Table of Contents

[redacted]

[redacted]

what the SQL dump file looks like:

```
CREATE TABLE addressbook (id serial, name varchar(255),
address text,
tel varchar(50), email varchar(255));

INSERT INTO addressbook values (nextval('addressbook_id_seq'),
'Bugs
```

[redacted]

```
-------
3
(1 row)

test=#
```

which, in English, means "count all the records in the table addressbook and give me the total".

[redacted]

```
There are currently 3 records in the database.
```

As you can see, using PHP to get data from a PostgreSQL database involves several steps, each of which is actually a pre−defined PHP function. Let's dissect each step:

1. The first thing to do is specify some important information needed to establish a connection to the database

[redacted]

Digging Deeper

[redacted]

```
<li><font size="-1"><b><? echo $row[0]; ?></b></font>
<br>
<font size="-1"><? echo $row[1]; ?></font>
<p>
<?
}
}
// if no records present
// display message
else
{
?>
<font size="-1">No data available.</font>
<?
}

// close database connection
pg_close($connection);

?>
</body>
</html>
```

Here's what the output looks like:

As in the previous example, the script first sets up a connection to the database. The query is formulated and the result set is returned to the browser. In this case, since I'm dealing with multiple rows of data, I've used the pg_fetch_row() function in combination with a "for" loop to iterate through the result set and print the data within each row.

The pg_fetch_row() function returns the columns within each row as array elements, making it possible to easilPfns wissis6 rsn_row(D.lr0pray elements, A of data, I've usrowser. ex1l ter0,a, I've e e hTi bepossible to

[redacted]

```
$connection = pg_connect ("host=$host dbname=$db user=$user
password=$pass");

if (!$connection)
{
die("Could not open connection to database server");
}

// generate and execute a query
$query = "SELECT name, address FROM addressbook ORDER BY
name"; $result
= pg_query($connection, $query) or die("Error in query:
$query. " .
pg_last_error($connection));

// get the number of rows in the resultset
// this is PG-specific
$rows = pg_num_rows($result);

// if records present
if ($rows > 0)
{
// iterate through resultset
for ($i=0; $i<$rows; $i++)
{
$row = pg_fetch_object($result, $i);
?>
<li><font size="-1"><b><? echo $row->name; ?></b></font>
<br>
<font size="-1"><? echo $row->address; ?></font>
<p>
<?
}
}
// if no records present
```

[redacted]

In this case, each row is returned as a PHP object, whose properties correspond to field names; these fields can be accessed using standard object notation.

```
" . pg_last_error($connection));

// now roll them back
$query = "ROLLBACK";
// if you want to commit them, comment out the line above
// and uncomment the one below
// $query = "COMMIT";
$result = pg_query($connection, $query) or die("Error in
query: $query.
" . pg_last_error($connection));

// now check to see how many records are there in the table
// and print this
$query = "SELECT * FROM addressbook";
$result = pg_query($connection, $query) or die("Error in
query: $query.
" . pg_last_error($connection)); $rows = pg_num_rows($result);
echo
"There are currently $rows records in the database";

// close database connection
pg_close($connection);

?>
```

Technically, there's nothing new here − this script uses the same functions you've seen in preceding examples. The difference lies in the use of multiple SQL statements to begin and end a transaction block, and in the use of COMMIT and ROLLBACK statements to commit and erase records from the database.

A Well−Formed Idea

Finally, how about one more example to wrap things up? This next script contains a form which can be used to enter new addresses into the table, together with a form processor that actually creates and executes the INSERT statement.

```
<html>
<head><basefont face="Arial"></head>
<body>
<h2>Address Book</h2>

<?
// form not yet submitted
// display form
if (!$submit)
{
?>
```

[redacted]

[redacted]