MODULE V
PHP & POSTGRESQL:

PostgreSQL is a powerful, enterprise-class open source object-relational database management system. PostgreSQL supports advanced data types and advance performance optimization, features only available in the expensive commercial database, like Oracle and SQL Server.

It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. This tutorial will give us quick start with PostgreSQL and make us comfortable with PostgreSQL programming.

PostgreSQL is known for its reliability, data integrity and correctness. It is compatible with all major operating systems like Linux, UNIX and Windows. It is based on POSTGRES 4.2 and is developed at Berkeley Computer Science Department, University of California. It is an advanced RDBMS which is capable of more than retrieving and updating data. PostgreSQL is fully ACID compliant. It supports foreign keys, joins, triggers, views and stored procedures. It includes SQL: 2008 data types like NUMERIC, CHAR, INTERVAL and INTEGAR, etc. The database helps store binary large objects including audio, video and images. It features a native programming interface for C/C++, .Net, Java, Perl, Ruby, Python, etc. Its other sophisticated features include Multi-Version Concurrency Control (MVCC), table spaces, point in time recovery, nested transactions, asynchronous replication, a sophisticated query planner, etc. It also supports multi-byte character coding, Unicode and international character sets. It is highly scalable in both quantity of data to be managed and the number of concurrent users to be accommodated.

PostgreSQL needs minimum efforts as it is quite stable. So, if we develop PostgreSQL-based applications then the total cost of ownership will be low as compared to other database management systems. It is a free database. Its source code is under the license of PostgreSQL. We can modify and distribute PostgreSQL in any form.

Advantages of PostgreSQL

- **Open Source DBMS**: Among current Open Source DBMS only the PostgreSQL provides unlimited development possibilities. It also enables users to join communities to post or share bugs and difficulties. o Freedom of use and modify: *We can run PostgreSQL for any use. It can be connected to multiple servers, cores and users. We are also free to modify it to fulfill our needs. o *Unlimited copying and distribution: PostgreSQL allows unlimited copying and distribution.
- **ACID and Transaction:** PostgreSQL support Atomicity, Consistency, Isolation and Durability (ACID). o Multiple indexing techniques: Apart from

an double os

the B+ tree index techniques, it also provides various other techniques like GIN (Generalized Inverted Index) and GST (Generalized Search Tree), etc.

- **Full-text search**: It offers full-text search when searching for strings.
- **Diverse replication methods**: It supports a variety of replication methods like cascading, Slony-I and Streaming replication. o Diverse extension functions: It is compatible with different techniques which are used to store geographic data such as Key-Value Store, PostGIS and DBLink. o Cost-effective: It is designed to have lower maintenance. o Cross-platform: It is compatible with almost all brands of Unix and with Windows via the Cygwin framework.
- **Suitable for high volume environments**: It is based on the multiple row data storage strategy (MVCC) which makes PostgreSQL highly responsive in high volume environments. o Complete Internet solution: It comes with everything that we need to Web-enable our company. Our website will be leveraged with PHP 4 Scripting Language, Apache Web Server and the Zone Application server with online access to the PostgreSQL datastore. o Easy migration: It comes with various tools that help migrating data from other DBMS.

A Brief History of PostgreSQL:

PostgreSQL started its journey as Postgres. It was created by Professor Michael Stonebraker at UCB. He started Postgres as a follow-up project to Ingres in 1986. Ingres was developed between 1977 and 1985 according to classic RDBMS theory. Later in I994, it was acquired by Computer Associates. Postgres was developed between 1986 and 1994. Its development included the development of INGRES concepts focused on query language Quel and object orientation. Its development was not based on the code base of INGRES. It was commercialized as illustra and bought by Informix. Later in 2001, Informix was bought by IBM. Postgres95 was developed between 1994 and 1995.

Two Ph.D. students, Jolly Chen and Andrew Yu at Stonebanker’s lab replaced the POSTQUEL query language of Postgres with a subset of SQL and renamed it Postgres95. In 1996, a group of developers outside of Berkeley science department realize the potential of the system and devoted themselves to the development of Postgres95. Over the next eight years, this group transformed the Postgres. The group created detailed regression tests for quality assurance, fixed bugs, set up a mailing list for bug reports, and brought consistency to the code base. They also filled the various gaps like documentation for users and developers. After it is transformed into new database it started a new life in open source world with various new features and it took its current name PostgreSQL.

PostgreSQL started with version 6.0, over the next four years it moved from version 6.0 to version 7 which was loaded with major improvements and new features such as: Unique SQL features: Many new features were added like subsets, constraints, defaults, foreign keys, primary keys, quoted identifiers, type casting, binary and hexadecimal integer input. Multiversion Concurrency Control (MVCC): The Multiversion concurrency has replaced the table-level

an double os

locking. It allows online backups when a database is running and enables readers to read consistent data. Better built-in types: Improved native types were added including the various date/time types and extra geometric types. Improved Speed: Speed and performance were increased by 20 to 40 percent and the backend start-up time was reduced by 80 percent. During next four years after the release of versions 7.0 and 7.4 again a number of features were added to PostgreSQL.

These features were Write-Ahead Log (WAL), prepared queries, SQL schemas, outer joins, SQL92 join syntax, complex queries, TOAST, IPv6, full-text indexing, auto-vacuum, improved SSL support, database statistics information, table functions, an optimizer overhaul, Perl/TCL procedural languages, Python, etc. Today, PostgreSQL has a large user base and it continues to improve. Version 8.0 of PostgreSQL is supposed to have features like table spaces, point in time recovery, nestled transactions and java stored procedures. A number of organizations including government entities and companies use PostgreSQL. We can easily find it in ADO, NTT Data, CISCO, NOAA, The US Forestry Service, Research in Motion and in The American Chemical Society.

FEATURES OF POSTGRESQL

Compatible with various platforms using all major languages and middleware. It offers a most sophisticated locking mechanism Support for multi-version concurrency control Mature Server-Side Programming Functionality Compliant with the ANSI SQL standard Full support for client-server network architecture Log-based and trigger-based replication SSL Standby server and high availability Object-oriented and ANSI-SQL2008 compatible Support for JSON allows linking with other data stores like NoSQL which act as a federated hub for polyglot databases.

MYSQL	POSTGRESQL
The MySQL project has made its source code available under the terms of the GNU License, and other proprietary agreements.	PostgreSQL is released under PostgreSQL License.
It's now owned by Oracle Corporation and offers several paid editions	It's free and open-source software. That means we will never need to pay anything for this service
MySQL is ACID compliant only when using with NDB and InnoDB Cluster Storage engines	PostgreSQL is completely ACID compliant
MySQL performs well in OLAP and OLTP systems where only read speed is important.	PostgreSQL performance works best in systems which demand the execution of complex queries
MySQL is reliable and works well with BI (Business Intelligence) applications, which are difficult to read	PostgreSQL works well with BI applications. However, it is more suited for Data Warehousing and data analysis applications which need fast read-write speeds

an double os

Advantage of POSTGRESQL

- PostgreSQL can run dynamic websites and web apps as a LAMP stack option
- PostgreSQL's write-ahead logging makes it a highly fault-tolerant database
- PostgreSQL source code is freely available under an open source license. This allows us the freedom to use, modify, and implement it as per our business needs.
- PostgreSQL supports geographic objects so we can use it for location-based services and geographic information systems
- PostgreSQL supports geographic objects so it can be used as a geospatial data store for location-based services and geographic information systems
- To learn Postgres, we don't need much training as its easy to use
- Low maintenance administration for both embedded and enterprise use

Disadvantage of POSTGRESQL

- Postgres is not owned by one organization. So, it has had trouble getting its name out there despite being fully featured and comparable to other DBMS systems
- Changes made for speed improvement requires more work than MySQL as PostgreSQL focuses on compatibility
- Many open source apps support MySQL, but may not support PostgreSQL
- On performance metrics, it is slower than MySQL.

Key Features of PostgreSQL

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC). PostgreSQL supports a large part of the SQL standard and offers many modern features including the following −

- Complex SQL queries
- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multi-version concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

DATA TYPES

PostgreSQL supports a wide set of Data Types. Besides, users can create their own custom data type using CREATE TYPE SQL command. There are different categories of data types in PostgreSQL. They are discussed below.

an double OS

Numeric Types

Numeric types consist of two-byte, four-byte, and eight-byte integers, four-byte and eight-byte floating-point numbers, and selectable-precision decimals. The following table lists the available types.

Name	Storage Size	Description	Range
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	9223372036854775808 to 9223372036854775807
decimal	variable	user-specified precision,exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision,exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision,inexact	6 decimal digits precision
double precision	8 bytes	variable-precision,inexact	15 decimal digits precision
smallserial	2 bytes	small auto-incrementing integer	1 to 32767
serial	4 bytes	Auto-incrementing integer	1 to 2147483647
bigserial	8 bytes	large auto-incrementing integer	1 to 9223372036854775807

Monetary Types

The money type stores a currency amount with a fixed fractional precision. Values of the numeric, int, and bigint data types can be cast to money. Using Floating point numbers is not recommended to handle money due to the potential for rounding errors.

Name	Storage Size	Description	Range
money	8 bytes	currency amount	-92233720368547758.08 to +92233720368547758.07

an doubleOS

Character Types

The table given below lists the general-purpose character types available in PostgreSQL.

Sl. No.	Name & Description
1	character varying(n), varchar(n) variable-length with limit
2	character(n), char(n) fixed-length, blank padded
3	text variable unlimited length

Binary Data Types

The bytea data type allows storage of binary strings as in the table given below.

Name	Storage Size	Description
bytea	1 or 4 bytes plus the actual binary string	variable-length binary string

Date/Time Types

PostgreSQL supports a full set of SQL date and time types, as shown in table below. Dates are counted according to the Gregorian calendar. Here, all the types have resolution of 1 microsecond / 14 digits except date type, whose resolution is day.

Name	Storage Size	Description	Low Value	High Value
timestamp [(p)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD
TIMESTAMPTZ	8 bytes	both date and time, with time zone	4713 BC	294276 AD
date	4 bytes	date (no time of day)	4713 BC	5874897 AD
time [(p)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00
time [(p)] with time zone	12 bytes	times of day only, with time zone	00:00:00+1459	24:00:00-1459
interval [fields] [(p)]	12 bytes	time interval	-178000000 years	178000000 years

Boolean Type

PostgreSQL provides the standard SQL type Boolean. The Boolean data type can have the states true, false, and a third state, unknown, which is represented by the SQL null value.

Name	Storage Size	Description
boolean	1 byte	state of true or false

an double os

Enumerated Type

Enumerated (enum) types are data types that comprise a static, ordered set of values. They are equivalent to the enum types supported in a number of programming languages.

Unlike other types, Enumerated Types need to be created using CREATE TYPE command. This type is used to store a static, ordered set of values. For example compass directions, i.e., NORTH, SOUTH, EAST, and WEST or days of the week as shown below –

CREATE TYPE week AS ENUM ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun');

Enumerated, once created, can be used like any other types.

Geometric Type

Geometric data types represent two-dimensional spatial objects. The most fundamental type, the point, forms the basis for all of the other types. (point, line, lseg, box, path, polygon, circle)

Network Address Type

PostgreSQL offers data types to store IPv4, IPv6, and MAC addresses. It is better to use these types instead of plain text types to store network addresses, because these types offer input error checking and specialized operators and functions. (cidr, inet, macaddr)

Bit String Type

Bit String Types are used to store bit masks. They are either 0 or 1. There are two SQL bit types: bit(n) and bit varying(n), where n is a positive integer.

Text Search Type

This type supports full text search, which is the activity of searching through a collection of natural-language documents to locate those that best match a query. There are two Data Types for this – (tsvector, tsquery)

Array Type

PostgreSQL gives the opportunity to define a column of a table as a variable length multidimensional array. Arrays of any built-in or user-defined base type, enum type, or composite type can be created.

Declaration of Arrays

Array type can be declared as

```
CREATE TABLE monthly_savings (
   name text,
   saving_per_quarter integer[],
   scheme text[][]
);
```

an double os

or by using the keyword "ARRAY" as

```
CREATE TABLE monthly_savings (
  name text,
  saving_per_quarter integer ARRAY[4],
  scheme text[][]
);
```

Inserting values

Array values can be inserted as a literal constant, enclosing the element values within curly braces and separating them by commas. An example is shown below:

```
INSERT INTO monthly_savings
VALUES (‘Manisha’,
‘{20000, 14600, 23500, 13250}’,
‘{{“FD”, “MF”}, {“FD”, “Property”}}’);
```

Accessing Arrays

An example for accessing Arrays is shown below. The command given below will select the persons whose savings are more in second quarter than fourth quarter.

```
SELECT name FROM monhly_savings WHERE saving_per_quarter[2] >
saving_per_quarter[4];
```

POSTGRESQL COMMANDS

CREATE DATABASE,

The CREATE DATABASE statement is used to create new PostgreSQL database.

Syntax:- *CREATE DATABASE databasename*;

Example:- CREATE DATADASE college;

CREATE TABLE

CREATE TABLE is a keyword, telling the database system to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement. Initially, the empty table in the current database is owned by the user issuing the command. Then, in brackets, comes the list, defining each column in the table and what sort of data type it is.

Syntax

```
CREATE TABLE table_name (
  Column1_name TYPE column_constraint,
  Column2_name TYPE column_constraint,
  Column3_name TYPE column_constraint,
  .........
  ColumnN_name TYPE column_constraint,)
);
```

am double os

Example

```
CREATE TABLE books (
    bookid       char(5) PRIMARY KEY,
    title      varchar(40) NOT NULL,
    publisher        varchar(40) NOT NULL,
    price              integer NOT NULL,
);
```

```
CREATE TABLE Persons  (
      PersonID int,
      LastName varchar(255),
      FirstName varchar(255),
      Address varchar(255),
      City varchar(255)
);
```

Note:-

To show the schema or description of table

\d table name ;

PostgreSQL column constraints

- **NOT NULL** – the value of the column cannot be NULL.
- **UNIQUE** – the value of the column must be unique across the whole table. However, the column can have many NULL values because PostgreSQL treats each NULL value to be unique. Notice that SQL standard only allows one NULL value in the column that has the UNIQUE constraint.
- **PRIMARY KEY** – this constraint is the combination of NOT NULL and UNIQUE constraints. We can define one column as PRIMARY KEY by using column-level constraint. In case the primary key contains multiple columns, we must use the table-level constraint.
- **CHECK** – enables to check a condition when we insert or update data. For example, the values in the price column of the product table must be positive values.
- **REFERENCES** – constrains the value of the column that exists in a column in another table. We use REFERENCES to define the foreign key constraint.

CREATE TABLE AS

The PostgreSQL CREATE TABLE AS statement is used to create a table from an existing table by copying the existing table's columns. It is important to note that when creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement).

The PostgreSQL CREATE TABLE AS statement is used to create a table from an existing table by copying the existing table's columns. It is important to note that when creating a table in this way, the new table will be

an double os

populated with the records from the existing table (based on the SELECT Statement).

Syntax

CREATE TABLE new_table_name AS query;

Example

```
CREATE TABLE inventory AS
  SELECT *
  FROM products
  WHERE quantity > 5;
```

SELECT

PostgreSQL SELECT statement is used to fetch the data from a database table, which returns data in the form of result table. These result tables are called result-sets.

Syntax:

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2...are the fields of a table whose values we want to fetch. If we want to fetch all the fields available in the field, then we can use the following syntax:

*SELECT * FROM table_name;*

Example

Example	Description
SELECT Empid, fname, salary FROM customer;	It will display Empid, fname, salary columns only
*SELECT * FROM customer;*	It will display all columns

Because of its complexity, we will break down the PostgreSQL SELECT statement tutorial into many shorter and easy-to-understand tutorials so that we can learn the functionality of each clause faster.

The SELECT statement has the following **clauses**:

- Select distinct rows using DISTINCT operator.
- Sort rows using ORDER BY clause.
- Filter rows using WHERE clause.
- Select a subset of rows from a table using LIMIT or FETCH clause.
- Group rows into groups using GROUP BY clause
- Filter groups using HAVING clause.
- Join with other tables using joins such as INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN clauses.
- Perform set operations using UNION, INTERSECT, and EXCEPT.

am double os

SELECT INTO

The PostgreSQL SELECT INTO statement allows us to create a new table and inserts data returned by a query. The new table columns have name and data types associated with the output columns of the SELECT clause. Unlike the SELECT statement, the SELECT INTO statement does not return data to the client.

Syntax

*SELECT **
INTO newtable [IN externaldb]
FROM table1;

SELECT column_name(s)
INTO newtable [IN externaldb]
FROM table1;

Example	Description
*SELECT * INTO CustomersBackup2019 FROM Customers;*	Create a backup copy of Customers:
*SELECT * INTO CustomersBackup2019 IN Backup.mdb' FROM Customers;*	Use the IN clause to copy the table into another database:
SELECT CustomerName, ContactName INTO CustomersBackup2019 FROM Customers;	Copy only a few columns into the new table:
*SELECT * INTO CustomersBackup2019 FROM Customers WHERE Country='Germany';*	Copy only the German customers into the new table:
SELECT Customers.CustomerName, Orders.OrderID INTO CustomersOrderBackup2019 FROM Customers LEFT JOIN Orders ON Customers.CustomerID=Orders.CustomerID;	Copy data from more than one table into the new table:

DELETE

The PostgreSQL DELETE statement is used to delete a single record or multiple records from a table in PostgreSQL.

Syntax:-

DELETE FROM table_name [WHERE conditions];

Example

DELETE FROM book WHERE price < 100 ;
DELETE FROM contacts WHERE first_name = 'Vimala';

The PostgreSQL UPDATE Query is used to modify the existing records in a table. We can use WHERE clause with UPDATE query to update the selected rows. Otherwise, all the rows would be updated.

Syntax

UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];

Example

UPDATE COMPANY SET SALARY = 15000 WHERE ID = 3;

UPDATE contacts SET first_name = 'Jane' WHERE contact_id = 35;

UPDATE contacts SET city='Miami', state='Florida' WHERE contact_id >= 200;

INSERT INTO

The PostgreSQL INSERT statement is used to insert a single record or multiple records into a table in PostgreSQL. One can insert a single row at a time or several rows as a result of a query.

Syntax

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);

If we can use all attribute in same order

INSERT INTO TABLE_NAME VALUES (value1, value2, value3,...valueN);

If we can insert multiple rows at a same time

INSERT INTO table (column1, column2, …)
VALUES
(value1, value2, …),
(value1, value2, …) ,...;

Example

INSERT INTO users (age, email, first_name, last_name) VALUES (30, 'pkandoubleos@gmail.com', 'Arjun', 'Kumar');

INSERT INTO users VALUES (2, 22, 'John', 'Smith', 'john@smith.com');

INSERT INTO "EMPLOYEES"(
"ID", "NAME", "AGE", "ADDRESS", "SALARY")
VALUES (1, 'Ajeet', 25, 'Mau ', 65000.00), (2, 'Rakul', 21, 'Shimla', 85000.00), (3, 'Manisha', 24, 'Mumbai', 65000.00), (4, 'Larry', 21, 'Paris', 85000.00);

an double os

PHP - POSTGRESQL INTEGRATION:

1. pg_connect(),
2. pg_connection_status()
3. pg_dbname()
4. pg_last_error()
5. pg_close()
6. pg_query()
7. pg_execute()
8. pg_fetch_row()
9. pg_fetch_array()
10. pg_fetch_all()
11. pg_fetch_assoc()
12. pg_fetch_object()
13. pg_num_rows()
14. pg_num_fields()
15. pg_affected_rows()
16. pg_free_result()

1. pg_connect()

It is a function to use to connect to a PostgreSQL database, returning a database handle. PostgreSQL requires connection parameters to be submitted as a single string, denoted by connection_string. Several parameters are recognized in this string, including:

- **connect_timeout**: The number of seconds to continue waiting for a connection response. Specifying zero or no value will cause the function to wait indefinitely.
- **dbname**: The name of the database we'd like to connect to.
- **host**: The server location as defined by a hostname, such as www.example.com, ecommerce, or localhost.
- **hostaddr**: The server location as defined by an IP address, such as 192.168.1.104.
- **password**: The connecting user's password.
- **port**: The port on which the server operates. By default, this is 5432; therefore, we need to specify this parameter only if the destination server is operating on another port.
- **user**: The connecting user.

Syntax

pg_connect(connection_string)

Example

$pg = pg_connect("host=localhost user=postgres password=gems dbname=college");

am double os

2. pg_close()

The database connections opened during the execution of a script are automatically closed once the script completes. The connection will be closed as soon as the script ends. To close the connection before end the script we can all the function pg_close().

Syntax

pg_close(connection)

Example

```
<?php
$pg = pg_connect("host=localhost user=postgres password=gems
dbname=college") or die("Can't connect to database.");
echo "This is where database operations are performed.";
pg_close();
?>
```

If multiple connections to, say, different databases are open, we can close each as its services are no longer needed. For instance:

```
<?php
$pg = pg_connect("host=localhost user=postgres password=gems
dbname=college");
$pg2 = pg_connect("host=example.com user=postgres password=gems
dbname=school");
echo "Perform some database operations. <br />";
// We're finished with $pg2, so close the connection
pg_close($pg2);
echo "Perform additional database operations.";
// Close the $pg connection
pg_close($pg);
?>
```

Example	Output

*<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db
user=postgres password=gems");
if (!$con) {
die('Could not connected to college:
'.pg_last_error());
}
else
echo 'Successfully Connected to '. $db;
pg_close($con);
?>*
	Successfully Connected to college

an double OS

3. pg_query()

The pg_query() is used to execute query on the default database. Before performing any operation on a PostgreSQL database, it is required to set a connection to the PostgreSQL database we want to work with it. And this is done by pg_connect() function.

Syntax

pg_query(query, connection_string);

Where query specifies the query string and is required. The connection specified the PostgreSQL connection to use and is required. If not specified the last connection opened by pg_connect() is used.

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
      $qry="select * from student";
      $result=pg_query($con, $qry);
      while ($row = pg_fetch_row($result)) {
            echo "<br>\n";
            echo "regno: $row[0] name: $row[1] age: $row[2] mark: $row[4]";
      }
}
else
      die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

regno: 102 name: raju age: 17 mark: 54
regno: 105 name: vani age: 19 mark: 67
regno: 103 name: arjun age: 20 mark: 81
regno: 108 name: kumar age: 17 mark: 55
regno: 107 name: sukanya age: 19 mark: 95

4. pg_execute()

It sends a request to execute a prepared statement with given parameters, and waits for the result. The command to be executed is specified by naming a previously-prepared statement, instead of giving a query string. This feature allows commands that will be used repeatedly to be parsed and planned

an double os

just once, rather than each time they are executed. The statement must have been prepared previously in the current session. pg_execute() is supported only against PostgreSQL 7.4 or higher connections; it will fail when using earlier versions.

Syntax

pg_execute ($connection, string, array)

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
       $result=pg_prepare($con, "myqry","select * from student where
name=$1");
       $result=pg_execute($con, "myqry",array("vinusha"));
       while ($row = pg_fetch_row($result)) {
              echo "<br>\n";
              echo "regno: $row[0] name: $row[1] age: $row[2] mark: $row[4]";
       }
       echo "<br><br>";
       $result=pg_execute($con, "myqry",array("vimala"));
       while ($row = pg_fetch_row($result)) {
              echo "<br>\n";
              echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
       }
}
else
       die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

5. pg_last_error()

It returns the last error message for a given connection. The error messages may be overwritten by internal PostgreSQL (libpq) function calls. It may not return an appropriate error message if multiple errors occur inside a PostgreSQL module function.

Syntax

pg_last_error(connection)

Example

```
<?php
$db=college;
```

an double os

```
$con = pg_connect("host=localhost dbname=$db user=postgares
password=gems");
if (!$con) {
    die('Could not connected to college: ');
        echo pg_last_error($con);
}
else
        echo 'Successfully Connected to '. $db;
pg_close($con);
?>
```

6. pg_connection_status()

It returns the status of the specified connection.

Syntax

```
pg_connection_status (connection );
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
$start=pg_connection_status($con);
if ($start == pgsql_connection_ok) {
    echo 'Connection status is OK';
} else {
    echo 'Connection status NOT OK';
}
pg_close($con);
?>
```

Output

Connection status is OK

7. pg_dbname()

It returns the name of the database of given PostgreSQL connection resource.

Syntax

```
pg_dbname (connection)
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
echo "The current database name is <b>".pg_dbname($con);
pg_close($con);
```

an double os

```
?>
```

Output
The current database name is college

8. pg_fetch_row()
It fetches one row of data from the result associated with the specified result resource.

Syntax

```
pg_fetch_row ($result)
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
       $qry="select name, mark from student";
       $result=pg_query($con, $qry);
       while ($row = pg_fetch_row($result)) {
              echo "<br>\n";
              echo "name: $row[0] mark: $row[1]";
       }
}
else
       die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output
Successfully Connected to college

name: raju   mark: 54
name: vani   mark: 67
name: arjun   mark: 81
name: kumar   mark: 55
name: sukanya mark: 95

9. pg_fetch_array()
It is an extended version of pg_fetch_row(). In addition to storing the data in the numeric indices (field number) to the result array, it can also store the data using associative indices (field name). It stores both indicies by default.

an double os

Syntax

```
pg_fetch_array (result)
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select name, mark from student";
        $result=pg_query($con, $qry);
        while($arr=pg_fetch_array($result,NULL,PGSQL_NUM)){
        echo $arr[0];
        echo $arr[1];
        echo "<br>";
        }
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

```
raju      54
vani      67
arjun     81
kumar     55
sukanya   95
```

10. pg_fetch_all()

It r returns an array that contains all rows (records) in the result resource.

Syntax

```
pg_fetch_all ($result);
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select * from student";
```

an double os

```
        $result=pg_query($con, $qry);
        if (!$result) {
    echo "An error occurred.\n";
    exit;
        }
        $arr = pg_fetch_all($result);
        print_r($arr);
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

Array ([0] => Array ([regno] => 102 [name] => raju [age] => 17 [gender] => m [mark] => 54) [1] => Array ([regno] => 105 [name] => vani [age] => 19 [gender] => f [mark] => 67) [2] => Array ([regno] => 103 [name] => arjun [age] => 20 [gender] => m [mark] => 81) [3] => Array ([regno] => 108 [name] => kumar [age] => 17 [gender] => m [mark] => 55) [4] => Array ([regno] => 107 [name] => sukanya [age] => 19 [gender] => f [mark] => 95))

11. pg_fetch_assoc()

It returns an associative array that corresponds to the fetched row (records). It is equivalent to calling pg_fetch_array() with PGSQL_ASSOC as the optional third parameter. It only returns an associative array. If we need the numeric indices, use pg_fetch_row().

Syntax

pg_fetch_assoc (result);

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
    echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select regno, name, mark from student";
        $result=pg_query($con, $qry);
        if (!$result) {
    echo "An error occurred.\n";
    exit;
        }
        while ($row = pg_fetch_assoc($result)) {
```

an double os

```
        echo $row['regno']."  ";
        echo $row['name']."  ";
        echo $row['mark']."  ";
        echo"<br>";
        }
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

102 raju 54
105 vani 67
103 arjun 81
108 kumar 55
107 sukanya 95

12. pg_fetch_object()

It returns an object with properties that correspond to the fetched row's field names. It can optionally instantiate an object of a specific class, and pass parameters to that class's constructor.

Syntax

```
g_fetch_object (result);
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
    echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select * from student order by name";
        $result=pg_query($con, $qry);
        if (!$result) {
    echo "An error occurred.\n";
    exit;
        }
        while ($data = pg_fetch_object($result)) {
        echo $data->regno . " (";
        echo $data->name . "): ";
        echo $data->mark . "<br />";
        }
```

an double os

```
        pg_free_result($result);
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

103 (arjun): 81
108 (kumar): 55
102 (raju): 54
107 (sukanya): 95
105 (vani): 67

13. pg_num_rows()

It will return the number of rows in a PostgreSQL result resource.

Synatx

pg_num_rows (result);

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
    echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select * from student";
        $result=pg_query($con, $qry);
        if (!$result) {
    echo "An error occurred.\n";
    exit;
        }
        $rows = pg_num_rows($result);
        echo $rows . " row(s) returned.\n";
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

an double os

Output

Successfully Connected to college

6 row(s) returned.

14. pg_num_fields()

It returns the number of fields (columns) in a PostgreSQL result resource.

Synatx

pg_num_fields (result);

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
    echo 'Successfully Connected to '. $db."<br><br>";
        $qry="select * from student";
        $result=pg_query($con, $qry);
        if (!$result) {
    echo "An error occurred.\n";
    exit;
        }
        $num = pg_num_fields($result);
        echo $num . " field(s) returned.\n";
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

5 field(s) returned.

15. pg_affected_rows()

It returns the number of tuples (instances/records/rows) affected by INSERT, UPDATE, and DELETE queries. Since PostgreSQL 9.0 and above, the server returns the number of selected rows. Older PostgreSQL return 0 for SELECT.

Syntax

pg_affected_rows (result);

an double os

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
       $qry="select * from student where mark > 60";
       $result=pg_query($con, $qry);
       if (!$result) {
   echo "An error occurred.\n";
   exit;
       }
       $cmdtuples = pg_affected_rows($result);
       echo $cmdtuples . " tuples are affected.\n";
}
else
       die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

4 tuples are affected.

16. pg_free_result()

It frees the memory and data associated with the specified PostgreSQL query result resource. This function need only be called if memory consumption during script execution is a problem. Otherwise, all result memory will be automatically freed when the script ends.

Synatx

```
pg_free_result (result);
```

Example

```
<?php
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo 'Successfully Connected to '. $db."<br><br>";
       $qry="select * from student where gender = 'f'";
       $result=pg_query($con, $qry);
       if (!$result) {
   echo "An error occurred.\n";
   exit;
```

an double os

```
        }
        $num = pg_num_rows($result);
        echo $num . " row(s) returned.\n";
        pg_free_result($result);
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
?>
```

Output

Successfully Connected to college

3 row(s) returned.

INSERTION AND DELETION OF DATA USING PHP

Example of Insertion of data using PHP

```
<html><head>
<title>Retriving of data from PHP</title>
</head><body>
<form action="" method="post">
        <h2>Enter the Details of a STUDENT</h2><br></br>
                Enter the Reg No
        <input type="text" name="reg"><br><br>
                Enter the Name
        <input type="text" name="fname"><br><br>
                Enter the Age
        <input type="text" name="years"><br><br>
                Enter the Gender
        <input type="text" name="gen"><br><br>
                Enter the Mark
        <input type="text" name="score"><br>
                 
        <br><input type="submit" />
</form></body></html>
<?php
if ($_POST){
$reg1=$_POST['reg'];
$fname1=$_POST['fname'];
$years1=$_POST['years'];
$gen1=$_POST['gen'];
$score1=$_POST['score'];
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
```

an double os

```
if ($con) {
    echo '<br><br><br>Successfully Connected to '. $db."<br><br>";
        $qry="select * from student";
        $result=pg_query($con, $qry);
        while ($row = pg_fetch_row($result)) {
                echo "<br>\n";
                echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
        }
        echo"<br>rrrrrrrr<br>";
        $qry1="insert into student
(regno,name,age,gender,mark)values($reg1,'$fname1',$years1,'$gen1',$score
1)";
        $result1=pg_query($con, $qry1);
        $result=pg_query($con, $qry);
        while ($row = pg_fetch_row($result)) {
                echo "<br>\n";
                echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
        }
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
}
?>
```

an double os

Output

Example of Deletion of data using PHP

```
<html>
<head>
<title>Deleting of data from PHP</title>
</head>
<body>
<form action="" method="post">
        <h2>Enter the Datais of a STUDENT</h2><br></br>
        <h3>Enter the Reg No to DELETE
        <input type="text" name="reg">
                 
        <br><input type="submit" />
</form>
</body>
</html>


<?php
```

```
if ($_POST){
$no=$_POST['reg'];
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
   echo '<br><br><br>Successfully Connected to '. $db."<br><br>";
       $qry1="select * from student";
       $qry2="delete from student where regno = $no";
       $result1=pg_query($con, $qry1);
       while ($row = pg_fetch_row($result1)) {
               echo "<br>\n";
               echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
       }
       echo "<font color=red><br><br>Regno $no is now deleted<br></font>";
       $result2=pg_query($con, $qry2);
       $result1=pg_query($con, $qry1);
       while ($row = pg_fetch_row($result1)) {
               echo "<br>\n";
               echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
       }
}
else
       die('Could not connected to college: '.pg_last_error());
pg_close($con);
}
?>
```

an double os

Output

Enter the Datais of a STUDENT

Enter the Reg No to DELETE

Submit

Successfully Connected to college

regno: 102 name: raju age: 17 gender: m mark: 54
regno: 105 name: vani age: 19 gender: f mark: 67
regno: 103 name: arjun age: 20 gender: m mark: 81
regno: 108 name: kumar age: 17 gender: m mark: 55
regno: 107 name: sukanya age: 19 gender: f mark: 95
regno: 110 name: fathima age: 22 gender: f mark: 69
regno: 444 name: rthh age: 12 gender: m mark: 45

Regno 444 is now deleted

regno: 102 name: raju age: 17 gender: m mark: 54
regno: 105 name: vani age: 19 gender: f mark: 67
regno: 103 name: arjun age: 20 gender: m mark: 81
regno: 108 name: kumar age: 17 gender: m mark: 55
regno: 107 name: sukanya age: 19 gender: f mark: 95
regno: 110 name: fathima age: 22 gender: f mark: 69

DISPLAYING DATA FROM POSTRGRESQL DATABASE IN WEBPAGE

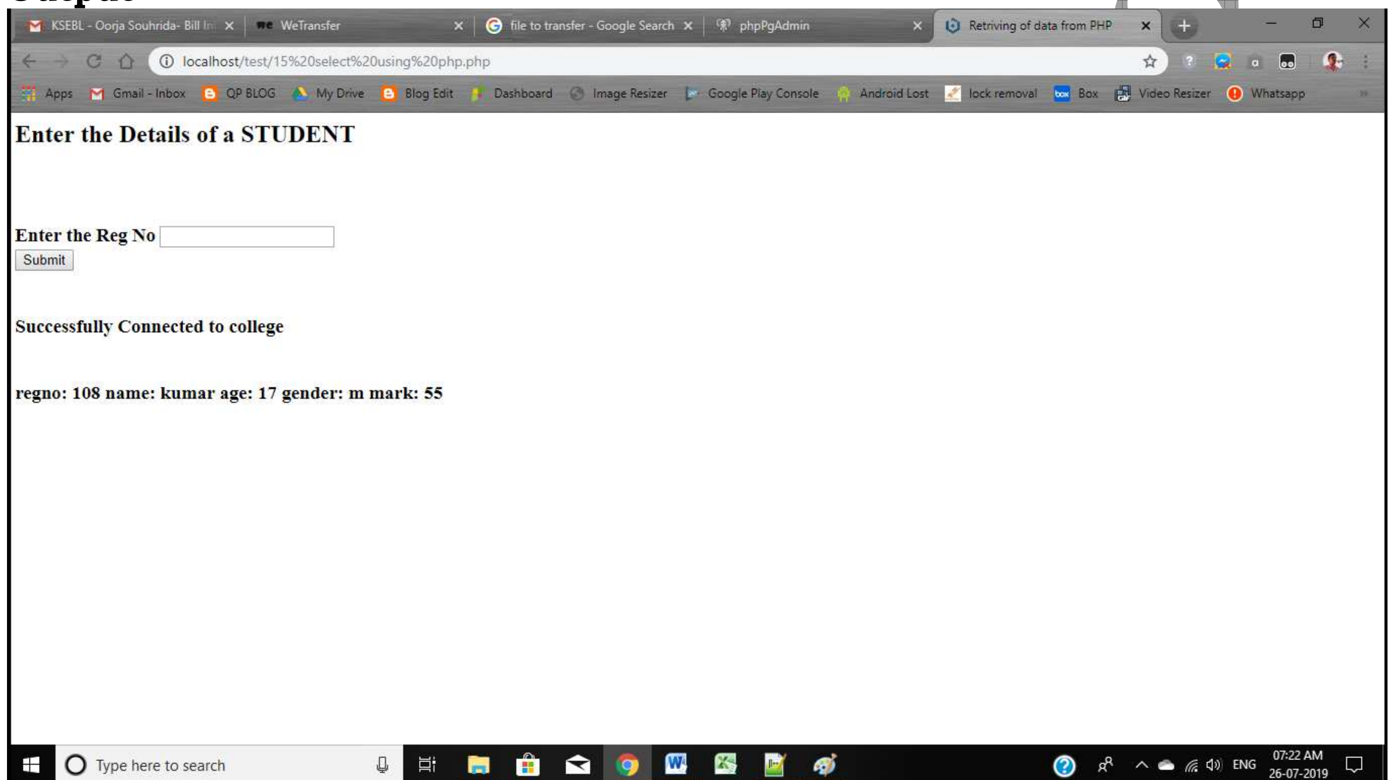
Example

```
<html><head>
<title>Retriving of data from PHP</title>
</head><body>
<form action="" method="post">
        <h2>Enter the Details of a STUDENT</h2><br></br>
        <h3>Enter the Reg No
        <input type="text" name="reg">
                 
        <br><input type="submit" />
</form></body></html>

<?php
if ($_POST){
$no=$_POST['reg'];
$db=college;
$con = pg_connect("host=localhost dbname=$db user=postgres
password=gems");
if ($con) {
    echo '<br><br><br>Successfully Connected to '. $db."<br><br>";
        $qry="select * from student where regno =$no";
        $result=pg_query($con, $qry);
        while ($row = pg_fetch_row($result)) {
```

```
                echo "<br>\n";
                echo "regno: $row[0] name: $row[1] age: $row[2] gender: $row[3]
mark: $row[4]";
        }
}
else
        die('Could not connected to college: '.pg_last_error());
pg_close($con);
}
?>
```

Output



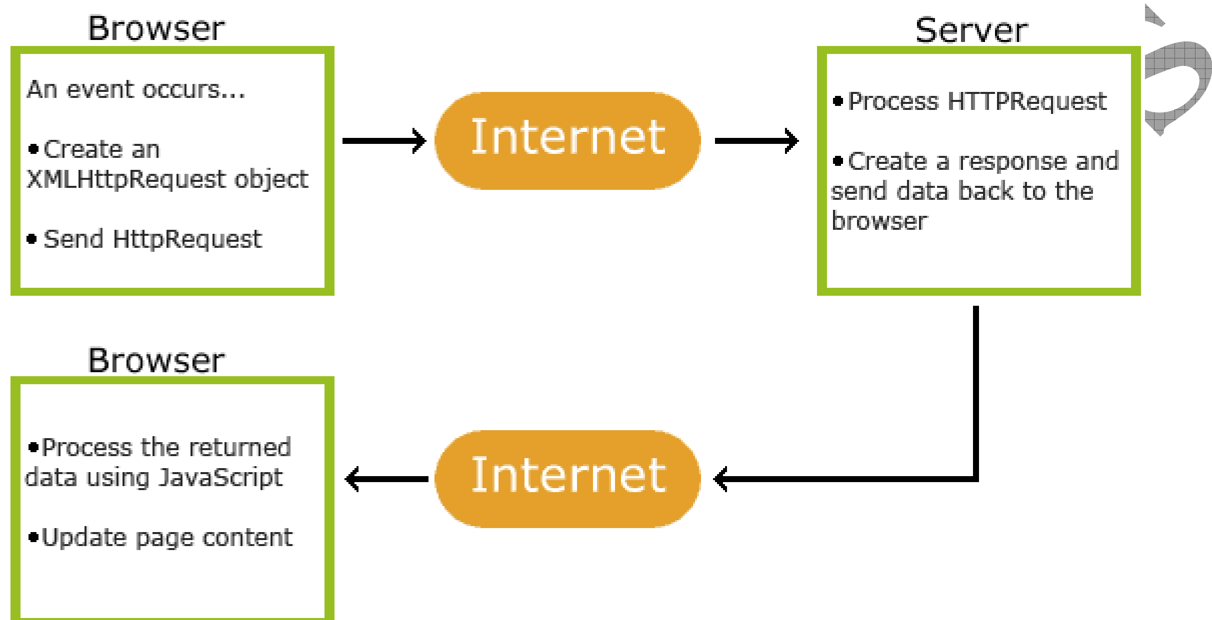
INTRODUCTION TO AJAX

AJAX is **A**synchronous **Ja**vaScript and **X**ML) is a set of web development techniques using many web technologies on the client-side to create asynchronous Web applications. Ajax is a client-side script that communicates to and from a server/database without the need for a post back or a complete page refresh. The best definition for Ajax is “the method of exchanging data with a server, and updating parts of a web page - without reloading the entire page.”

AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script. Conventional web application transmit information to and from the sever using synchronous requests. This means we fill out a form, hit submit, and get directed to a new page with new information from the server. With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the

results and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change. Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs. Following diagram shows how AJAX works.



AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent. AJAX was made popular in 2005 by Google, with Google Suggest. Google Suggest is using AJAX to create a very dynamic web interface: When we start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

IMPLEMENTATION OF AJAX IN PHP

AJAX is used to create more interactive applications. The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field:

Start typing a name in the input field below:

First name:

Suggestions:

In the example above, when a user types a character in the input field, a function called "showHint()" is executed. The function is triggered by the onkeyup event. Here is the HTML code:

```
<html> <head> <script>
function showHint(str) {
if (str.length == 0) {
document.getElementById("txtHint").innerHTML = "";
return;
} else {
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
document.getElementById("txtHint").innerHTML = this.responseText;
}
};
xmlhttp.open("GET", "gethint.php?q=" + str, true);
xmlhttp.send();
}
}
</script> </head> <body>
<p<b>Start typing a name in the input field below:</b ></p >
<form>
First name: <input  type=="text" onkeyup=="showHint(this.value)"=>
</form>
<p>Suggestions: <span  id=="txtHint"=></span ></p > </body> </html>
```

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function. However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added to the url (gethint.php?q="+str)
- And the str variable holds the content of the input field

an double os

The PHP File - "gethint.php"

The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];
$hint = "";
// lookup all hints from array if $q is different from ""
if ($q !== "") {
    $q = strtolower($q);
    $len=strlen($q);
    foreach($a as $name) {
        if (stristr($q, substr($name, 0, $len))) {
            if ($hint === "") {
                $hint = $name;
```

an double os

```
            } else {
                $hint .= ", $name";
            }
        }
    }
}
// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

an double os