSifC,
                                       CHAR_DATE ( vDatum ) ;
              }
          }
        }
/*   Finalizacija */
          {
           return vIshod ;
          }
    }
```

Pri tome smo koristili slede}e funkcije konverzije koje pod istim ili druga~ijim nazivima podr`ava ve}ina SQL implementacija:

DATE_CHAR	konvertuje tip **DATE** u niz znakova;
DATETIME_CHAR	konvertuje tip **TIMESTAMP** u niz znakova;
CHAR_DATE	konvertuje niz znakova u tip **DATE**;
CHAR_DATETIME	konvertuje niz znakova u tip **TIMESTAMP**.