Co to jest embedded SQL?

Zacznijmy od tego, że stosowane w skrypcie określenie „zagnieżdżony SQL" nie jest formalnie poprawne. Zagnieżdżeniem jest powiem umieszczanie jednej instrukcji w drugiej, tak jak na przykład robiliśmy to w tym przypadku:

```
SELECT id FROM samochod WHERE id_kolor IN
        (SELECT id FROM kolor)
```

To jest przykład instrukcji SQL zagnieżdżonej w innym poleceniu.

Zajmijmy się jednak embedded SQL, czyli SQL osadzonym. Jak pamiętamy, wyniki zwracane przez wszelkie zapytania to tabele, czyli zbiory krotek. W wielu językach programowania odpowiada takiej strukturze typ tablicowy. W językach tych, dostęp do tablic zapewniają nam indeksowane pola.

Wyobraźmy sobie, że chcielibyśmy wypisać te elementy tablicy, dla których numer wiersza jest liczbą podzielną przez 3. Na przykład, kod w C, który realizuje to zadanie wyglądałby tak:

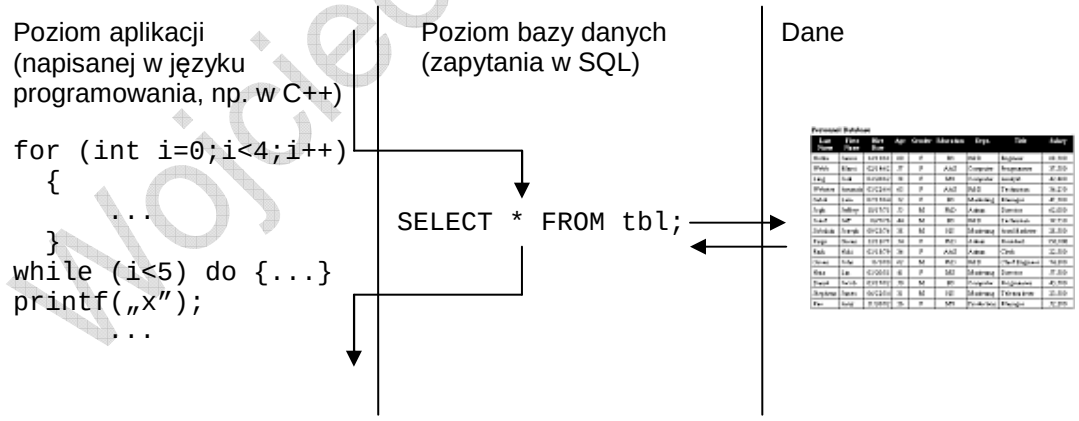
```
int tablica[10];
for (int i=0; i<10; i++) {
        if (i % 3 ==0) printf (tablica[i]);
};
```

Podobnie zresztą w php:

```
tablica = array();
for (int $i=0; $i<10; $i++) {
        if ($i % 3 ==0) echo ($tablica[$i]);
};
```

W przypadku SQL zadanie to jest utrudnione (choć nie niemożliwe). Okazuje się jednak, że w tym przypadku języki wysokiego poziomu, w których pisane jest oprogramowanie radzą sobie znacznie lepiej. W tym miejscu najpełniej ujawnia się fakt: formalnie, SQL NIE JEST językiem programowania (jest to tylko język zapytań do baz danych).

Powstaje pytanie, jak połączyć poziom języka programowania z poziomem zapytań? Chcielibyśmy bowiem uzyskać coś takiego:




Innymi słowy chcielibyśmy podczas pracy aplikacji „wyskoczyć na chwilę do SQL" i pobrać z bazy danych wynik jakiegoś zapytania. Oczywiście trzeba pamiętać, że nie mamy w tej chwili już dostępu do aplikacji klienckiej (takiej jaką jest programik mysql, z którego korzystaliśmy do połączenia z bazą mySQL).

Generalnie, wyjścia są dwa. Po pierwsze, można do naszej aplikacji dorzucić bibliotekę programistyczną, która będzie pełniła rolę „klienta". Taka biblioteka jest de facto ROZSZERZENIEM naszego języka programowania o nowe instrukcje.

Takie rozwiązanie jest bardzo wygodne, bo pozwala stosować znane w języku programowania podejście do zmiennych i innych obiektów. Na przykład, w języku C++ w dialekcie opracowanym przez Borlanda, można (po dodaniu za pomocą #include) stworzyć obiekt typu TDatabase i korzystając z jego metod pracować z bazą danych. W dużym uproszczeniu mogłoby to wyglądać na przykład tak:

```
#include „vcl.h”
TDatabase* BazaStudentow = new TDatabase();
BazaStudentow->User = ”root”;
BazaStudentow->Password = ”kidaj;345k8”;
...
BazaStudentow->Open();
...
```

Tego typu bibliotek zwykle potrafią uniezależnić się od silników bazodanowych, z którymi się łączą. Powstała pewna wspólna, uzgodniona i zestandaryzowana metoda, która wymaga od serwerów bazodanowych podobnego mechanizmu łączenia się. Metoda ta to oczywiście platforma ODBC (Open Database Connectivity). Wówczas biblioteka ODBC dla danego języka programowania zapewnia możliwość połączenia się ze „standardową bazą ODBC”, zaś sam serwer bazy powinien „umieć” zachowywać w zgodzie z tym standardem (większość głównych baz danych, w tym mySQL to potrafi).

Zostawmy jednak te rozważania i zajmijmy się DRUGIM sposobem połączenia z języka programowania z bazami danych. Jest on stosowany bardzo rzadko – jest to bowiem dość przestarzała metoda. Niestety, nasz skrypt jej wymaga ;-)

Metoda druga polega na umieszczeniu w języku programowania swoistego rodzaju preprocesora SQL. Preprocesor to część „serca” języka programowania, która potrafi poradzić sobie (od biedy) ze zrozumieniem instrukcji SQL.

Różnica pomiędzy pierwszym i drugim podejściem jest m.in. taka, że w pierwszym podejściu stosuje się dodatkowe biblioteki dostarczane przez twórców baz danych, zaś w drugim – przez twórców języka programowania – ale to na marginesie. Nawiasem mówiąc – żeby przetestować naszą metodę konieczny jest jakiś interpreter/kompilator innego języka...

W naszym sposobie pojawi się instrukcja EXEC SQL. NIE JEST ona pisana w naszym kliencie SQL, tylko oczywiście w kodzie programu C/C++. Będziemy więc musieli uczyć się trochę na sucho.
Tak, ja TEŻ uważam, że to nienajlepszy pomysł, mówiąc delikatnie....

Załóżmy, że w trakcie działania naszego programu w C, chcemy wywołać instrukcję SQL, która doda nam nowy rekord to tablicy klik. Tablica klik składa się z kolumny cos typu int.

Ułóżmy najpierw nasze zapytanie SQL:

```
INSERT INTO klik (cos) VALUES (1);
```

Załóżmy, że nasz kod w C (na przykład obsługujący naciśnięcie klawisza w okienku aplikacji) wygląda tak):

```
void on_click() {
  ...
  if (x>4) x++;
  ...
  // tu chcemy wykonać SQL
  ...
};
```

Linie powyższego kodu są nieistotne – chodzi tylko oto, aby pokazać, że jest to program napisany w C. Po dodaniu naszego SQL kod będzie wyglądał tak:

```
void on_click() {
  ...
  if (x>4) x++;
```

```
  ...
  EXEC SQL INSERT INTO klik (cos) VALUES (1);
  ...
};
```

Hmmm..... nasza tablica wygląda nieco dziwnie. W dodatku, jak widać wymusiliśmy wpisanie do niej wartości 1, a przecież na pewno chcielibyśmy mieć możliwość wpisywania wartości, którą ustalimy nie w trakcie pisania programu, ale w trakcie jego wykonania! Taką wartością mógłby być na przykład numer użytkownika, który kliknął klawisz (numery użytkowników aplikacji – nie bazy danych!).

Sęk w tym, że preprocesor SQL, który jest odpowiedzialny za działanie linijki EXEC SQL jest bardzo prosty i stosuje inny zapis zmiennych niż C. Zauważmy też, że wstawka EXEC SQL jakby „zmienia świat" programy w C na „świat" programu SQL (świat oznacza konieczność stosowania charakterystycznej składni):

```
void on_click() {                                  // tu jest C
  ...                                              // tu jest C
  if (x>4) x++;                                    // tu jest C
  ...                                              // tu jest C
  EXEC SQL INSERT INTO klik (cos) VALUES (1);      // ... a tu jest SQL!
  ...                                              // tu jest C
};
```

Trzeba więc wykonać nieco ekwilibrystyczny manewr: utworzyć w „świecie C" zmienną (robiąc to tak, jak robi się w C), potem utworzyć zmienną w świecie SQL (robiąc to w charakterystyczny dla SQL sposób), a następnie powiedzieć kompilatorowi, że chodzi nam o ten sam obiekt (mechanizm ten przypomina trochę stosowanie zmiennych globalnych).

W C zmienną tworzymy tak:
`int user;`
Załóżmy, że mamy funkcję w C, która nadaje numer użytkownikowi, czyli:
`user = nadaj_numer();`

Zmienną w SQL stworzymy nadając jej tą samą nazwę (od razu osadzając ją w C).

```
void on_click() {
  int user;
  user = nadaj_numer();

  ...
  if (x>4) x++;
  ...
  EXEC SQL BEGIN DECLARE SECTION;
    int user;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL INSERT INTO klik (cos) VALUES (:user);
  ...
};
```

Jak widać odwołanie do zmiennej w EXEC SQL realizujemy używając dwukropka :user

Działanie tego kodu będzie następujące: w momencie, gdy użytkownik kliknie przycisk, aplikacja wywoła funkcję on_click(). Zmiennej (w C) user zostanie przypisana wartość całkowita (na przykład 124234 – zależnie, jak działa funkcja nadaj_numer()). Następnie wykona się EXEC SQL definiujący zmienną SQL o nazwie user. Ponieważ na „wyższym poziomie" (czyli w C) jest zmienna o tej samej nazwie, nowa zmienna SQL zostanie zainicjowana wartością zmiennej z C. Wreszcie wykona się EXEC SQL zawierający INSERT, a w miejsce :user zostanie wpisana aktualna wartość zmiennej. Uff...

W naszym przypadku użyliśmy zmiennych współdzielonych (bo istnieją na poziomie C i na poziomie SQL) do przesłania wartości do „części SQL” (chodzi o osadzony mikroprogramik). Można także przesyłać wartość w drugą stronę (z SQL do C), stosuje się wówczas składnię

SELECT coś_tam INTO zmienna

Załóżmy, że po kliknięciu na klawisz chcemy teraz uzyskać (i przesłać do programu w C) nazwisko studenta o określonym numerze albumu:

```
void on_click() {
  char student[50];
  int album = tu_jakas_funkcja_wyznacza_numer_albumu();
  ...
  if (x>4) x++;
  ...
  EXEC SQL BEGIN DECLARE SECTION;
    char student[50];
    int album;
  EXEC SQL END DECLARE SECTION;
  EXEC SQL SELECT nazwisko FROM STUDENT WHERE album = :album INTO :student;
  if  (nazwisko=””) mesg(”Nazwisko tego studenta nie jest ustawione”);
  ...
};
```

Niestety, taka metoda nie zawsze jest tak prosta. W rzeczywistości może być ona zastosowana tylko wtedy, jeśli w wyniku zapytania zwrócona zostanie jedna wartość (a nie tabela).

Dla zapytań zwracających tabele (czyli zbiory danych) konieczne jest stosowanie kursorów – o czym opowiemy sobie w następnych odcinku...