38.Példa

Adjuk meg azt az SQL parancsot, amely beírja a hiányzó adatokat a fenti Dolgozó táblába!

```
UPDATE Dolgozó SET 'A dolgozó születési helye' =
"Szeged", 'A dolgozó születési ideje' = 1967.05.12.,
'A dolgozó fizetése' = 120000 WHERE 'A dolgozó
törzsszáma'="T429877"
```

Az így kapott tábla:

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T343234	Kiss Timót	Eger	1970. 02. 28.	105000
T429877	Kovács János	Szeged	1967. 05. 12.	120000

Az SQL beépítése programozási nyelvekbe

Interface az SQL és a befogadó nyelv között

Mint láthattuk az SQL nem programozási nyelv. Nem tartalmaz semmilyen vezérlőszerkezetet, nem lehet változókat deklarálni, csupán az adatbázisban található azonosítókat, illetve az SQL megvalósításban létező utasításokat, függvényeket használhatjuk, amelyek főképp az adathozzáférést szolgálják. Egy komplett alkalmazás elkészítéséhez azonban ez nem elegendő. Ebben különböző algoritmusokat kell megvalósítani, gondoskodni kell az adatok megfelelő beolvasásáról, megjelenítéséről. Ehhez pedig programozási munkára van szükség, amely nem valósítható meg programozási nyelv nélkül. Vagyis ha az SQL-t valódi alkalmazások fejlesztésénél is alkalmazni kívánjuk, akkor valamilyen formában **be kell építeni az ABKR programozási környezetébe.** Az ABKR-ek nagyon különböző programozási eszközöket tartalmaznak. Vannak olyan rendszerek, amelyek eleve **saját, adatbázis-kezelést is támogató nyelvvel rendelkeznek.** Ebbe általában jól beilleszthetők az SQL utasítások, mondhatni azt is, hogy ezek a nyelv részét képezik.

Egy másik lehetőség, hogy az **ABKR hagyományos programozási nyelvre épül**. Erre az esetre definiáltak egy technikát, melynek segítségével az SQL utasítások beépíthetők az adott programnyelvbe. Ebben az esetben a programozó az utasítások között a megfelelő interface-t használva **elhelyezi az SQL utasításokat. Egy előfordító segítségével ezután az SQL utasítások az adott programnyelvre fordítódnak.** Ez például úgy történhet, hogy az SQL utasításokat átalakítják olyan függvényhívássá, amely paraméterében a megfelelő SQL utasítást várja, és végrehajtja azt. Ezután már a hagyományos fordítóprogram is képes lefordítani és összeszerkeszteni a megfelelő kódot, amely így a szükséges SQL utasításokat is végrehajtja.

A következőkben ismertetjük azokat a technikákat, amelyek segítségével **közös változókat** lehet definiálni, vagyis olyanokat, amelyeket mind a **programozási nyelv, mind az SQL utasítások használhatnak.**

Ezeket ***megosztott elérésű változóknak*** nevezzük, a programozási nyelvben a hagyományos módon hivatkozhatunk rájuk, míg

az SQL-ben, hogy **az oszlopnevektől meg tudjuk különböztetni őket, egy kettőspontot kell eléjük tennünk.**

A megosztott elérésű változók között általában speciális szerepet játszik az **SQLSTATE** nevű változó, amelyet a legtöbb implementáció definiál. Ez általában egy 5 karaktert tartalmazó változó, amely az SQL parancsok végrehajtása során bekövetkező esetleges hibákat tárolja, és ezeket kapja vissza ilyen módon a befogadó környezet. Így az mindig ellenőrizni tudja, hogy egy SQL parancs végrehajtása sikeres volt-e, illetve milyen probléma lépett fel.

A beágyazott SQL utasításokat a legtöbb rendszerben az

EXEC SQL kulcsszavakkal kell bevezetni.

Ez arra szolgál, hogy az SQL parancsokat el lehessen választani a program többi részétől.

A megosztott elérésű változók deklarációját a következő módon kell végrehajtani:

```
EXEC SQL BEGIN DECLARE SECTION
...
EXEC SQL END DECLARE SECTION
```

A fenti két utasítás közötti részt deklarációs résznek nevezzük. Itt kell deklarálni azokat a változókat, amelyeket megosztott elérésű változóként szeretnénk használni. A változók deklarációjának szintaxisa megegyezik a befogadó nyelvben használatos deklarálási szintaxissal.

39.Példa

Adjuk meg azt a deklarációt, amely két megosztott elérésű változót definiál C befogadó környezetben, amelyben a dolgozók nevét és fizetését tárolhatjuk!

```
EXEC SQL BEGIN DECLARE SECTION;
    char dnev[50];
    long dfiz;
    char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

40.Példa

Adjuk meg azt az SQL parancsot, amely az előző példában definiált dfiz nevű megosztott elérésű változóba beolvassa a T429877 törzsszámú dolgozó adatát az alábbi táblából, majd a fizetés értékét 20%-kal megemelve az új adatot visszaírja.

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T343234	Kiss Timót	Eger	1970. 02. 28.	105000
T429877	Kovács János	Szeged	1967. 05. 12.	120000

A megoldás a következő:

```
EXEC SQL SELECT 'A dolgozó fizetése' INTO :dfiz FROM
Dolgozó WHERE 'A dolgozó törzsszáma' = "T429877";
dfiz = 1.2*dfiz;
EXEC SQL UPDATE Dolgozó SET 'A dolgozó fizetése'=
:dfiz WHERE 'A dolgozó törzsszáma' = "T429877";
```

Eredményül az alábbi táblát kapjuk:

A dolgozó törzsszáma	A dolgozó neve	A dolgozó születési helye	A dolgozó születési ideje	A dolgozó fizetése
T234578	Kiss István	Eger	1968. 12. 11.	120000
T343234	Kiss Timót	Eger	1970. 02.	105000

			28.	
T429877	Kovács János	Szeged	1967. 05. 12.	144000

SQL parancsok eredményének felhasználása a befogadó nyelvben

Mint tudjuk általános esetben az SQL parancsok eredménye egy halmaz, amely ráadásul bonyolult adatstruktúrájú elemekből áll.
Mindenegyes eleme tulajdonképpen egy rekord, vagyis különböző típusú, logikailag összetartozó elemek összessége. Tudjuk hogy még a legfejlettebb programozási nyelvek sem képesek ilyen összetett adatstruktúrák kezelésére.
A legtöbb általános célú programozási nyelv alapértelmezésben halmazokat egyáltalán nem, vagy csak erős megszorításokkal tud kezelni. Ezért nem várhatjuk azt, hogy az SQL lekérdezések eredményeit közvetlen módon át tudjuk adni hagyományos programváltozóknak.
Ez legfeljebb csak néhány speciális esetben valósulhat meg.
Ennek megfelelően két esetet különböztethetünk meg.

- **Amennyiben a lekérdezés eredménye egyetlen sort eredményez, azt közvetlen módon eltárolhatjuk úgy, hogy minden egyes komponens külön változóba kerül.**
- **Amennyiben a lekérdezés egynél több sort ad vissza, ennek feldolgozása csak lépésenként történhet. Ehhez egy *sormutatót* kell definiálnunk, melynek segítségével végighaladhatunk az eredményreláció sorain. Az egyes sorok komponenseit mindig változókba helyezhetjük, és így azokat fel tudjuk dolgozni.**

Az egy sort eredményező SQL lekérdezésben a sor komponenseit az `INTO` kulcsszó után megadott változólistába helyezhetjük el. Ennek pontos formája a következő:

```
INTO <megosztott változó>[,<megosztott változó>]
```

Több sort eredményező lekérdezések esetén a sormutató kezelése a következő módon történik. Használat előtt a sormutatót deklarálni kell.

```
EXEC SQL DECLARE <sormutatónév> CURSOR FOR
<lekérdezés>
```

A `<sormutatónév>` paraméterben tetszőleges azonosítót adhatunk meg. A későbbiek során **ezzel tudunk hivatkozni az adott sormutatóra.**
Használat előtt **inicializálni** kell a mutatót.

Ez a <lekérdezés> paraméterben leírt SQL utasítással történik. A deklarációs parancs végrehajtása után képzeletben végrehajtódik a megadott SQL utasítás, és az eredmény tábla tartalma hozzárendelődik a mutatóhoz. Ezután a sorokon lépésenként haladhatunk végig
a **FETCH** utasítás segítségével.
Minden egyes lépésben az aktuális sor tartalmát betölthetjük a megadott változókba.
Az utasítás formája az alábbi:

```
EXEC SQL FETCH FROM <sormutatónév> INTO
<változólista>
```

Azt hogy nincs több sor az **SQLSTATE** változó segítségével érzékelheti a program. Ennek egy speciális értékével jelzi a rendszer azt, hogy a sorok feldolgozása teljesen megtörtént.
A sormutató segítségével meghatározott sorra vonatkozóan speciális SQL utasításokat is kiadhatunk. Így például törölhetjük, vagy módosíthatjuk az aktuális sort. Ez a már jól ismert módosító, illetve törlési parancsokkal történik, azzal a változtatással, hogy
a **WHERE** kulcsszó után feltételként a **CURRENT OF** parancsot kell használnunk, az alábbi módon:

```
CURRENT OF <sormutatónév>
```

A parancs hatására a megfelelő művelet a sormutatóval jelzett rekordon hajtódik végre.
Lehetőségünk van a **FETCH** parancsban a sormutató mozgási irányának a megváltoztatására is. Ha ezt akarjuk alkalmazni, akkor a kurzor deklarációjánál a **SCROLL CURSOR FOR** kulcsszavakkal kell jeleznünk. Eddig azt feltételeztük, hogy az utasítás hatására mindig a következő sorra lép a mutató.

Ezt a következőképpen módosíthatjuk:

```
EXEC SQL FETCH PRIOR|NEXT|FIRST|LAST|RELATIVE
<szám>|ABSOLUTE <szám> FROM <sormutatónév> INTO
<változólista>
```

A **PRIOR**
kulcsszó hatására a mutató mozgási iránya ellenkezőjére változik, vagyis nem a következő, hanem az előző sorra lép.
A **NEXT**
az alapértelmezett mozgási irányt, vagyis az előre lépést jelenti.
A **FIRST és a LAST**

parancsok segítségével az első illetve az utolsó sorokra léphetünk.
A **RELATIVE és ABSOLUTE**
kulcsszavak közvetlen pozicionálást tesznek lehetővé. Az előbbinél a pozíció meghatározása mindig az aktuálishoz képest relatív módon történik, méghozzá úgy, hogy a szó után megadott szám hozzáadódik az aktuális sor sorszámához, és az így kapott szám adja meg az új sor számát. Ha az ABSOLUTE kulcsszót alkalmazzuk, akkor a pozicionálás mindig a tábla elejéről, illetve végéről történik, attól függően, hogy a megadott szám pozitív, vagy negatív.

Végül a sormutatót a CLOSE utasítással zárhatjuk le.

EXEC SQL CLOSE <sormutatónév>

Az eddigiekben csak olyan esetekkel foglalkoztunk, amelyeknél az SQL parancs a fordítási időben már ismert volt.
Elképzelhető, hogy
a lekérdezéshez használt SQL parancsot csak futási időben ismeri meg a rendszer.
Lehetséges például, hogy a felhasználó gépeli be a parancsot direkt módon. Ilyenkor a megadott utasítást értelmezni kell, és amennyiben az helyes, akkor végre kell hajtani.
Ezeket a lépéseket az úgynevezett
dinamikus SQL utasítások segítségével végezhetjük el.
Ennek formája az alábbi:

EXEC SQL PREPARE <lekérdezésnév> FROM <kérdés>

Ez a parancs egy lekérdezés előkészítését végzi. A <kérdés> paraméterben megadott SQL utasítást értelmezi, amennyiben azt helyesnek találja, akkor előkészíti a végrehajtást.
A <lekérdezésnév> paraméterben egy logikai azonosítót rendelhetünk a kérdéshez. Ezzel hivatkozhatunk a végrehajtás során a megfelelő előkészített lekérdezésre. A végrehajtás a következőképpen történhet:

EXEC SQL EXECUTE <lekérdezésnév>

A fenti két lépést össze is vonhatjuk egybe. Tulajdonképpen a szétválasztása csak amiatt logikus, hogy amennyiben többször szeretnénk végrehajtani ugyanazt a kérdést, akkor nem kell csak egyszer értelmezni. A továbbiakban már csak a gyorsan végrehajtható lekérdezést kell futtatni. Amennyiben erre nincs szükség, akkor a következőképpen járhatunk el:

EXEC SQL EXECUTE IMMEDIATE <kérdés>

41.Példa

Adjuk meg azt a C nyelvű programot, amely a Dolgozó tábla alapján a név és fizetés adatokat kilistázza a képernyőre és összesíti a listában található összegeket!

```
#define TABLA_VEGE "02000"
void jovedelem()
{
    long Szumma;

    EXEC SQL BEGIN DECLARE SECTION;
        long Jovedelem;
        char Nev[50];
        char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE DolgozoSormutato CURSOR FOR
        SELECT 'A dolgozó neve',
            'A dolgozó fizetése' FROM Dolgozó;

    Szumma = 0;
    while (!strcmp(SQLSTATE,TABLA_VEGE))
    {
        EXEC SQL FETCH FROM DolgozoSormutato
            INTO :Nev,:Jovedelem;
        printf("%s,%ld\n",Nev,Jovedelem);
        Szumma += Jovedelem;
    }
    printf("Összesen:%ld\n",Szumma);

    EXEC SQL CLOSE DolgozoSormutato;
}
```

A programrészletben szereplő TABLA_VEGE konstans arra szolgál, hogy ellenőrizni tudjuk, végigolvastuk-e már a teljes táblázatot. A SELECT illetve a FETCH utasítások úgy módosítják az SQLSTATE változó tartalmát, hogy amennyiben nincs, vagy már elfogytak a feldolgozható sorok, akkor annak értéke "02000" lesz. A fő ciklus feltételének vizsgálatkor ebből dönti el a program, hogy kell-e tovább dolgoznia. A DolgozoSormutato azonosítóval

adjuk meg a tábla sormutatóját, amelyet aztán a FETCH utasítás használ. A Szumma változó az összegzésre való, a végén ebben található a végösszeg, amit ki is ír a program. A Nev és a Jovedelem megosztott elérésű változók, amelyekbe a tábla sorainak komponensei kerülnek.

42.Példa

Adjuk meg azt a C nyelvű programot, amely a Dolgozó tábla alapján a fizetési adatokat úgy módosítja, hogy akinek a fizetése 50000 Ft alatti, annak a billentyűzetről beolvasott fizetési adatot írja be!

```
#define TABLA_VEGE "02000"
#define JOVEDELEM_HATAR 50000

void jovedelemmodositas()
{

    long Szumma;

    EXEC SQL BEGIN DECLARE SECTION;
        long Jovedelem;
        char Nev[50];
        char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE DolgozoSormutato CURSOR FOR
        SELECT 'A dolgozó neve',
            'A dolgozó fizetése' FROM Dolgozó;

    Szumma = 0;
    while (!strcmp(SQLSTATE,TABLA_VEGE))
    {
        EXEC SQL FETCH FROM DolgozoSormutato
            INTO :Nev,:Jovedelem;
```

```
        if (Jovedelem < JOVEDELEM_HATAR)
        {
            printf("%s új jövedelme:");
            scanf("%ld\n",&Jovedelem);
            EXEC SQL
            UPDATE  SET  'A  dolgozó  fizetése'  =
            :Jovedelem
            WHERE CURRENT OF DolgozoSormutato;
        }
    }

    EXEC SQL CLOSE DolgozoSormutato;
}
```

A függvény felépítése hasonló az előző példában találhatóhoz. Új konstans a JOVEDELEM_HATAR, amelynek értéke 50000. Ennek vizsgálatával dönti el az eljárás, hogy szükséges-e módosítani a fizetést, és amennyiben igen, akkor beolvassa az új adatot, és az UPDATE utasítással módosítja az eredeti táblában a sormutató által jelzett sor tartalmát. Ezt a feltételben a WHERE CURRENT OF kulcsszavakkal adhatjuk meg.

43.Példa

Adjuk meg azt a C nyelvű programot, amely a Dolgozó táblából a névsor szerinti legutolsó dolgozót törli!

```
void utolsotorles()
{

    EXEC SQL BEGIN DECLARE SECTION;
        char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE DolgozoSormutato SCROLL
            CURSOR FOR SELECT * FROM Dolgozó
            ORDER BY 'A dolgozó neve';

    EXEC SQL FETCH LAST FROM DolgozoSormutato;
    EXEC SQL DELETE WHERE CURRENT OF
    DolgozoSormutato;

    EXEC SQL CLOSE DolgozoSormutato;
```

}

Megfigyelhetjük, hogy eltérően az előzőektől a sormutató definiálása a SCROLL CURSOR FOR kulcsszavakkal történik, és a SELECT parancsban a rendezés megvalósítása érdekében használjuk az ORDER BY záradékot. Ezután a FETCH LAST utasítással az utolsó sorra pozícionálunk, amit a DELETE utasítással törlünk, felhasználva ismét a WHERE CURRENT OF kulcsszavakat.

44.Példa

Adjuk meg azt a C nyelvű programot, amely a billentyűzetről beolvas egy SQL utasítást, és végrehajtja azt!

```
#define ROSSZ_UTASITAS "20000"

void SQLfgv()
{
    EXEC SQL BEGIN DECLARE SECTION;
        char InputUtasitas[80];
        char SQLSTATE[6];
    EXEC SQL END DECLARE SECTION;

    printf("Kérem az SQL utasítást:");
    gets(InputUtasitas);
    EXEC SQL PREPARE SQLutasitas FROM
    :InputUtasitas
    if (strcmp(SQLSTATE,ROSSZ_UTASITAS))
        EXEC SQL EXECUTE SQLutasitas;
    else
        printf("Az utasítás nem megfelelő\n");
}
```

Az eljárásban deklarált megosztott elérésű InputUtasitas nevű változó fogja tartalmazni a felhasználó által megadott SQL parancsot. Beolvasása a C nyelvben szokásos gets() függvénnyel történik az input eszközről. Ezután a program elemzi a beírt utasítást a PREPARE paranccsal, és amennyiben az SQLSTATE változó a "20000" értékkel reprezentált "helytelen utasítás" kódot adja vissza, a program kijelzi ezt. Amennyiben a felhasználó által megadott SQL utasítás helyes, az EXECUTE parancs végrehajtja azt.