# PHP4 és PostgreSQL: komoly webes alkalmazások készítése nyílt forráskódú programeszközökkel

Tim Perdue végigvezet bennünket egy egyszerű webes alkalmazás készítésének lépésein, hogy bemutassa a PHP és a PostgreSQL képességeit.

Nem is olyan rég egy komoly webalkalmazás könnyű felépítése egyet jelentett kemény pénzösszegek kipengetésével egy nagyobb rendszerért (mint amilyen például a Cold Fusion), valamilyen üzleti adatbázis-kezelőért (például a Sybase) és egy Sun kiszolgálóért. Szerencsére, ezeknek a napoknak immár vége van. Az Apache felemelkedése és a jelenlegi ingyenes adatbázis-kezelők kifejlődése végre valós (esetleg még jobb) választást kínálnak az üzleti alkalmazások mellett.

A nyílt forráskód legjobb hajtásai: egy Perl-szerű parancsnyelv, a PHP és egy hatékony objektumközpontú adatbázis-kezelő, a PostgreSQL. Ha összekapcsoljuk a PHP-t és a PostgreSQL-t, szinte bármit felépíthetünk az egyszerű vendégkönyvtől kezdve a hatalmas webalapú bejelentkezési rendszerekig. A PHP biztosítja az agyat, míg a Postgres adja az erőt.

Be szeretnék mutatni egy nagyon egyszerű PHP vásárlási kártya- és leltárrendszert, ami kihasználja a Postgres tranzakciós képességeit. A honlapomon (PHPBuilder.com) letöltheted a forráskódot, és egyéb példaprogramokat is találsz.

Az első dolog, amivel foglalkozni szeretnék, az alkalmazás szerkezete. Minden webalkalmazásban, amit csak PHP-ben fejlesztek, első lépésként mindig létrehozok egy átfogó könyvtárat, amit aztán a webhely minden lapjáról becsatolhatok. Ezt a könyvtárat common.php-nak neveztem el, és az include nevű könyvtárban található meg. A programkönyvtár segít elvégezni az olyan megszokott feladatokat, mint például az adatbázishoz való csatlakozást, a felhasználó kilétének megállapítását, a lap fejlécének és láblécének beállítását stb. Azáltal, hogy ezeket az eljárásokat egy helyen tartjuk, az alkalmazásunk sokkal átláthatóbb és könnyebben alakítható lesz. Például ahelyett, hogy mindenhol közvetlenül a kódba helyeznénk el az adatbázis-kapcsolat létrehozását, csak egyszer írjuk meg, magában a programkönyvtárban.

Nos, a könyvtárunk első változata máris használható. Kapcsolatot teremt az adatbázissal, sőt, néhány egyszerű HTML-absztrakciót is ad számunkra. Ahogy a honlap HTML része egyre bonyolódik, a legtöbb részt beszúrhatjuk a fejlécbe/láblécbe (site_header/site_footer) ezáltal egyszerre végezve el a változtatást a teljes alkalmazásban.

A könyvtár egy egyszerű absztrakciós réteget is nyújt a Postgres-lekérdezésekhez. Ahhoz, hogy csökkenteni tudjam a megírandó kód mennyiségét, a lehető legegyszerűbb módon készítettem el. Végül pedig meghívja a PHP4 beépített munkafolyamat kezelőkódját. A session_start meghívása arra kéri a PHP4-et, hogy töltse újra a bejegyzett változókat, és így azok elérhetők lesznek a teljes alkalmazás alatt. Honlapunk minden lapja alapvetően valahogy így fog kinézni:

```
?php

// common programkönyvtár használatba vétele
require ($DOCUMENT_ROOT.'/include/common.php');

echo site_header('Példa honlap');

/*
       a lap belső logikája
*/

echo site_footer();

?
```

Általában minden alkalmazás tervezésekor érdemes annyira elválasztani a gerincet alkotó belső logikát az éppen időszerű megjelenítéstől (jelen esetben a HTML-től) amennyire csak lehetséges. Én általában a belső logika részeit funkcióhívások belsejébe rejtem, amelyeket aztán a webhely bármely lapjáról csatolni és hívni lehet. Ez azért előnyös, mert esetenként előfordulhat, hogy más csatolófelületeket is szeretnél az alkalmazáshoz illeszteni – talán épp egy különlegesen könnyű súlyú felületet a vezeték nélküli alkalmazások számára. Ha a logika elemei a HTML-megjelenítés részeként szerepelnének a kódban, meg kellene ismételni a teljes logikát minden egyes felülethez. Ha viszont függvénykönyvtárakba választjuk szét, minden felület ugyanazt a logikát használhatja.

A PHP függvénykezelésével csak egy gond van: nincs általános kivételkezelő eljárás. Ha egy belső hiba történik egy függvényben, honnan fogja tudni ezt a hívó kód, hogy figyelmeztethesse a felhasználót? Más nyelvekben, mint például a Javában berakhat-


A webkiszolgáló és az ügyfél közti kapcsolat

nád a kivételt a függvény belsejébe munkafolyamatként (method). Amikor meghívod a Java munkafolyamatot, átirányíthatnád egy try/catch utasítással, s így a gond meg is oldódna.
PHP-ban általában úgy oldom meg ezt a nehézséget, hogy minden függvény visszatérési értéke igaz vagy hamis, és mindig beállítom a `$feedback` (visszajelzés) nevű globális változót. Az eredmény így lekérdezhető, és a $feedback tartalma szükség esetén a képernyőre írható. Van egy PEAR ➲ http://pear.php.net/ néven futó ilyen irányú kezdeményezés, amely megpróbálja egységesíteni többek közt a hibakezelést és az adatbázis-elérést, de jelenleg még nem igazán terjedt el.
Íme egy példa, hogyan hívhatunk függvényeket az általam megadott igaz/hamis módszerrel:

```
<?php

$eredmeny=fuggvenynev();

if (!$eredmeny) {
        // hiba történt, jelenítsd meg
        echo $feedback;
} else {
        // folytasd, ha sikeres volt
}

?>
```

Nos, lássuk akkor a bevásárlókosarunkat. Szükségünk lesz néhány alapvető adatszerkezetre, amiben elraktározhatjuk a kosár adatait: kezdésként csupán egy készletadatbázisra, ami felsorolja a nevet, az alkatrészszámot, az árat és a készlet mennyiségét. Ezenkívül nélkülözhetetlen a vásárlóink és az általuk vásárolt áruk követése. Ez már elég összetett feladat lesz mindnyájunknak.
Ennyi elég ahhoz, hogy egy csökevényes virtuális bevásárlókosarat készítsünk. Az adatbázis séma egységesítése céljából külön táblát készítettem, ez felsorolja a vásárló kosarának tartalmát. Így, hogy a vásárló többször tehet árut a kosarába, emellett egyszerű módon csatolhatjuk kosarának tartalmát a készletadatbázishoz.
Most ideje meggondolni, milyen műveleteket szeretnénk elvégeztetni a virtuális boltunk függvényeivel. A legalapvetőbb feladatuk: új kosár kérése, az áru kosárba helyezése, valamint a kijelentkezés. Egy valódi hálózati boltban természetesen sokkal több dologra lenne szükség. Ilyen például az áruk közötti válogatás lehetősége, a mennyiség megváltoztatása és a többi megszokott tevékenység. Ezeket a feladatokat azonban önökre bízom.
Egy egyszerű feladattal kezdem, ezzel egy új vásárlót lehet létrehozni. A következők tartoznak ide: az általunk készített vásárló (costumer) szekvencia következő értékének lekérése, ezen érték beszúrása a costumer táblába, majd pedig az érték felvétele a PHP4 beépített folyamatkezelő kódjába.
Ez kicsit több kódot vett igénybe, mint szerettem volna, viszont bemutatja, miként kell helyesen elindítani és befejezni egy tranzakciót Postgresben, és hogyan kell ellenőrizni a lekérdezések hibamentességét. Ugyanazt a rutint fogom hibaellenőrzésre használni a teljes kód során, mint amit mindenkinek ajánlok.
Mindig tervezzük meg, miként fogjuk kezelni a helyzetet, ha a lekérdezés sikertelen. Teljesen leállítjuk a parancsfájlt, újra próbáljuk a lekérdezést, vagy csak egyszerűen továbblépünk, mintha mi sem történt volna? Óvatosan mérlegeljük az összes lehetőség hatását. Például, ha nem tudunk lekérni a következő vásárló azonosítóját, nem igazán kellene továbblépni az új vásárlórekord készítésére. Ha a vásárlórekord létrehozása sikertelen, később nem tudjuk frissíteni a cím adatait, és nem tudunk árut rakni a kosarába.

## Csak logikusan, rendben?

Most lássuk az áru kosárba helyezésének munkafolyamatát. Ez szintén elég egyszerű eset. Mielőtt beraknánk egy árut a kocsiba, előbb ellenőrizni kell, létezik-e egyáltalán ilyen azonosító az adatbázisban. Ezt gyakorlati megfontolásokból tesszük, mivel az áruazonosító a böngészőtől érkezik, amibe így könnyen belepiszkálhattak.
Ha egyszer tudjuk, hogy az azonosító létezik, megnézhetjük, hogy már a kocsiban van-e. Ha már van, akkor megnöveljük a mennyiségét új sor beszúrása helyett. Ha viszont nincs még a kocsiban, akkor beszúrjuk a bevásárlókocsiba az alapértelmezett egy mennyiséggel.
Most már tudunk új vásárlókat létrehozni, és árut rakni a kocsijukba. Már csak arra van szükségünk, hogy a vásárlók ki tudjanak jelentkezni a boltból, és emellett megfelelően csökkentsük a készleteket.
Ez a legbonyolultabb része az egész boltnak, és most jó hasznát vesszük a Postgres tranzakcióinak, valamint fejlett zárolási rendszerének. Kezdetnek a Postgres SELECT...FOR UPDATE szintaxisát fogjuk használni, ami tulajdonképpen lezárja a kiválasztott sorokat, így

*1. lista* Példa a programkönyvtár kódjára

common.php:

```
<?php

//Csatlakozás a PHP adatbázishoz.
$conn=pg_pconnect("user=tim dbname=db_example");

//ellenõrizzük, sikeres volt-e a csatlakozás
if (!$conn) {
        //csatlakozási-hiba lap
        //hiba esetén is folytathatod a munkát
        //az adatbázis nélkül. Tõled függ.
        echo pg_errormessage($conn);
        exit;
}

//Állítsunk be valami általános fejlécet.

function site_header ($title) {
        return '<HTML>
        <HEAD>
        <TITLE>'.$title.'</TITLE>
        </HEAD>
        <BODY>';
}

//Egy közös HTML-rész a lapok végére

function site_footer () {

        return '</BODY></HTML>';

}

//Egy egyszerû rövidítés
//minden postgres lekérdezéshez.

function query($sql) {
        global $conn;
        return pg_exec($conn,$sql);
}

// PHP4 utasítása, hogy automatikusan minden lapon
// beállítsa/visszatöltse a megfelelõ
// munkafolyamat-állapotot.
session_start();

?>
```

*2. lista* Bevásárlókosár-adatszerkezetek

```
Cart.sql:

# Egy szekvencia segítségével hozzuk létre
# a felhasználói azonosítókat.
# Nagyobb lépésközt állítottam be, hogy
# csökkentsem az esélyét annak, hogy a fel-
# használók egymás azonosítóját kitalálják.
#
#
create sequence seq_customer_id
       increment 26 start 1;

create table customers (
customer_id int not null default 0 primary key,
name text,
address text,
credit_card text,
total_order MONEY DEFAULT '$0.00'
);

create table cart_items (
cart_item serial,
customer_id int,
part_number int,
quantity int
);

create index idx_cart_customer
       on cart_items(customer_id);

create table item_inventory (
part_number serial,
name text,
price float,
inventory int
);
```

frissítheted őket, és elküldheted a változásokat a tranzakcióval. Ennek a szintaxisnak a tranzakción belüli használatával biztos lehetsz afelől, hogy az adatok konzisztensek maradnak. Némely adatbázis-kezelőben, például a MySQL-ben, könnyen zárolhatsz megadott adatsorokat, hogy megakadályozd a többi folyamatot abban, hogy csökkentsék a készleteket, amikor magad is ugyanezt szeretnéd tenni.

Ez a lekérdezés alválasztásokat (subselect) is használ, ez szintén egy általános lehetőség a hatékonyabb adatbázis-kezelők közt. Az alválasztások lehetővé teszik, hogy két lekérdezést foglalj egybe, hogy megkönnyítsd az életed.

Miután zároltuk a sort, egy lekérdezést kell végrehajtanunk, hogy csökkentsük a készletszámot a kocsiban lévő minden egyes áru esetén. Az egyszerűség kedvéért, nem figyelmeztetünk a készletek kimerülésekor, de sikeresen állítjuk negatív értékre az áruszámlálót (item_count), ha a készlet mennyisége nulla alá csökken. Esetleg felállíthatsz egy vezérlőlapot, amin megnézheted a negatív darabszámú árukat, és rendelhetsz utánpótlást.

Végül, szeretnénk kibővíteni a vásárlótáblát a látogató hitelkártyaszámával, szállítási adataival, valamint a teljes eladási mennyiséggel, majd pedig megsemmisítjük a PHP4 vásárló munkafolyamatot. Íme egy viszonylag összetett tranzakció, amelynek minden lépését megfelelően kell végrehajtani. Ha ez nem történik meg, mindent vissza kell állítani az eredeti helyzetbe.

Ha nem részesültél a példámban szereplő tranzakciók élvezetében, és a lekérdezés sikertelen volt valamelyik készletadat módosítása

*3. lista* Új vásárló bejegyzése

```
<?php

function cart_new() {
   global $conn, $customer_id, $feedback;

   //Nyitunk egy tranzakciót.
   query("BEGIN WORK");

   //A következõ azonosító lekérdezése.
   $res=query(
        "SELECT nextval('seq_customer_id')");

   //A hibák ellenõrzése.
   if (!$res || pg_numrows($res)<1) {
         $feedback .= pg_errormessage($conn);
         $feedback .= ' Hiba - Nem kaptuk meg a
         ➥következõ azonosítót';
         query("ROLLBACK");
         return false;
   } else {
         //A visszatérési érték elmentése egy
         //helyi változóban.
         $customer_id=pg_result($res,0,0);

         //Az azonosító bejegyzése a PHP4-gyel.
         session_register('customer_id');

         //Az új vevõhöz tartozó sor beszúrása.
         $res=query("INSERT INTO customers
         ➥(customer_id) VALUES
         ➥('$customer_id')");

         //Hibák ellenõrzése.
         if (!$res || pg_cmdtuples($res)<1) {
               $feedback .=
                     pg_errormessage($conn);
               $feedback .= ' Hiba - nem tudtam
               ➥bejegyezni az új vásárlót';
               query("ROLLBACK");
               return false;
          } else {
               //A tranzakció érvényesítése.
               query("COMMIT");
               return true;
          }
      }
}

?>
```

közben, akkor bizony sok gondod lesz. Szét kell választanod a készletet, megváltozott és változatlan részekre. Ha a látogató újra megkísérli elérni a lapot, honnan fogod tudni, melyik készletadatot kell ismét csökkenteni? A válasz: sehonnan, bizony egy pontatlan készletet kapsz eredményül.

Ennek a cikknek nem célja, hogy teljes körű vásárlókocsi-leírást adjon – egy egész könyvet írhatnék erről, ha lenne elég időm. A cél tulajdonképpen annyi, hogy bemutassuk a háttérmunkákat, és szemléltessük azt a tervezési és végrehajtási módszertant, amit minden webfejlesztőnek tanácsolok. További adatokért keresd a ➲ PHPBuilder.com lapot.

Az a néhány rész, amit kihagytam, főleg a kocsi tartalmának megtekintésével, illetve az árukészlet böngészésével foglalkozik. Ezek a részek, az itt található kódokkal együtt, zipfájlban elérhetők a ➲ http://www.phpbuilder.com/columns/linuxjournal200009.php3L honlap vitaasztalán (discussion/comment board).

*4. lista* Új elem kosárba helyezése

```
<?php

function cart_add_item($item_id,$quantity=1) {
   global $customer_id, $feedback, $conn;

   //Egy tranzakciót kell nyitni, hogy csak egy
   //lekérdezés módosíthassa az adatbázist.

   //Lekérdezzük a termékek közül a
   //következõt.
   $res=query("SELECT * FROM item_inventory
               ➥WHERE part_number='$item_id'");

   //A hibák ellenõrzése.
   if (!$res || pg_numrows($res)<1) {
      $feedback .= pg_errormessage($conn);
      $feedback .= ' Error-item not found ';
      return false;
   } else {
      //Megnézzük, hogy az elem már a kosárban
      //van-e. Ha igen, megnöveljük a mennyiséget.

      //Tranzakciót kezdünk, hogy zároljuk a
      //sorokat, ha megtaláljuk azokat.
      query("BEGIN WORK");

      $res=query("SELECT * FROM cart_items ".
          "WHERE part_number='$item_id' AND
          ➥customer_id='$customer_id' FOR
          ➥UPDATE");

      //A hibák ellenõrzése.
      if (!$res || pg_numrows($res)<1) {
          //Rakjuk a kosárba!

             $res=query("INSERT INTO cart_items ".
             "(customer_id,part_number,quantity)".
             "VALUES
             ➥($customer_id,$item_id,$quantity)");

             //Volt a beszúrásnál hiba?
             if (!$res || pg_cmdtuples($res)
                 ➥< 1) {
                $feedback .=
                pg_errormessage($conn);
                $feedback .= ' Hiba - A kosárba
                ➥helyezés nem sikerült';

                //Semmi sem változott, mégis jobb,
                //ha érvénytelenítjük az egész
                //tranzakciót.
                query("ROLLBACK");
                    return false;
                } else {
                    query("COMMIT");
                    return true;
                }
          } else {
                //Az elem már jelen volt.

                //Mennyiség.
                $res=query("UPDATE cart_items
                ➥SET quantity = quantity +
                ➥$quantity WHERE part_number=
                ➥'$item_id'AND customer_id=
                ➥'$customer_id'");

                if (!$res || pg_cmdtuples($res)
                    ➥< 1) {
                   $feedback .=
                   ➥pg_errormessage($conn);
                   $feedback .= ' Hiba - nem tudom
                   ➥megnövelni a mennyiséget';

                   //Most sem történt semmi.
                   query("ROLLBACK");
                   return false;
             } else {
                   //A változtatott mennyiség
                   //érvényesítése.
                   query("COMMIT");
                   return true;
                }
             }
       }
}

?>
```

*5. lista* Ellenőrzés és a lista csökkentése

```
<?php

function cart_checkout($credit_card,$address,$name) {
global $conn, $customer_id, $feedback;

//Tranzakciót kezdünk.
query("BEGIN WORK");

/*
  A vásárló kosara alapján zároljuk a megfelelõ
  sorokat.

  Ezt egy egyszerû subselecttel oldjuk meg.
*/
$sql="SELECT * FROM item_inventory ".
  "WHERE part_number ".
  "IN (SELECT part_number FROM cart_items ".
  "WHERE customer_id='$customer_id') ".
  "FOR UPDATE";
$res=query($sql);

//A hibák ellenõrzése.
if (!$res || pg_numrows($res)<1) {
  //Nincs megfelelõ adat, vagy adatbázishiba.
  $feedback .= pg_errormessage($conn);
  $feedback .= ' Hiba - nem zároltunk elemeket ';

  //A tranzakció megszakítása.
  query("END WORK");
  return false;
} else {
  /*
        A megfelelõ sorokat zároltuk.

        Az adatok kiolvasása a kosárból.
  */
```

*5. lista (folytatás )*

```
   $sql="SELECT part_number,quantity ".
        "FROM cart_items ".
        "WHERE customer_id='$customer_id' ".
        "ORDER BY part_number DESC";
  $res2=query($sql);

  //A hibák ellenõrzése.
  if (!$res2 || pg_numrows($res2)<1) {
  //Nincs kiválasztott elem.
        $feedback .= pg_errormessage($conn);
        $feedback .= ' Hiba - üres a kosár ';

        //terminate the transaction
        query("END WORK");
        return false;
  } else {
        $rows=pg_numrows($res2);
        /*
          Zároltuk a megfelelõket, és ismerjük a
          kosár tartalmát.

          Végigmegyünk rajta, és frissítjük az
          egyenlegeket.
        */

              for ($i=0; $i < $rows;$i++) {
              //A következõ sor.
$quantity=pg_result($res2,$i,'quantity');

$item_id=pg_result($res2,$i,'part_number');

             $res3=query("UPDATE item_inventory".
             "SET inventory =inventory-$quantity ".
             "WHERE part_number='$item_id'");

             //Megnézzük, hogy volt-e hiba a le-
             //kérdezésnél, vagy volt-e egyáltalán
             //feldolgozott sor.
             if (!$res3 ||pg_cmdtuples($res3) < 1)
{
             //Ezt a sort nem tudtuk frissíteni.
             $feedback .=pg_errormessage($conn);
             $feedback .= ' Hiba - Nem tudtuk
             ➥frissíteni az egyenleget';

             //A tranzakció érvénytelenítése.
             query("ROLLBACK");
             return false;
              }
         }

         /*
              A frissítés készen van.

              Végül számoljuk ki a végösszeget, és
              a kapott adattal frissítsük a vásárló
              rekordját.
         */
         $res=query("SELECT sum(cart_items.
         ➥quantity*item_inventory.price) ".
         "FROM cart_items,item_inventory ".
         "WHERE cart_items.customer_id=
         ➥'$customer_id' ".
         "AND cart_items.part_number=item_
         ➥inventory.part_number");

          //A hibák ellenõrzése.
          if (!$res || pg_numrows($res) < 1) {
          //Nem kaptuk meg a rendelés összegét.
              $feedback .= pg_errormessage($conn);
              $feedback .= ' Hiba - nem kaptuk
              ➥meg a rendelés végösszegét';

              //A tranzakció érvénytelenítése.
              query("ROLLBACK");

         } else {
         /*
              Most már tudjuk a végösszeget,
              frissítjük a vásárló adatait.
         */

              //Az összeget eltároljuk egy helyi
              //változóban.
              $total=pg_result($res,0,0);
              $res=query("UPDATE customers ".
              "SET address='$address',
              ➥name='$name',".
              "total_order='$total',credit_card=
              ➥'$credit_card' ".
              "WHERE customer_id='$customer_id'");

              //Hibaellenõrzés.
                 if (!$res || pg_cmdtuples($res) \
                      < 1) {

              //A frissítés nem mûködött, vagy nem
              //volt frissítendõ sor.
                 $feedback .=
                 ➥pg_errormessage($conn);
                 $feedback .= ' Hiba - a vásárló
                 ➥adatainak frissítésekor ';

                  //A tranzakció érvénytelenítése.
                 query("ROLLBACK");
                 return false;
            } else {
                  /*
                  Készen vagyunk!
                  Most írjuk ki a változásokat
                  az adatbázisba.
                                        */

                  //Minden változás kiírása.
                  query("COMMIT");

                  //A PHP4 munkafolyamat törlése.
                  $customer_id=0;
                  session_destroy();

                           }
                  }
             }
      }
}

?>
```

*Tim Perdue* (tim@perdue.net) Kaliforniában, Sunnyvale-ben él, a SourceForge.net vezető mérnöke, a PHPBuilder.com és a Geocrawler.com alapítója. Szeret vitorlázni a San Francisco-öbölben, illetve tanulmányozni a csillagokat. Tim és felesége, Lisa harmadik gyermeküket várják.