Gästbok i PHP

Jonas Björk

Oktober 2007

Skicka kommentarer på detta dokument till: jonas@jonasbjork.net .

Licens

Innehållsförteckning

Gästbok i PHP med MySQL

I det här dokumentet kommer vi gå igenom hur vi skapar en gästbok för webben med programmeringsspråket PHP och databasservern MySQL för lagringen av inläggen.

Vi behöver en fungerande MySQL-server och webbservern Apache med PHP-stöd. Om vi inte har tillgång till det får vi installera det först.

Det första vi behöver göra är att tänka till vad det är vi behöver lagra i vår databas. Vi ska skapa en publik gästbok som alla besökare kan skriva i, så vi behöver inte fundera på att skapa någon form av användardatabas med inloggning. Vad vi vill lagra är primärt:

- ✔ Inlägget besökaren skriver.
- ✔ Besökarens namn.
- ✔ Besökarens e-postadress.

Dessutom är det ganska bra att registera vilket datum och vilken tid inlägget gjordes. Om någon postar något elakt kan det vara bra att registera IP-adressen som postade inlägget. Datum, klockslag och IP-adress kan vi lämna till polismyndigheten för att de skall kunna göra en utredning, om någon postar något som bryter mot lagstiftningen.

- ✔ Datum/Tid
- ✔ IP Adress

Nu har vi kontroll på vad vi behöver lagra i vår databas. Nu är det dags att skapa databasen och tabellen som skall lagra inläggen.

Skapa databasen

Vi börjar med att skapa en databas i MySQL om vi inte redan har en:

```
mysql> CREATE DATABASE mindatabas;
Query OK, 1 row affected (0.00 sec)
```

Nu skall vi använda vår databas för att skapa en tabell i den:

```
mysql> USE mindatabas;
Database changed
```

Så där, nu har vi en databas som vi kan jobba med. Nu skall vi ta och skapa tabellen som skall lagra våra inlägg i gästboken.

Skapa tabellen

Tidigare kom vi fram till att vi vill lagra inlägget, namnet, e-postadressen, datum/tid och IP-adressen. Vilka datatyper bör vi använda för lagringen av dessa?

Inägget kan vara ett ganska långt inlägg. Om vi tittar på datatyperna **char** och **varchar** kan de båda rymma max 255 tecken. Detta ger oss ganska lite utrymme att lagra inlägg på. Datatypen **text** rymmer 65,000 tecken och passar oss bättre.

För att lagra namnet och e-postadressen som kan vara ganska korta och sällan är mycket långa väljer vi datatypen **varchar** och begränsar längden till 100 tecken.

Datum och tid kan vi lagra på flera olika sätt. Ett sätt skulle kunna vara att lagra det som ett heltal som representerar UNIX-tid. Det är ganska enkelt och vi kan göra en del roliga saker med det. Trots detta väljer vi datatypen **timestamp**, eftersom den är gjort för just det ändamål vi vill uppnå – att lagra datum

och tid i databaser.

IP-adresser ser ut ungefär så här: 123.123.123.123 . Vi kan snabbt räkna ut att det är tolv (12) siffror och tre (3) punkter i en IP-adress. Totalt blir det 12+3 = 15 tecken. För att lagra IP-adressen väljer vi datatypen **char** och begränsar den till 15 tecken.

Nu har vi kommit fram till vilka datatyper vi behöver använda i vår tabell. Innan vi skapar tabellen skall vi fundera på om det inte vore bra att använda ett fält i tabellen som enbart innehåller en räknare, som ökas med ett (1) för varje post vi lagrar i tabellen. Denna räknare kan vara mycket användbar när vi kommer så långt så att vi skall börja redigera inlägg eller kanske rent av ta bort inlägg i databasen. Vi behöver ett heltal som ökar med ett (1) för varje post som lagras och använder därför datatypen **int** tillsammans med **auto_increment** och sätter fältet som primärnyckel, **primary key**.

Dags att sätta ihop allt och skapa tabellen i databasen:

```
mysql> CREATE TABLE gastbok(id INT AUTO_INCREMENT PRIMARY KEY, inlagg TEXT, namn
VARCHAR(100), epost VARCHAR(100), datum TIMESTAMP, ip_addr CHAR(15));
Query OK, 0 rows affected (0.05 sec)
```

Databasen är skapad och en tabell har vi. Vi kikar på hur den ser ut med:

```
mysql> DESCRIBE gastbok;
+---------+-------------+------+-----+-------------------+----------------+
| Field   | Type        | Null | Key | Default           | Extra          |
+---------+-------------+------+-----+-------------------+----------------+
| id      | int(11)     | NO   | PRI | NULL              | auto_increment |
| inlagg  | text        | YES  |     | NULL              |                |
| namn    | varchar(100)| YES  |     | NULL              |                |
| epost   | varchar(100)| YES  |     | NULL              |                |
| datum   | timestamp   | NO   |     | CURRENT_TIMESTAMP |                |
| ip_addr | char(15)    | YES  |     | NULL              |                |
+---------+-------------+------+-----+-------------------+----------------+
6 rows in set (0.00 sec)
```

Om din tabell inte ser lika ”rak” ut i kanterna gör det inget. Det är ditt teckensnitt som skapar den effekten och det påverkar inte databasen alls.

Allt är klart med databasen. Nu är det dags att börja skapa applikationen som skall hantera vår gästbok. Vi behöver inte lagra något i databasen ännu, det är ju det vår gästbok skall göra åt oss.

Börja med PHP

Vi kör igång direkt med lite grunder. En PHP-applikation är en vanlig textfil som kan skapas i vilken texteditor som helst. PHP-filer bör ha (men måste inte alltid) ha filändelsen **.php** , till exempel **hello.php**. Ett enkelt PHP-program kan se ut så här:

```
<?php
    echo ”Hello PHP”;
?>
```

<?php anger att här börjar PHP-kod och **?>** anger att här slutar PHP. Allt mellan de taggarna kommer tolkas som PHP-kod. Kommandot **echo** skriver ut strängen som kommer efter echo. Exemplet ovanför kommer skriva ut **Hello PHP** och inte göra speciellt mycket mer.

Spara filen som **hello.php** och kör den genom din webbläsare.

Skapa formuläret

Så var det dags att börja skapa gästboken. Vi skriver ett formulär i HTML där användaren kommer kunna skriva in sitt inlägg, sitt namn och e-postadress. *Vi bryr oss inte alls om utseendet på gästboken, det viktiga i vårt skapande är att få webbsidan kopplad till databasen med hjälp av PHP – inte att skapa en vacker gästbok.*

Skriv in följande i en texteditor och spara filen som **gastbok.php** i din webbkatalog:

```
<form method="post" action="gastbok.php">
Ditt namn: <input type="text" name="namn" /><br />
Din e-postadress: <input type="text" name="epost" /><br />
```

```
Din kommentar: <textarea name="inlagg" ></textarea><br />
<input type="submit" name="nyKommentar" value="Posta" />
</form>
```

Notera att detta inte är giltig HTML, men det bryr vi oss inte om just nu.

<form ...> anger vilken metod vi vill använda för att skicka information från formuläret till webbsidan. Vi kan använda POST eller GET. Vi väljer att använda POST. Inmatningsfälten skapar vi med **input**-taggen och textrutan för inlägget skapas med en **textarea**. Input och textarea namnger vi med attibutet **name**, så vi kan fånga upp rätt fält när vi skall lagra informationen i databasen.

Prova nu att komma åt formuläret med din webbläsare, för att se att det fungerar.

Fånga upp det användaren matar in i formuläret

När vi postar formuläret laddas sidan om och inget händer, det är för att vi inte fångar upp det som postats i formuläret. Vi behöver styra detta lite grann och vi lägger till följande ovanför formuläret i filen **gastbok.php**:

```
<?php
     echo "Ditt namn är: ".$_POST['namn']."<br />";
     echo "Din e-postadress är: ".$_POST['epost']."<br />";
     echo "Ditt inlägg är: ".$_POST['inlagg'];
?>
```

Om vi fyller i formulärets fält och postar det (genom att klicka på knappen Posta) kommer sidan skriva ut det vi skrivit i formuläret. Om vi laddar sidan utan att fylla i något kommer vi få felmeddelanden om att index är odefinerade (detta beror lite grann på hur PHP är konfigurerad på din dator). Felmeddelanden får vi för att vi försöker använda variabler ($_POST) som inte är satta.

Lägg märke till hur vi använder samma namn i $_POST som vi gör i formuläret.

I formuläret står det:

```
Ditt namn: <input type="text" name="namn" /><br />
```

Vi fångar upp inmatningen i fältet med PHP-koden:

```
echo "Ditt namn är: ".$_POST['namn']."<br />";
```

Det innebär att vi måste hålla koll på vad vi anger för namn i inmatningsfälten (input/textarea) i formuläret så vi kan fånga upp dem med $_POST. Om vi istället skulle använda method=”get” för forumläret skulle vi fånga upp inmatningarna med $_GET på samma sätt som vi gör för $_POST.

För att ordna detta med felmeddelanden skriver vi om PHP-koden så den ser ut så här:

```
<?php
     if(isset($_POST['nyKommentar'])) {
            echo "Ditt namn är: ".$_POST['namn']."<br />";
            echo "Din e-postadress är: ".$_POST['epost']."<br />";
            echo "Ditt inlägg är: ".$_POST['inlagg'];
     }
?>
```

Allt är frid och fröjd. Vi publicerar ett formulär för våra besökare och vi kan fånga upp vad de matar in och skriver ut det på sidan. Nu skall vi titta på hur vi ansluter vår applikation till den MySQL-databas vi skapade tidigare för att lagra inläggen i vår gästbok.

Koppla applikationen till MySQL

För att kunna ansluta en PHP-applikation till MySQL-servern behöver vi två funktioner: **mysql_connect()** och **mysql_select_db()**. mysql_connect() använder vi för att skapa en koppling till MySQL-servern och mysql_select_db() använder vi för att välja vilken databas vi vill arbeta mot på MySQL-servern.

mysql_connect() tar tre argument: vilken server, vilken användare och lösenordet till den användaren.

Om vi arbetar mot en lokal MySQL-server (den är installerad på samma dator som webbservern) är det enklast att ange IP-adressen 127.0.0.1 som server. Användarnamnet och lösenordet är det användarnamn och lösenord vi använder för att ansluta till MySQL-servern. Säg att vi har en standardinstallation av MySQL och inte har satt något lösenord själv, då är vårt användarnamn root med ett tomt lösenord. Vår anslutning från PHP skulle då se ut så här:

```
mysql_connect( "127.0.0.1", "root", "" );
```

Vad händer om vi inte kan ansluta till servern? Då kommer vi få ett felmeddelande från PHP som kanske inte är så snyggt, så vi skapar ett eget felmeddelande, med hjälp av felhantering så här:

```
mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
```

Om vi inte kan ansluta till databasservern kommer användaren få felmeddelandet *Kunde inte ansluta till databasservern!* på vår sida. Resten av applikationen kommer inte köras, då **die()** gör att PHP-applikationen avslutas. Om PHP är konfigurerad att visa felmeddelanden kommer vi trots **or die()** att få felmeddelanden från PHP. Dessa kan vi få bort genom att lägga till ett **@**-tecken innan mysql_connect:

```
@mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
```

*Notera att det **inte** är lämpligt att använda användarnamnet root med ett tomt lösenord om vi publicerar vår applikation på webben!*

Då var anslutningen till MySQL-servern avklarad, nu skall vi berätta för den vilken databas vi vill arbeta med. Vi skapade tidigare en databas som vi döpte till **mindatabas**, vi väljer den:

```
mysql_select_db( "mindatabas" );
```

Och precis som med mysql_connect() behöver vi ha felhantering, om PHP-applikationen inte kan välja vår databas, så vi lägger till **or die()**:

```
mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );
```

Nu är det dags att lägga in detta i gastbok.php så vi får med det i vår applikation. Vi lägger in det ovanför *if-satsen* vi skapade tidigare:

```
<?php
     @mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
     mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );

     if(isset($_POST['nyKommentar'])) {
            echo "Ditt namn är: ".$_POST['namn']."<br />";
            echo "Din e-postadress är: ".$_POST['epost']."<br />";
            echo "Ditt inlägg är: ".$_POST['inlagg'];
     }
?>
```

Notera att formuläret fortfarande skall vara kvar i filen, även om vi inte visar det här ovanför!

Se till att data är säker att lagra i databasen

Innan vi börjar lagra data som matas in från formuläret bör vi fundera på vad vi egentligen gör. När vi tillåter en okänd användare (en gästbok publicerad på Internet är en osäker applikation) öppnar vi upp för sårbarheter i vår applikation. En användare kan till exempel skicka in SQL-kommandon genom formulärfälten och på så vis till exempel köra en **DROP TABLE gastbok;** för att radera hela vår gästbok inklusive lagrad data. Det vill vi inte, eller hur?

Utan att gå in på detaljer hur det fungerar kan vi konstatera att vi behöver skydda oss från detta. I PHP finns funktionerna **mysql_real_escape_string()** och **addslashes()** som hjälper oss med detta. Funktionerna *escape:ar* farliga tecken, som kan användas för att skicka in SQL-kommandon till databasen. I praktiken gör den om ett '-tecken till \', och ” till \” med mera. \-tecknet (backslash) kallas *escapetecken* och gör att databasen skall tolka tecknet som ett vanligt tecken och inte som ett kommandotecken.

Nu skall vi se hur vi skall lösa detta.

```
<?php
     @mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
     mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );

     if(isset($_POST['nyKommentar'])) {
            $namn = mysql_real_escape_string( $_POST['namn'] );
            $epost = mysql_real_escape_string( $_POST['epost'] );
            $inlagg = mysql_real_escape_string( $_POST['inlagg'] );
            echo "Ditt namn är: ".$namn."<br />";
            echo "Din e-postadress är: ".$epost."<br />";
            echo "Ditt inlägg är: ".$inlagg;
     }
?>

<form method="post" action="gastbok.php">
Ditt namn: <input type="text" name="namn" /><br />
Din e-postadress: <input type="text" name="epost" /><br />
Din kommentar: <textarea name="inlagg" ></textarea><br />
<input type="submit" name="nyKommentar" value="Posta" />
</form>
```

Så där, vi lade till tre (3) rader innan **echo**-raderna. Vi ändrade också variabeln vi vill skriva ut från **$_POST** till de vi skapar med **mysql_real_escape_string()**-funktionen. Än så länge är det ingen större skillnad från innan, men vi kan testa vår applikation genom att fylla i ”- och '-tecken i formulärfälten. Vi kommer få tillbaka dem som \” och \'.

Lagra data från formuläret i databasen

Nu är det dags att lagra den data vi får via formuläret i databasen. Det är egentligen bara en funktion vi behöver: **mysql_query()** . Men vi väljer att dela upp lagringen i två delar: först skapar vi SQL-frågan i en variabel som vi döper till **$sql** och sedan anropar vi funktionen **mysql_query()**.

```
$sql = ”INSERT INTO gastbok(inlagg, namn, epost) VALUES('”.$inlagg.”','”.$namn.”','”.
$epost.”')”;
mysql_query($sql);
```

*Notera att raden med **$sql** = skall vara en rad, utan radbrytning!*

Vi avslutar inte våra SQL-frågor med ett semikolon-tecken (**;**) när vi använder oss av **mysql_query()** som vi gör när vi använder den vanliga MySQL-klienten (**mysql**). Funktionen tar hand om det åt oss.

```
<?php
     @mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
     mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );

     if(isset($_POST['nyKommentar'])) {
            $namn = mysql_real_escape_string( $_POST['namn'] );
            $epost = mysql_real_escape_string( $_POST['epost'] );
            $inlagg = mysql_real_escape_string( $_POST['inlagg'] );
            echo "Ditt namn är: ".$namn."<br />";
            echo "Din e-postadress är: ".$epost."<br />";
            echo "Ditt inlägg är: ".$inlagg;
            $sql = ”INSERT INTO gastbok(inlagg, namn, epost) VALUES('”.$inlagg.”','”.
$namn.”','”.$epost.”')”;
            mysql_query($sql);
     }
?>
```

För att spara plats på pappret har vi valt att inte visa formuläret i ovanstående listning.

Om vi nu testar vår applikation genom webbläsaren och postar data i formuläret kommer det att lagras i databasen. Detta kan vi verifiera genom att logga in i databasservern och kontrollera genom att skriva **SELECT * FROM gastbok;** . Glöm inte att välja databas först med **USE mindatabas;** . Datum och tid när inlägget postades uppdaterades automatiskt eftersom vi använder **CURRENT_TIMESTAMP** på fältet **datum**.

Vi hade som krav att applikationen skulle lagra IP-adressen besökaren har, så vi har en möjlighet att spåra

användaren om hon skriver något olagligt eller dumt. Hur löser vi detta? I PHP finns en speciell indexerad variabel som heter $_SERVER, den innehåller en del information – bland annat IP adressen till besökaren.

Vi använder oss av följande PHP-kod:

```
$ip = $_SERVER['REMOTE_ADDR'];
```

Nu kommer variablen **$ip** innehålla besökarens IP-adress. För att lagra den i databasen ändrar vi på raden med **$sql** = till:

```
$sql = ”INSERT INTO gastbok(inlagg, namn, epost, ip_addr) VALUES('”.$inlagg.”','”.
$namn.”','”.$epost.”','”.$ip.”')”;
```

Applikationskoden skall nu se ut så här:

```
<?php
     @mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
     mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );

     if(isset($_POST['nyKommentar'])) {
            $namn = mysql_real_escape_string( $_POST['namn'] );
            $epost = mysql_real_escape_string( $_POST['epost'] );
            $inlagg = mysql_real_escape_string( $_POST['inlagg'] );
            $ip = $_SERVER['REMOTE_ADDR'];
            echo "Ditt namn är: ".$namn."<br />";
            echo "Din e-postadress är: ".$epost."<br />";
            echo "Ditt inlägg är: ".$inlagg;
            $sql = ”INSERT INTO gastbok(inlagg, namn, epost, ip_addr) VALUES('”.
$inlagg.”','”.$namn.”','”.$epost.”','”.$ip.”')”;
            mysql_query($sql);
     }
?>
```

Notera att raden med $sql = ovanför inte skall radbrytas!

Nu provar vi att posta ett inlägg till i vår gästbok och ser vad som händer i databasen. IP-adressen bör ha lagrats.

Hämta data och visa gästboken

Nu när vi kan lagra inläggen (data) i vår databas är det dags att visa dem också. Under **<form> ... </form>** i applikationen skall vi skapa mer PHP-kod. Kod som hämtar inläggen och visar dem. Vi fokuserar på att få ut data från databasen och presentera den, inte på att göra en snygg gästbok. Det är en uppgift vi kan ta tag i senare.

För att hämta data från databasen med PHP använder vi **mysql_query()**, det vill säga – precis samma som för att lagra data. Vi tittar lite snabbt på hur det skulle kunna se ut:

```
$sql = ”SELECT * FROM gastbok”;
$q = mysql_query( $sql );
```

Här ser vi en skillnad mot när vi lagrade data med **mysql_query()** . Vi tilldelar variabeln **$q** det som kan komma tillbaka från vår fråga. Det vill säga de poster vi får tillbaka. Så, vad får vi tillbaka?

I variabeln **$q** kommer det finnas något som kallas *MySQL Resource*. Den innehåller de poster från databasen som hämtades med **SELECT** . Vi kan inte hämta ut data från den direkt utan måste använda oss av funktionen **mysql_fetch_array()** i PHP. **mysql_fetch_array()** hämtar nästa tillgängliga post från den fråga vi ställde med **SELECT**. Vi tilldelar en variabel svaret från **mysql_fetch_array()** och kan då hämta ut data genom att referera till variabeln och dess index. Ett exempel:

```
$sql = ”SELECT * FROM gastbok”;
$q = mysql_query( $sql );
$r = mysql_fetch_array( $q );
```

Nu kan vi hämta data från databasen med hjälp av variabeln **$r**. För att hämta data från fältet **id** skriver vi till exempel:

```
echo $r['id'];
```

Ofta kommer vi få fler än en post som svar efter en SELECT-fråga. Detta gör att vi måste börja fundera på att skapa en **loop**. En **while**-loop i PHP är mycket användbar då vi kan bestämma att så länge det finns något kvar att hämta skall vi hämta det och skriva ut det på webbsidan. Då skriver vi så här:

```
while( $r = mysql_fetch_array( $q ) ) {
     echo $r['id'];
}
```

Så länge det finns poster kvar att hämta, hämtar vi dem och skriver ut fältet id från posten.

Skall vi försöka skapa lite PHP som vi kan använda i vår gästbok nu? Vi hoppar ner till raden under **</form>** och skriver in följande:

```
<?php
     $sql = "SELECT * FROM gastbok";
     $q = mysql_query( $sql );

     while( $r = mysql_fetch_array( $q ) ) {
            echo "Namn: ".$r['namn']."<br />";
            echo "E-post: ".$r['epost']."<br />";
            echo "IP-adress: ".$r['ip_addr']."<br />";
            echo "Inlägg: ".$r['inlagg']."<br />";
            echo "<hr />";
     }
?>
```

Nu visas alla inläggen vi har i gästboken under formuläret på sidan. Har vi lagrat inläggen med **mysql_real_escape_string()** kommer vi se att det finns en del \-tecken med i texten. För att ta bort dem när vi presenterar inläggen använder vi oss av funktionen **stripslashes()**:

```
<?php
     $sql = "SELECT * FROM gastbok";
     $q = mysql_query( $sql );

     while( $r = mysql_fetch_array( $q ) ) {
            echo "Namn: ".stripslashes( $r['namn'] )."<br />";
            echo "E-post: ".stripslashes( $r['epost'] )."<br />";
            echo "IP-adress: ".stripslashes( $r['ip_addr'] )."<br />";
            echo "Inlägg: ".stripslashes( $r['inlagg'] )."<br />";
            echo "<hr />";
     }
?>
```

Så där, nu är det ju riktigt vackert.

Vi skall lägga till lite felhantering, om vi inte har några inlägg i gästboken kommer besökaren få ett felmeddelande och det är snyggare om vi själva skapar meddelandet. För att kunna kontrollera det här använder vi oss av funktionen **mysql_num_rows()** som berättar för oss hur många poster vi får tillbaka från **mysql_query()**. Om vi får tillbaka fler än noll (0) har vi något att visa, vi skriver om vår kod lite grann, så den anpassas för detta:

```
<?php
     $sql = "SELECT * FROM gastbok";
     $q = mysql_query( $sql );

     if( mysql_num_rows( $q ) > 0 ) {
            while( $r = mysql_fetch_array( $q ) ) {
                   echo "Namn: ".stripslashes( $r['namn'] )."<br />";
                   echo "E-post: ".stripslashes( $r['epost'] )."<br />";
                   echo "IP-adress: ".stripslashes( $r['ip_addr'] )."<br />";
                   echo "Inlägg: ".stripslashes( $r['inlagg'] )."<br />";
                   echo "<hr />";
            }
     } else {
            echo "Gästboken är tom.";
     }
?>
```

Vi lade till en **if-sats** där vi kontrollerar om vi får tillbaka fler än noll poster. Om vi får en eller flera kommer vi skriva ut dem, om vi inte får tillbaka något (**else**) kommer vi skriva ut texten *Gästboken är tom.*

Vart efter vi får in nya inlägg i vår gästbok kommer vi upptäcka att det vore smidigare att få de senaste inläggen överst, istället för underst som nu. Vi gör en enkel justering i **$sql =** raden:

```
$sql = “SELECT * FROM gastbok ORDER BY datum DESC”;
```

Nu är gästboken klar, efter de mål vi satte upp i början. Din uppgift nu är att bygga vidare på gästboken så den passar dina behov. Idéer på vad som kan göras:

- En mer grafiskt tilltalande sida.
- En funktion för att ta bort inlägg.
- Låta besökaren välja om hon vill sortera inläggen fallande eller stigande i datumordning.
- Visa inläggen på ett snyggare sätt, där namnet blir en länk som gör att man kan skicka e-post.

Applikationen i sin helhet

```
<?php
     @mysql_connect( "127.0.0.1", "root", "" ) or die( "Kunde inte ansluta till
databasservern!" );
     mysql_select_db( "mindatabas" ) or die( "Kunde inte välja databas!" );

     if(isset($_POST['nyKommentar'])) {
            $namn = mysql_real_escape_string( $_POST['namn'] );
            $epost = mysql_real_escape_string( $_POST['epost'] );
            $inlagg = mysql_real_escape_string( $_POST['inlagg'] );
            $ip = $_SERVER['REMOTE_ADDR'];
            $sql = ”INSERT INTO gastbok(inlagg, namn, epost, ip_addr) VALUES('”.
$inlagg.”','”.$namn.”','”.$epost.”','”.$ip.”')”;
            mysql_query($sql);
     }
?>

<form method="post" action="gastbok.php">
Ditt namn: <input type="text" name="namn" /><br />
Din e-postadress: <input type="text" name="epost" /><br />
Din kommentar: <textarea name="inlagg" ></textarea><br />
<input type="submit" name="nyKommentar" value="Posta" />
</form>

<?php
     $sql = “SELECT * FROM gastbok ORDER BY datum DESC”;
     $q = mysql_query( $sql );

     if( mysql_num_rows( $q ) > 0 ) {
            while( $r = mysql_fetch_array( $q ) ) {
                   echo “Namn: “.stripslashes( $r['namn'] ).”<br />”;
                   echo “E-post: “.stripslashes( $r['epost'] ).”<br />”;
                   echo “IP-adress: “.stripslashes( $r['ip_addr'] ).”<br />”;
                   echo “Inlägg: “.stripslashes( $r['inlagg'] ).”<br />”;
                   echo “<hr />”;
            }
     } else {
            echo “Gästboken är tom.”;
     }
?>
```