SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Project Report

For

Database Management System

15IT302J

On

Travel Management System

Name : Somesh Dave

Reg. Number : RA1711020010087

Department : Software Engineering

Subject : DBMS

INDEX

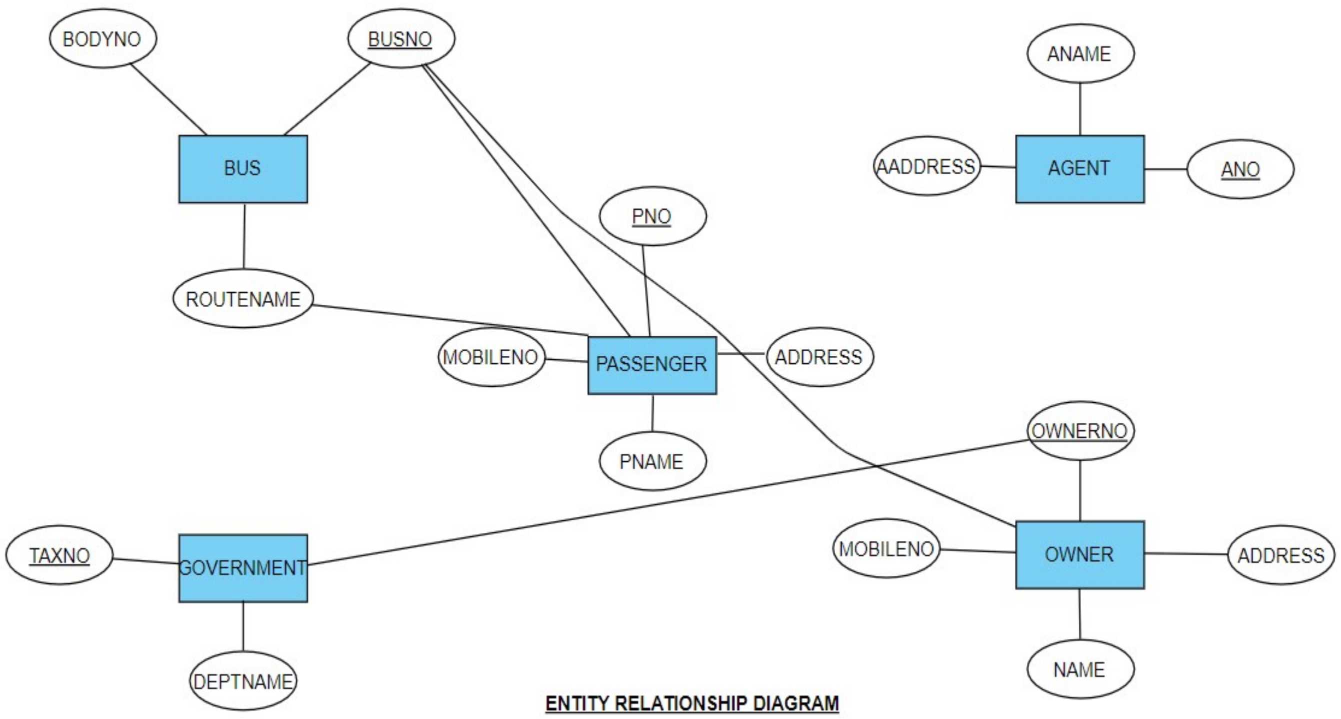
Objective

PHP along with HTML and CSS is one of the most powerful front end tools available today.

This “Travel Management System” project is developed using PHP along HTML and CSS as front end and MySql as backend.

The Travel Management System application is a complete solution for managing Bus Transportation across India. The report is organized in such way that it reflects all the facilities provided by the application, including it’s distinct features and advantages.

ER DIAGRAM



ENTITY RELATIONSHIP DIAGRAM

SQL DDL COMMANDS

Data Definition Language SQL COMMANDS

Data Definition Language (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

The Create Table Command

The create table command defines each column of the table uniquely. Each column has minimum of three attributes.

- Name
- Data type
- Size(column width).

Each table column definition is a single clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statement is terminated with a semicolon.

The Structure of Create Table Command

Example:

```
CREATE TABLE Student
             (Reg_no varchar2(10),
              Name char(30),
              DOB date,
              Address varchar2(50));
```

The DROP Command

Syntax:

```
DROP TABLE <table_name>
```

Example:

```
DROP TABLE Student;
```

It will destroy the table and all data which will be recorded in it.

The TRUNCATE Command

Syntax:

```
TRUNCATE TABLE <Table_name>
```

Example:

```
TRUNCATE TABLE Student;
```

The RENAME Command

Syntax:

```
RENAME <OldTableName> TO <NewTableName>
```

Example:

```
RENAME <Student> TO <Stu>
```

The old name table was **Student** now new name is the **Stu.**

The ALTER Table Command

By The use of ALTER TABLE Command we can **modify** our exiting table.

Adding New Columns

Syntax:

```
ALTER TABLE <table_name>
        ADD (<NewColumnName> <Data_Type>(<size>),......n)
```

Example:

```
ALTER TABLE Student ADD (Age number(2), Marks number(3));
```

The Student table is already exist and then we added two more columns **Age** and **Marks** respectively, by the use of above command.

Dropping a Column from the Table

Syntax:

```
ALTER TABLE <table_name> DROP COLUMN <column_name>
```

Example:

```
ALTER TABLE Student DROP COLUMN Age;
```

This command will drop particular column

Modifying Existing Table

Syntax:

```
ALTER TABLE <table_name> MODIFY (<column_name> <NewDataType>(<NewSize>))
```

Example:

```
ALTER TABLE Student MODIFY (Name Varchar2(40));
```

The Name column already exist in Student table, it was char and size 30, now it is modified by Varchar2 and size 40.

Restriction on the ALTER TABLE

Using the ALTER TABLE clause the following tasks cannot be performed.

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

Perform the all the above DDL operations on the sample table given below…

SELECT * FROM STATION;

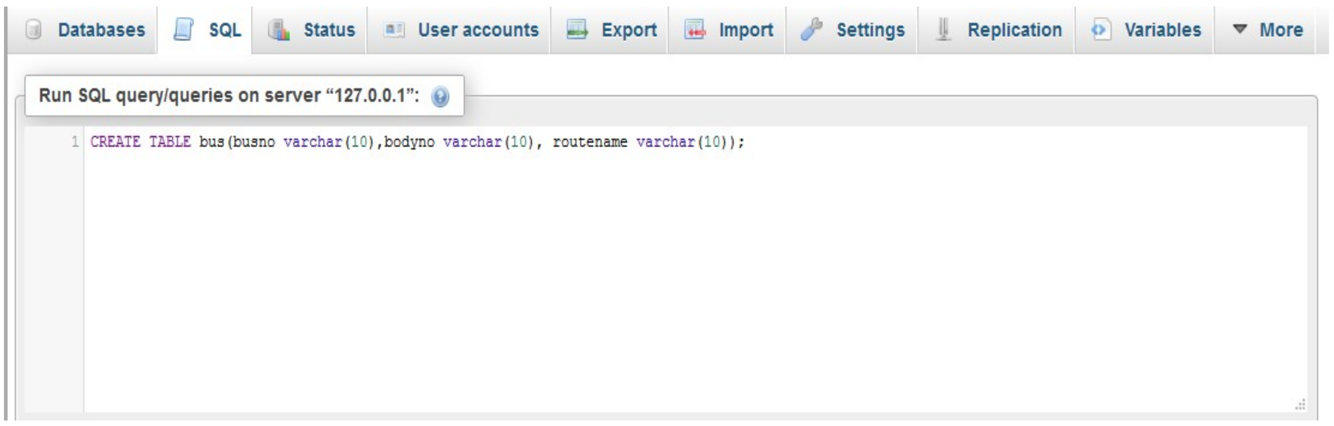
SELECT * FROM employee;

SELECT * FROM STATION;

SELECT * FROM CUSTOMER;

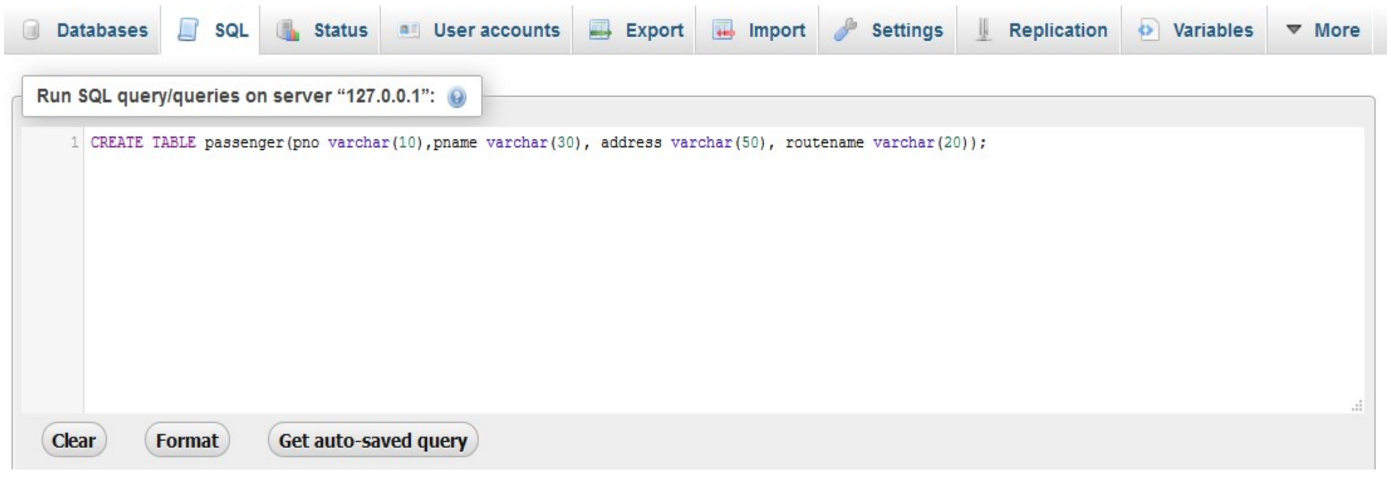
SELECT * FROM FLYER;

SELECT * FROM SHOWBOOKING;



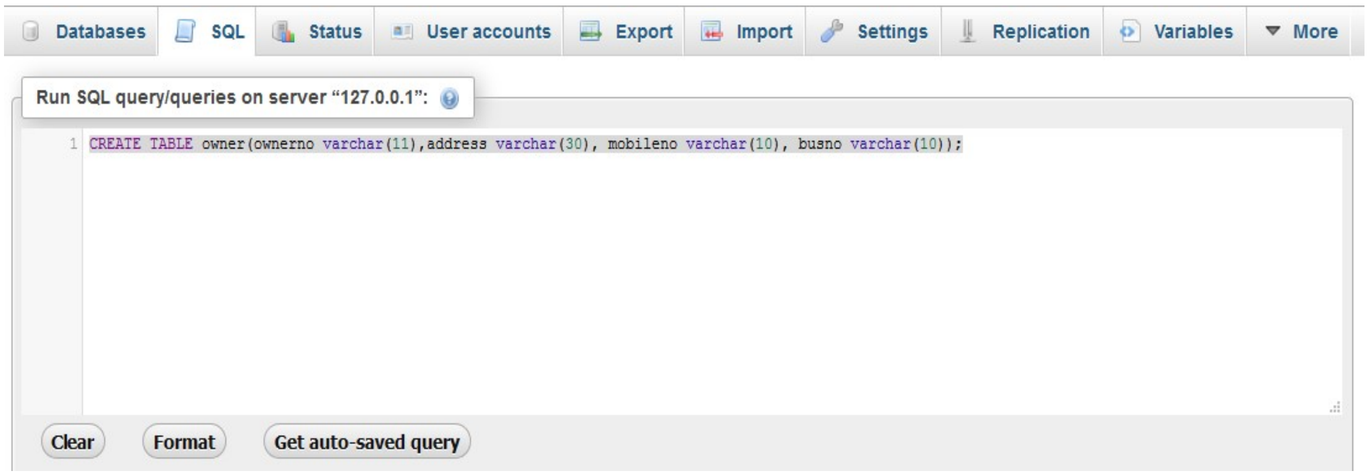
```
CREATE TABLE bus(busno varchar(10),bodyno varchar(10), routename varchar(10));
```

CREATE TABLE bus(busno varchar(10),bodyno varchar(10), routename varchar(10));



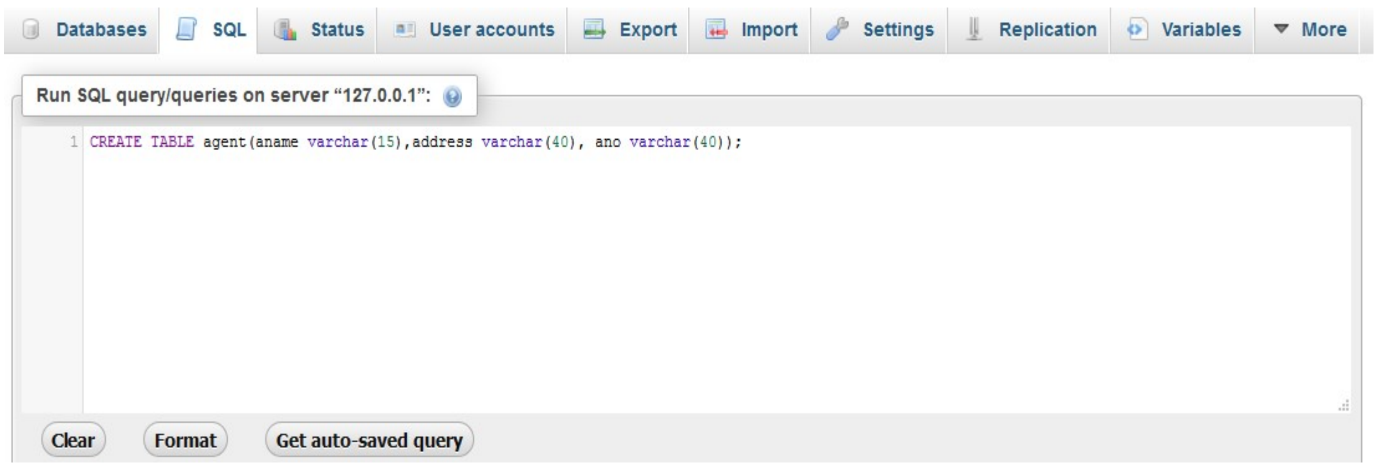
```
CREATE TABLE passenger(pno varchar(10),pname varchar(30), address varchar(50), routename varchar(20));
```

CREATE TABLE passenger(pno varchar(10),pname varchar(30), address varchar(50), routename varchar(20));



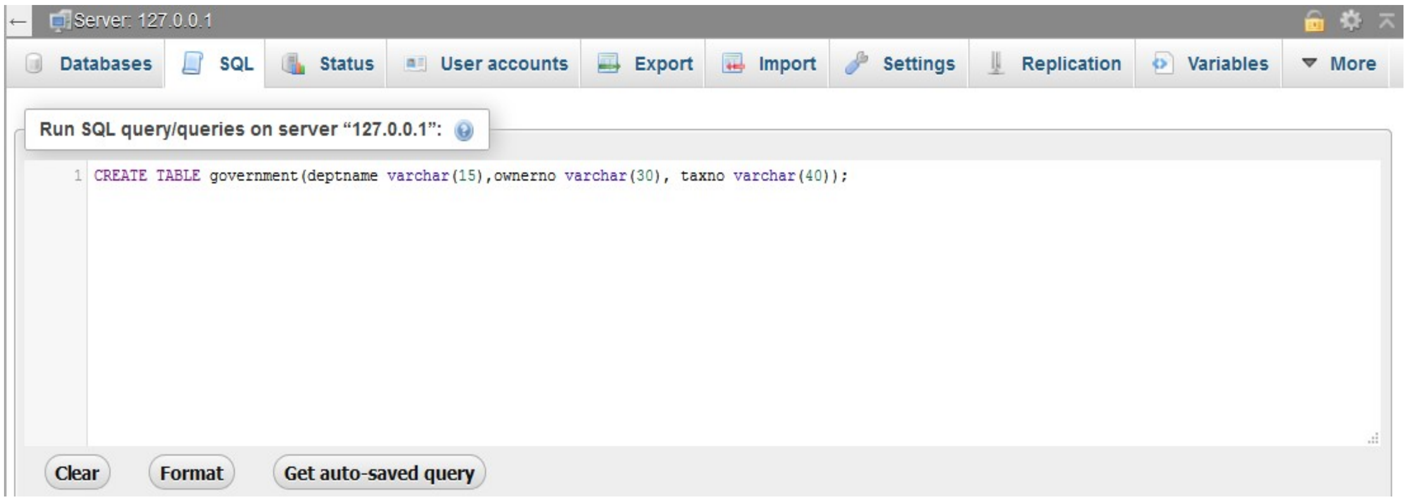
```sql
CREATE TABLE owner(ownerno varchar(11),address varchar(30), mobileno varchar(10), busno varchar(10));
```

CREATE TABLE owner(ownerno varchar(11),address varchar(30), mobileno varchar(10), busno varchar(10));



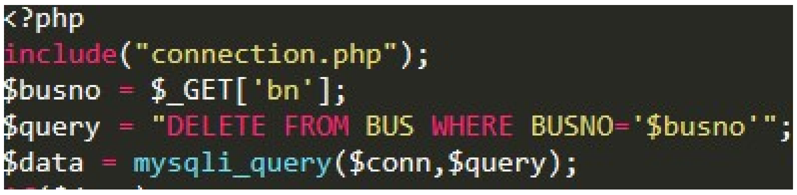
```sql
CREATE TABLE agent(aname varchar(15),address varchar(40), ano varchar(40));
```

CREATE TABLE agent(aname varchar(15),address varchar(40), ano varchar(40));



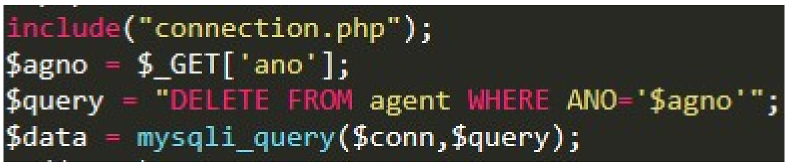
```sql
CREATE TABLE government(deptname varchar(15),ownerno varchar(30), taxno varchar(40));
```

CREATE TABLE government(deptname varchar(15),ownerno varchar(30), taxno varchar(40));



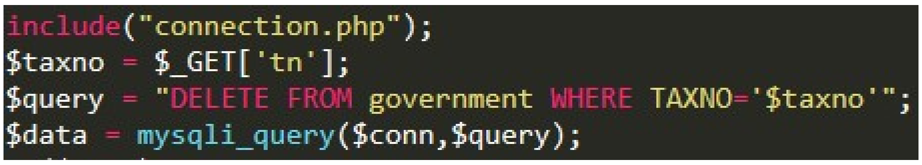
```php
<?php
include("connection.php");
$busno = $_GET['bn'];
$query = "DELETE FROM BUS WHERE BUSNO='$busno'";
$data = mysqli_query($conn,$query);
```

DELETE FROM BUS WHERE BUSNO='$busno'



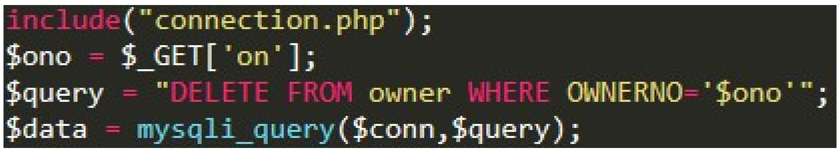
```
include("connection.php");
$agno = $_GET['ano'];
$query = "DELETE FROM agent WHERE ANO='$agno'";
$data = mysqli_query($conn,$query);
```

DELETE FROM agent WHERE ANO='$agno'



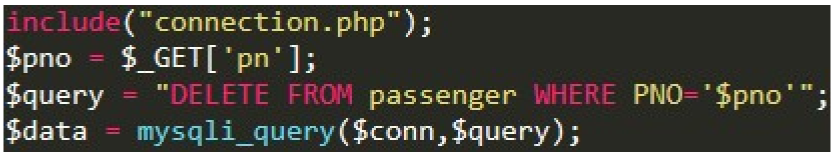
```
include("connection.php");
$taxno = $_GET['tn'];
$query = "DELETE FROM government WHERE TAXNO='$taxno'";
$data = mysqli_query($conn,$query);
```

DELETE FROM government WHERE TAXNO='$taxno'



```
include("connection.php");
$ono = $_GET['on'];
$query = "DELETE FROM owner WHERE OWNERNO='$ono'";
$data = mysqli_query($conn,$query);
```

DELETE FROM owner WHERE OWNERNO='$ono'



```
include("connection.php");
$pno = $_GET['pn'];
$query = "DELETE FROM passenger WHERE PNO='$pno'";
$data = mysqli_query($conn,$query);
```

DELETE FROM passenger WHERE PNO='$pno'

SQL DML COMMANDS

DML (Data Manipulation Language)

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:

SELECT – select records from a table
INSERT – insert new records
UPDATE – update/Modify existing records
DELETE – delete existing records

DML command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

INSERT COMMAND

Insert command is used to insert data into a table. Following is its general syntax,

INSERT into *table-name* values(data1,data2,..)

Example,

Consider a table **Student** with following fields.

```
INSERT  into  Student  values  (101,'Adam',15);
```

The above command will insert a record into **Student** table.

Example to Insert NULL value to a column

Example to Insert NULL Value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

```
INSERT  into  Student(id,name)  values(102,'Alex');
```

Or,

```
INSERT  into  Student  values(102,'Alex',null);
```

The above command will insert only two column value other column is set to null.

Example to Insert Default value to a column

```
INSERT into Student values(103,'Chris',default)
```

Suppose the **age** column of student table has default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

```
INSERT into Student values(103,'Chris')
```

UPDATE COMMAND

Update command is used to update a row of a table. Following is its general syntax,

UPDATE *table-name* set column-name = value *where* **condition**;

Lets see an example,

```
update Student set age=18 where s_id=102;
```

Example to Update multiple columns

```
UPDATE Student set s_name='Abhi',age=17 where s_id=103;
```

The above command will update two columns of a record.

DELETE COMMAND

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

DELETE from *table-name*;

Example to Delete all Records from a Table

```
DELETE from Student;
```

The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table

```
DELETE from Student where s_id=103;
```

The above command will delete the record where s_id is 103 from **Student** table.

WHERE clause

Where clause is used to specify condition while retriving data from table. ***Where*** clause is used mostly with *Select*, *Update* and *Delete* query. If condititon specified by ***where*** clause is true then only the result from table is returned.

Syntax for WHERE clause

SELECT column-name1,
 column-name2,
 column-name3,
 column-nameN
from table-name **WHERE** **[condition]**;

Example using WHERE clause

Consider a **Student** table,

Now we will use a SELECT statement to display data of the table, based on a condition, which we will add to the SELECT query using WHERE clause.

```
SELECT  s_id,
  s_name,
  age,
  address
from  Student  WHERE  s_id=101;
```

SELECT COMMAND

SELECT Query

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query

SELECT column-name1, column-name2, column-name3, column-nameN from *table-name*;

Example for SELECT Query

Consider the following **Student** table,

```
SELECT  s_id,  s_name,  age  from  Student;
```

The above query will fetch information of s_id, s_name and age column from Student table

---

Example to Select all Records from Table

A special character **asterisk** * is used to address all the data(belonging to all columns) in a query. *SELECT* statement uses * character to retrieve all records from a table.

```
SELECT  *  from  student;
```

The above query will show all the records of Student table, that means it will show complete Student table as result.

Example to Select particular Record based on Condition

```
SELECT  *  from  Student  WHERE  s_name  =  'Abhi';
```

Example to Perform Simple Calculations using Select Query

Conside the following **Employee** table.

```
SELECT  eid,  name,  salary+3000    from  Employee;
```

The above command will display a new column in the result, showing 3000 added into existing salaries of the employees.

Like Clause

Like clause is used as condition in SQL query. **Like** clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

Wildcard operators

There are two wildcard operators that are used in like clause.

- **Percent sign %** : represents zero, one or more than one character.
- **Underscore sign** _ : represents only one character.

Example of LIKE clause

Consider the following **Student** table.

```
SELECT * from Student where s_name like 'A%';
```

The above query will return all records where **s_name** starts with character 'A'.

Example

```
SELECT * from Student where s_name like '_d%';
```

The above query will return all records from **Student** table where **s_name** contain 'd' as second character.

Example

```
SELECT * from Student where s_name like '%x';
```

The above query will return all records from **Student** table where **s_name** contain 'x' as last character.

Order By Clause

Order by clause is used with **Select** statement for arranging retrieved data in sorted order. The **Order by**clause by default sort data in ascending order. To sort data in descending order **DESC** keyword is used with**Order by** clause.

Syntax of Order By

```
SELECT column-list|* from table-name order by asc|desc;
```

Example using Order by

Consider the following **Emp** table,

```
SELECT * from Emp order by salary;
```

The above query will return result in ascending order of the **salary**.

Example of Order by DESC

Consider the **Emp** table described above,

```
SELECT * from Emp order by salary DESC;
```

The above query will return result in descending order of the **salary**.

Group By Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax for using Group by in a statement.

```
SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

Example of Group by in a Statement

Consider the following **Emp** table.

Here we want to find name and age of employees grouped by their salaries

SQL query for the above requirement will be,

```
SELECT name, age from Emp group by salary;
```

Result will be,

Example of Group by in a Statement with WHERE clause

Consider the following **Emp** table

SQL query will be,

```
select name, salary from Emp where age > 25group by salary;
```

Result will be.

You must remember that Group By clause will always come at the end, just like the Order by clause.

HAVING Clause

having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group based SQL functions, just like WHERE clause.

Syntax for having will be,

```
select column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
```

HAVING function(column_name) condition

Example of HAVING Statement

Consider the following **Sale** table.

Suppose we want to find the customer whose previous_balance sum is more than 3000.

We will use the below SQL query,

```
SELECT * from sale group customer having sum(previous_balance) > 3000;
```

Result will be,

Distinct clause

The **distinct** keyword is used with **Select** statement to retrieve unique values from the table. **Distinct**removes all the duplicate records while retrieving from database.

Syntax for DISTINCT Keyword

SELECT *distinct* column-name from *table-name*;

Example

Consider the following **Emp** table.

```
select distinct salary from Emp;
```

The above query will return only the unique salary from **Emp** table

AND & OR clause

AND and **OR** operators are used with **Where** clause to make more precise conditions for fetching data from database by combining more than one condition together.

AND operator

AND operator is used to set multiple conditions with *Where* clause.

Example of AND

Consider the following **Emp** table

SELECT * from Emp WHERE salary < 10000 **AND** age > 25 ;

The above query will return records where salary is less than 10000 and age greater than 25.

OR operator

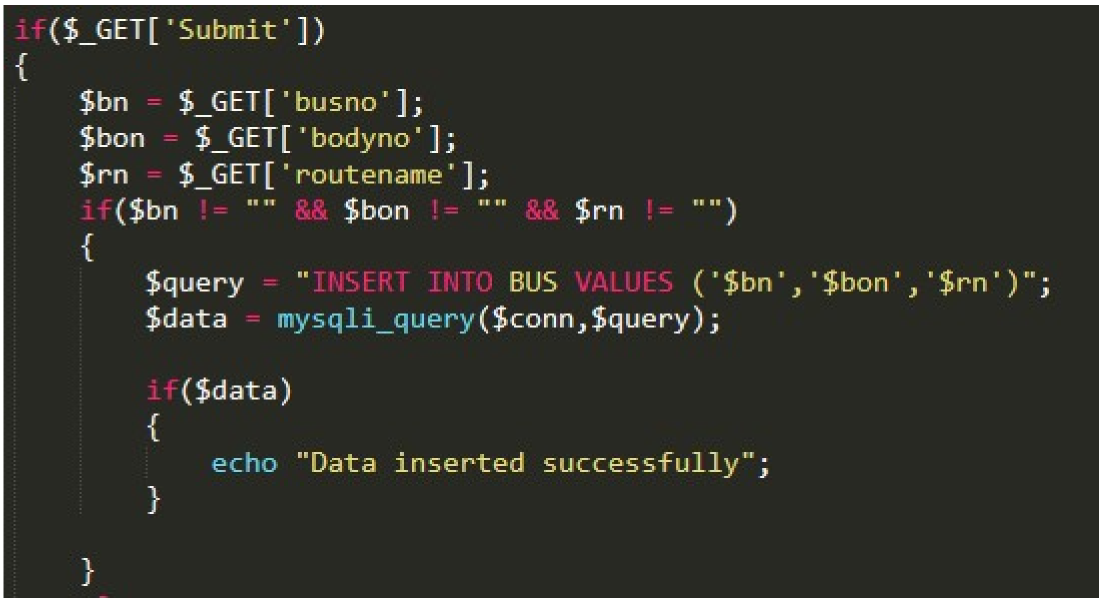
OR operator is also used to combine multiple conditions with *Where* clause. The only difference between AND and OR is their behaviour. When we use AND to combine two or more than two conditions, records satisfying all the condition will be in the result. But in case of OR, atleast one condition from the conditions specified must be satisfied by any record to be in the result.

Example of OR

Consider the following **Emp** table

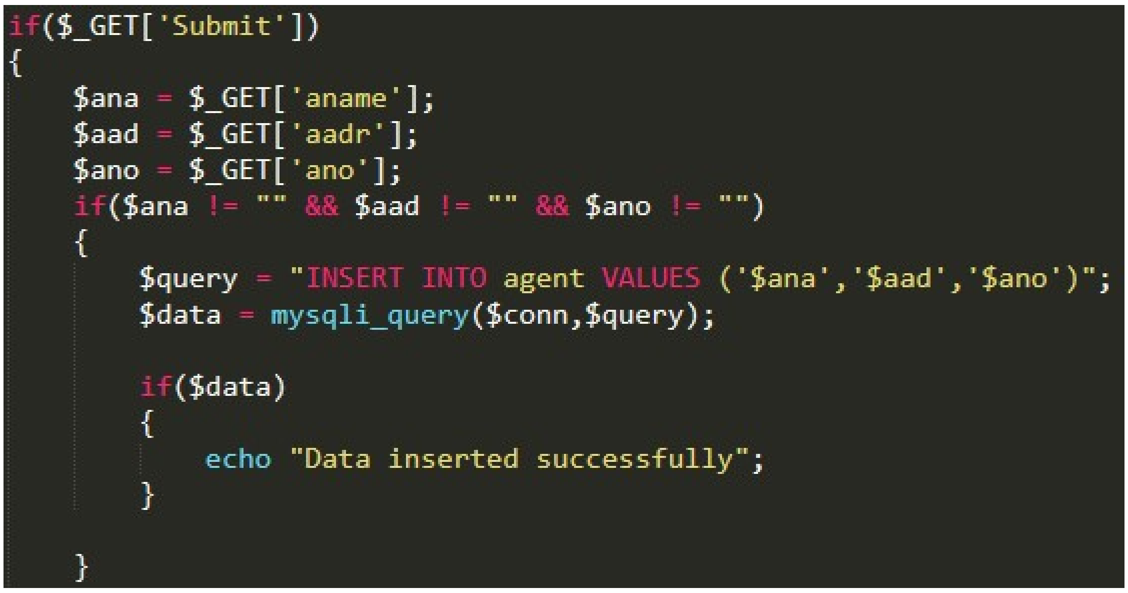
```
SELECT * from Emp WHERE salary > 10000 OR age > 25 ;
```

The above query will return records where either salary is greater than 10000 or age greater than 25.



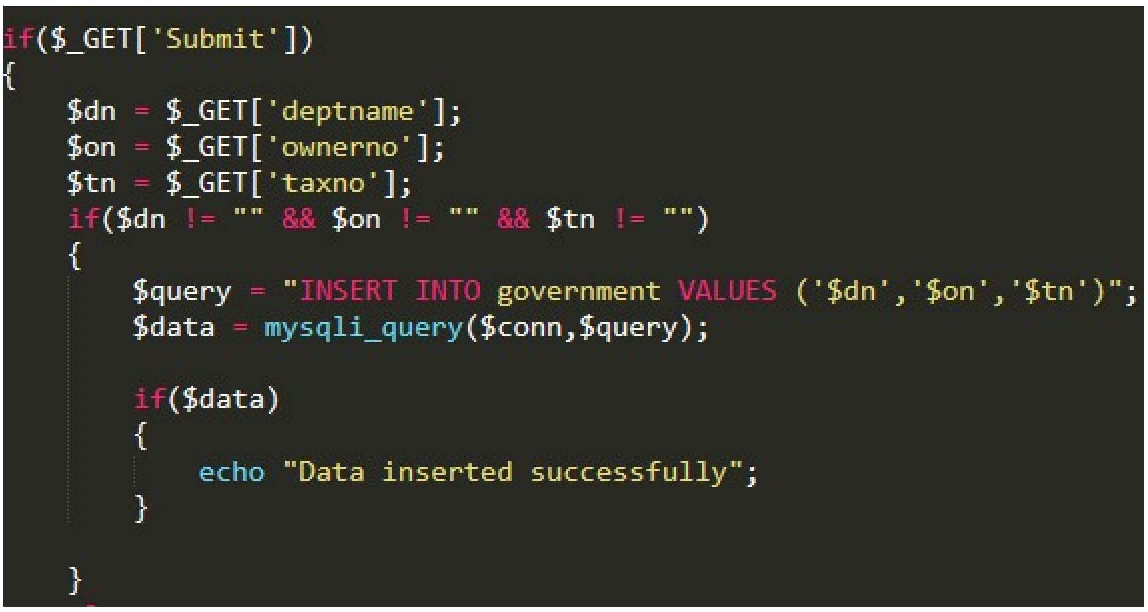
```
if($_GET['Submit'])
{
    $bn = $_GET['busno'];
    $bon = $_GET['bodyno'];
    $rn = $_GET['routename'];
    if($bn != "" && $bon != "" && $rn != "")
    {
        $query = "INSERT INTO BUS VALUES ('$bn','$bon','$rn')";
        $data = mysqli_query($conn,$query);

        if($data)
        {
            echo "Data inserted successfully";
        }

    }
```

INSERT INTO BUS VALUES ('$bn','$bon','$rn')



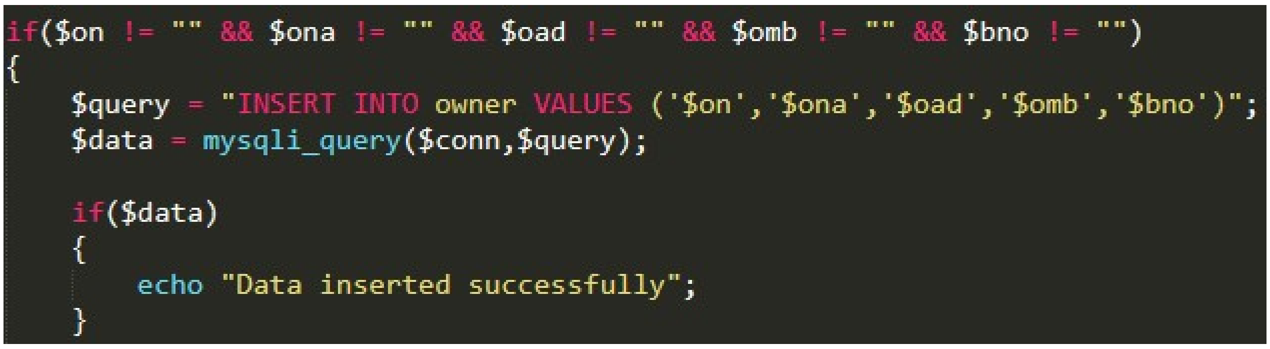
```
if($_GET['Submit'])
{
    $ana = $_GET['aname'];
    $aad = $_GET['aadr'];
    $ano = $_GET['ano'];
    if($ana != "" && $aad != "" && $ano != "")
    {
        $query = "INSERT INTO agent VALUES ('$ana','$aad','$ano')";
        $data = mysqli_query($conn,$query);

        if($data)
        {
            echo "Data inserted successfully";
        }

    }
```

INSERT INTO agent VALUES ('$ana','$aad','$ano')



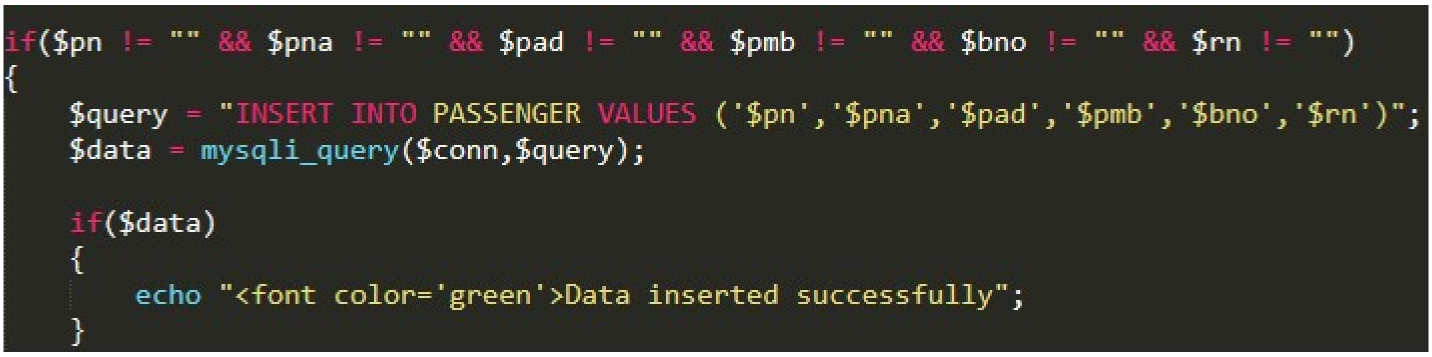
```
if($_GET['Submit'])
{
    $dn = $_GET['deptname'];
    $on = $_GET['ownerno'];
    $tn = $_GET['taxno'];
    if($dn != "" && $on != "" && $tn != "")
    {
        $query = "INSERT INTO government VALUES ('$dn','$on','$tn')";
        $data = mysqli_query($conn,$query);

        if($data)
        {
            echo "Data inserted successfully";
        }

    }
```

INSERT INTO government VALUES ('$dn','$on','$tn')



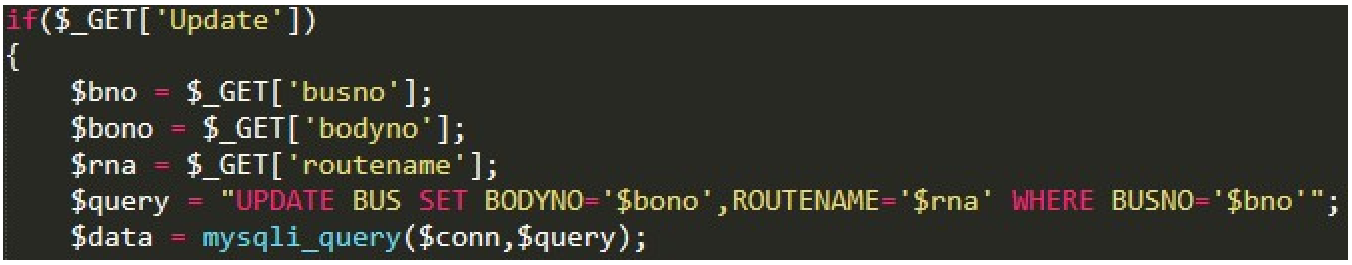
```
if($on != "" && $ona != "" && $oad != "" && $omb != "" && $bno != "")
{
    $query = "INSERT INTO owner VALUES ('$on','$ona','$oad','$omb','$bno')";
    $data = mysqli_query($conn,$query);

    if($data)
    {
        echo "Data inserted successfully";
    }
```

INSERT INTO owner VALUES ('$on','$ona','$oad','$omb','$bno')



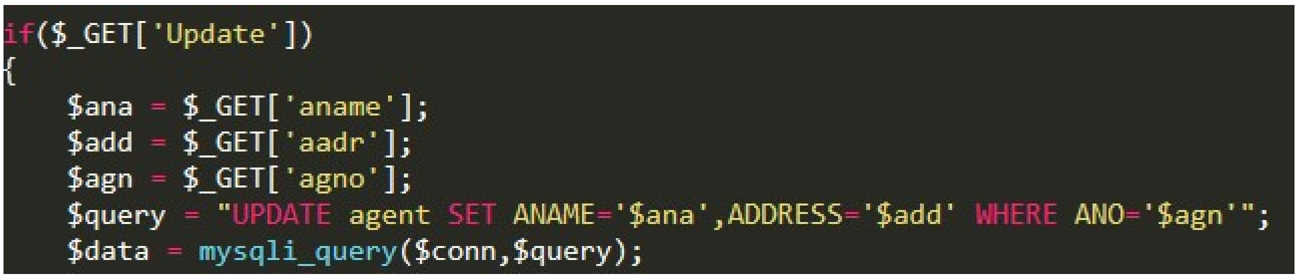
```
if($pn != "" && $pna != "" && $pad != "" && $pmb != "" && $bno != "" && $rn != "")
{
    $query = "INSERT INTO PASSENGER VALUES ('$pn','$pna','$pad','$pmb','$bno','$rn')";
    $data = mysqli_query($conn,$query);

    if($data)
    {
        echo "<font color='green'>Data inserted successfully";
    }
```

INSERT INTO PASSENGER VALUES ('$pn','$pna','$pad','$pmb','$bno','$rn')



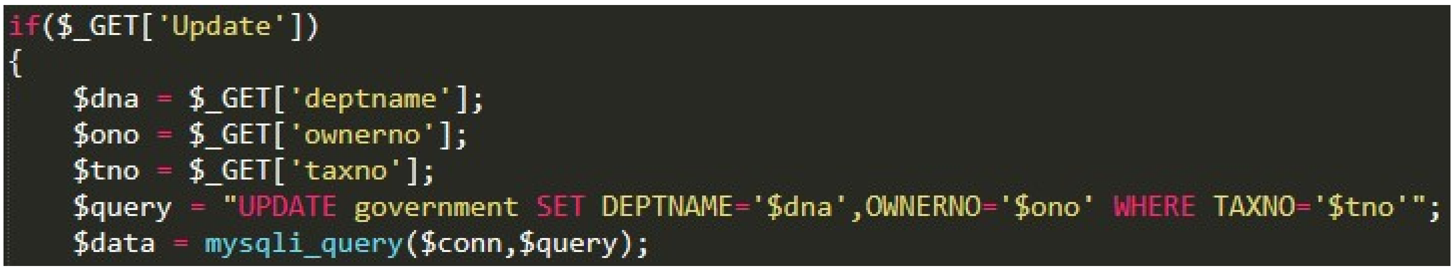
```
if($_GET['Update'])
{
    $bno = $_GET['busno'];
    $bono = $_GET['bodyno'];
    $rna = $_GET['routename'];
    $query = "UPDATE BUS SET BODYNO='$bono',ROUTENAME='$rna' WHERE BUSNO='$bno'";
    $data = mysqli_query($conn,$query);
```

UPDATE BUS SET BODYNO='$bono',ROUTENAME='$rna' WHERE BUSNO='$bno'



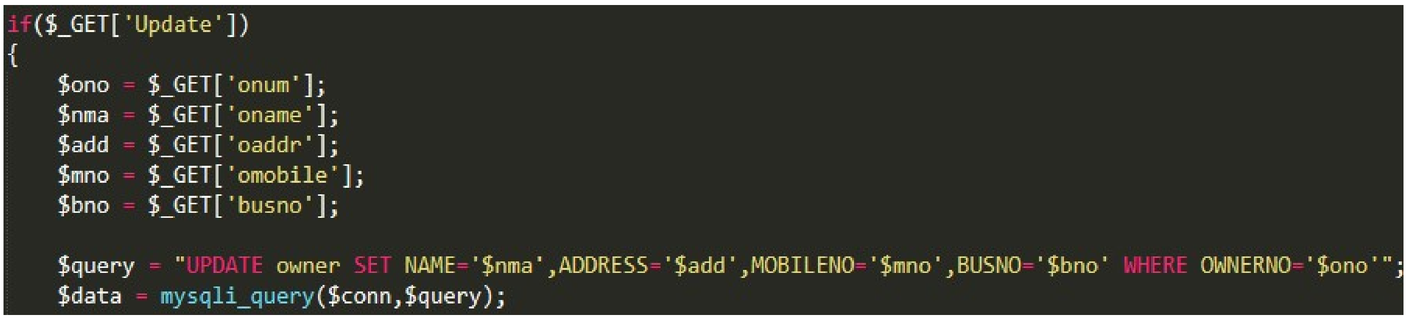
```
if($_GET['Update'])
{
    $ana = $_GET['aname'];
    $add = $_GET['aadr'];
    $agn = $_GET['agno'];
    $query = "UPDATE agent SET ANAME='$ana',ADDRESS='$add' WHERE ANO='$agn'";
    $data = mysqli_query($conn,$query);
```

UPDATE agent SET ANAME='$ana',ADDRESS='$add' WHERE ANO='$agn'



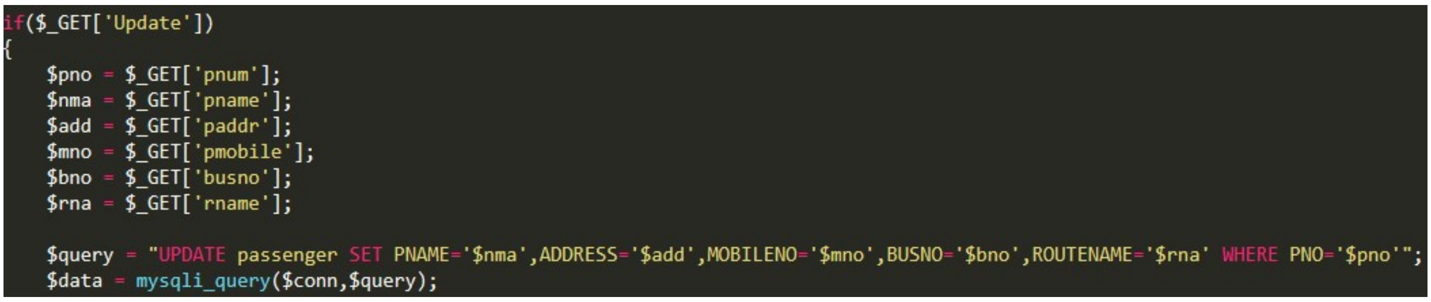
```
if($_GET['Update'])
{
    $dna = $_GET['deptname'];
    $ono = $_GET['ownerno'];
    $tno = $_GET['taxno'];
    $query = "UPDATE government SET DEPTNAME='$dna',OWNERNO='$ono' WHERE TAXNO='$tno'";
    $data = mysqli_query($conn,$query);
```

UPDATE government SET DEPTNAME='$dna',OWNERNO='$ono' WHERE TAXNO='$tno'



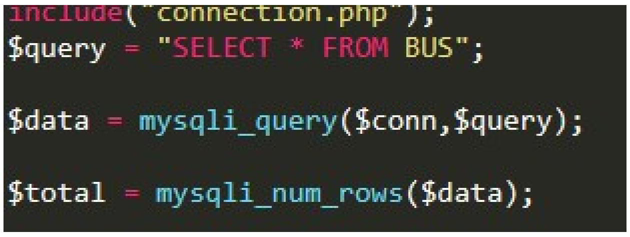
```
if($_GET['Update'])
{
    $ono = $_GET['onum'];
    $nma = $_GET['oname'];
    $add = $_GET['oaddr'];
    $mno = $_GET['omobile'];
    $bno = $_GET['busno'];

    $query = "UPDATE owner SET NAME='$nma',ADDRESS='$add',MOBILENO='$mno',BUSNO='$bno' WHERE OWNERNO='$ono'";
    $data = mysqli_query($conn,$query);
```

UPDATE owner SET NAME='$nma',ADDRESS='$add',MOBILENO='$mno',BUSNO='$bno' WHERE OWNERNO='$ono'



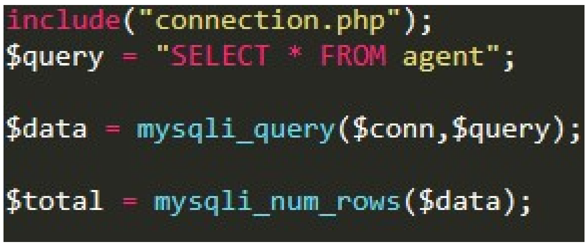
```
if($_GET['Update'])
{
    $pno = $_GET['pnum'];
    $nma = $_GET['pname'];
    $add = $_GET['paddr'];
    $mno = $_GET['pmobile'];
    $bno = $_GET['busno'];
    $rna = $_GET['rname'];

    $query = "UPDATE passenger SET PNAME='$nma',ADDRESS='$add',MOBILENO='$mno',BUSNO='$bno',ROUTENAME='$rna' WHERE PNO='$pno'";
    $data = mysqli_query($conn,$query);
```

UPDATE passenger SET PNAME='$nma',ADDRESS='$add',MOBILENO='$mno',BUSNO='$bno',ROUTENAME='$rna' WHERE PNO='$pno'



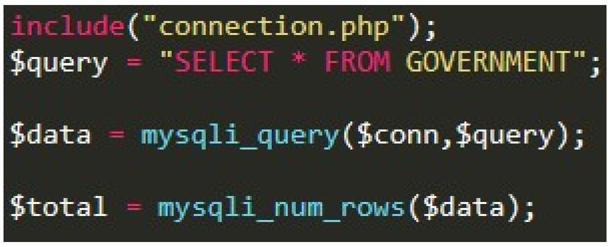
```
include("connection.php");
$query = "SELECT * FROM BUS";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM BUS



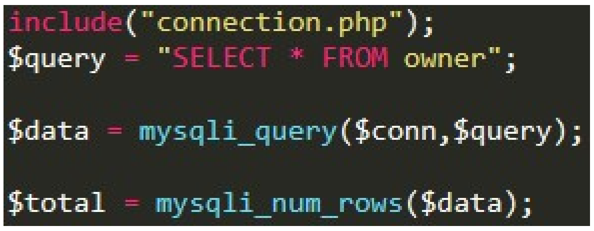
```
include("connection.php");
$query = "SELECT * FROM agent";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM agent



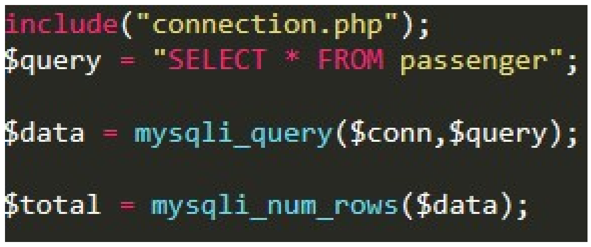
```
include("connection.php");
$query = "SELECT * FROM GOVERNMENT";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM GOVERNMENT



```
include("connection.php");
$query = "SELECT * FROM owner";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM owner



```
include("connection.php");
$query = "SELECT * FROM passenger";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM passenger

INBUILT FUNCTIONS

Built-In functions in SQL

Functions
Function accept zero or more arguments and both return one or more results. Both are used to manipulate individual data items.Operators differ from functional in that they follow the format of function_name(arg..). Function can be classifies into **single row function and group functions**.

Single Row functions

The single row function can be broadly classified as,
o Date Function
o Numeric Function
o Character Function
o Conversion Function
o Miscellaneous Function

The example that follows mostly uses the symbol table "**dual**". It is a table, which is automatically created by oracle along with the data dictionary

Date Function

1. Add_month
This function returns a date after adding a specified date with specified number of months.

Syntax: Add_months(d,n); where d-date n-number of months
Example: *Select add_months(sysdate,2) from dual;*

2. last_day
It displays the last date of that month.

Syntax: last_day (d); where d-date
Example: *Select last_day ('1-jun-2009') from dual;*

3. Months_between
It gives the difference in number of months between d1 & d2.

Syntax: month_between (d1,d2); where d1 & d2 –dates
Example: *Select month_between ('1-jun-2009','1-aug-2009') from dual;*

4. next_day
It returns a day followed the specified date.

Syntax: next_day (d,day);
Example: *Select next_day (sysdate,'wednesday') from dual*

5. round
This function returns the date, which is rounded to the unit specified by the format model.
Syntax : round (d,[fmt]);
where d- date, [fmt] – optional. By default date will be rounded to the nearest day
Example: *Select round (to_date('1-jun-2009','dd-mm-yy'),'year') from dual;*
Select round ('1-jun-2009','year') from dual;
Numerical Functions

Conversion Function

1. to_char()

Syntax: to_char(d,[format]);
This function converts date to a value of varchar type in a form specified by date format. If format is negelected then it converts date to varchar2 in the default date format.
Example: *select to_char (sysdate, 'dd-mm-yy') from dual;*

2. to_date()

Syntax: to_date(d,[format]);
This function converts character to date data format specified in the form character.
Example: *select to_date('aug 15 2009','mm-dd-yy') from dual;*

Miscellaneous Functions

1. uid – This function returns the integer value (id) corresponding to the user currently logged in.
Example: *select uid from dual;*

2. user – This function returns the logins user name.
Example: *select user from dual;*

3. nvl – The null value function is mainly used in the case where we want to consider null values as zero.
Syntax; nvl(exp1, exp2)
If exp1 is null, return exp2. If exp1 is not null, return exp1.

Example: *select custid, shipdate, nvl(total,0) from order;*

4. vsize: It returns the number of bytes in expression.

Example: *select vsize('tech') from dual;*

Group Functions
A group function returns a result based on group of rows.

1. avg
Example: *select avg (total) from student;*

2.max
Example: *select max (percentagel) from student;*

3.min
Example: *select min (marksl) from student;*

4. sum
Example: *select sum(price) from product;*

Count Function
In order to count the number of rows, count function is used.

1. count(*) – It counts all, inclusive of duplicates and nulls.
Example: *select count(*) from student;*

2. count(col_name)– It avoids null value.

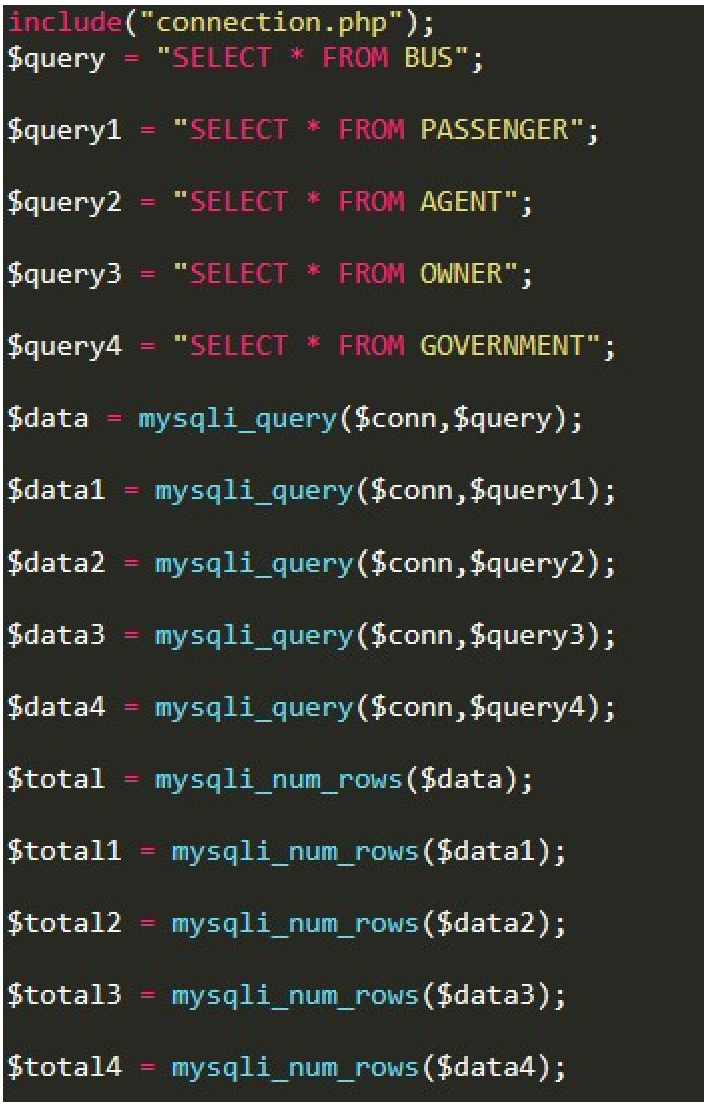
Example: *select count(total) from order;*

3. count(distinct col_name) – It avoids the repeated and null values.
Example: *select count(distinct ordid) from order;*

Group by clause
This allows us to use simultaneous column name and group functions.
Example: *Select max(percentage), deptname from student group by deptname;*

Having clause
This is used to specify conditions on rows retrieved by using group by clause.
Example: *Select max(percentage), deptname from student group by deptname having count(*)>=50;*

Special Operators:
In / not in – used to select a equi from a specific set of values
Any - used to compare with a specific set of values
Between / not between – used to find between the ranges
Like / not like – used to do the pattern matching



```
include("connection.php");
$query = "SELECT * FROM BUS";

$query1 = "SELECT * FROM PASSENGER";

$query2 = "SELECT * FROM AGENT";

$query3 = "SELECT * FROM OWNER";

$query4 = "SELECT * FROM GOVERNMENT";

$data = mysqli_query($conn,$query);

$data1 = mysqli_query($conn,$query1);

$data2 = mysqli_query($conn,$query2);

$data3 = mysqli_query($conn,$query3);

$data4 = mysqli_query($conn,$query4);

$total = mysqli_num_rows($data);

$total1 = mysqli_num_rows($data1);

$total2 = mysqli_num_rows($data2);

$total3 = mysqli_num_rows($data3);

$total4 = mysqli_num_rows($data4);
```

COUNT

JOIN QUERIES

JOIN QUERIES

Join in SQL

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. **Join** Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself known as, **Self Join**.

Types of Join
The following are the types of JOIN that we can use in SQL.

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product
This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.
Cross JOIN Syntax is,

SELECT column-name-list
from *table-name1*
CROSS JOIN
table-name2;

Example of Cross JOIN
The **class** table,
The **class_info** table,
Cross JOIN query will be,

```
SELECT *
  from class,
  cross JOIN class_info;
```

The result table will look like,

INNER Join or EQUI Join
This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.
Inner Join Syntax is,

SELECT column-name-list
from *table-name1*
INNER JOIN
table-name2
WHERE table-name1.column-name = table-name2.column-name;

Example of Inner JOIN
The **class** table,
The **class_info** table,
Inner JOIN query will be,

```
SELECT * from class, class_info where class.id = class_info.id;
```

The result table will look like,

Natural JOIN
Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

Natural Join Syntax is,

```
SELECT *
from table-name1
NATURAL JOIN
table-name2;
```

Example of Natural JOIN

The **class** table,

The **class_info** table,

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

The result table will look like,

In the above example, both the tables being joined have ID column(same name and same datatype), hence the records for which value of ID matches in both the tables will be the result of Natural Join of these two tables.

Outer JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Join

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.

Left Outer Join syntax is,

```
SELECT column-name-list
from table-name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Left outer Join Syntax for **Oracle** is,

```
select column-name-list
from table-name1,
table-name2
on table-name1.column-name = table-name2.column-name(+);
```

Example of Left Outer Join

The **class** table,

The **class_info** table,

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id=class_info.id);
```

The result table will look like,

Right Outer Join

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.

Right Outer Join Syntax is,

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Example of Right Outer Join

The **class** table,

The **class_info** table,

Right Outer Join query will be,

```
SELECT * FROM class RIGHT OUTER JOIN class_info on (class.id=class_info.id);
```

The result table will look like,

Full Outer Join

The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

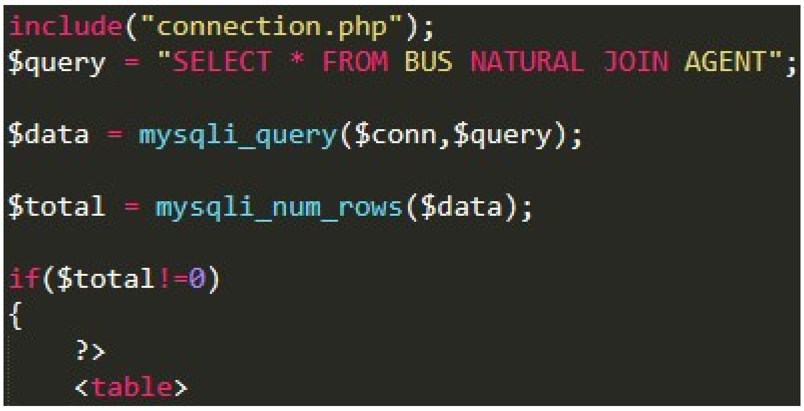
Full Outer Join Syntax is,

select column-name-list
from *table-name1*
FULL OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;

Example of Full outer join is,

Full Outer Join query will be like,

SELECT * FROM class FULL OUTER JOIN class_info on (class.id=class_info.id);



```
include("connection.php");
$query = "SELECT * FROM BUS NATURAL JOIN AGENT";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);

if($total!=0)
{
    ?>
    <table>
```

SELECT * FROM BUS NATURAL JOIN AGENT

SUBQUERIES

SQL SUBQUERIES

Subquery or **Inner query** or **Nested query** is a query in a query. SQL subquery is usually added in the WHERE Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database.

Subqueries are an alternate way of returning data from multiple tables.

Subqueries can be used with the following SQL statements along with the comparision operators like =, <, >, >=, <= etc.

- SELECT
- INSERT
- UPDATE
- DELETE

SQL Subquery Example:
1) Usually, a subquery should return only one record, but sometimes it can also return multiple records when used with operators LIKE IN, NOT IN in the where clause. The query syntax would be like,

```
SELECT first_name, last_name, subject
FROM student_details
WHERE games NOT IN ('Cricket', 'Football');
```

Subquery output would be similar to:
2) Lets consider the student_details table which we have used earlier. If you know the name of the students who are studying science subject, you can get their id's by using this query below,

```
SELECT id, first_name
FROM student_details
WHERE first_name IN ('Rahul', 'Stephen');
```

but, if you do not know their names, then to get their id's you need to write the query in this manner,

```
SELECT id, first_name
FROM student_details
WHERE first_name IN (SELECT first_name
FROM student_details
WHERE subject= 'Science');
```

Subquery Output:

In the above sql statement, first the inner query is processed first and then the outer query is processed.

SQL Subquery; INSERT Statement

3) Subquery can be used with INSERT statement to add rows of data from one or more tables to another table. Lets try to group all the students who study Maths in a table 'maths_group'.

```
INSERT INTO maths_group(id, name)
SELECT id, first_name || ' ' || last_name
FROM student_details WHERE subject= 'Maths'
```

SQL Subquery; SELECT Statement

4) A subquery can be used in the SELECT statement as follows. Lets use the product and order_items table defined in the sql_joins section.

```
select p.product name, p.supplier name, (select order id from order items where product id = 101) as
order_id from product p where p.product_id = 101
```

Correlated Subquery

A query is called correlated subquery when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

```
SELECT p.product_name FROM product p
WHERE p.product_id = (SELECT o.product_id FROM order_items o
WHERE o.product_id = p.product_id);
```

Subquery Notes

Nested Subquery

1) You can nest as many queries you want but it is recommended not to nest more than 16 subqueries in oracle

Non-Corelated Subquery

2) If a subquery is not dependent on the outer query it is called a non-correlated subquery

Subquery Errors

3) Minimize subquery errors: Use drag and drop, copy and paste to avoid running subqueries with spelling and database typos. Watch your multiple field SELECT comma use, extra or to few getting SQL error message "Incorrect syntax".

SQL Subquery Comments

Adding SQL Subquery comments are good habit (/* your command comment */) which can save you time, clarify your previous work .. results in less SQL headaches

Nested Queries and Performance Issues in SQL

Nested Queries are queries that contain another complete SELECT statements nested within it, that is, in the WHERE clause. The nested SELECT statement is called an “inner query” or an “inner SELECT.” The main query is called “outer SELECT” or “outer query.” Many nested queries are equivalent to a simple query using JOIN operation. The use of nested query in this case is to avoid explicit coding of JOIN which is a very expensive database operation and to improve query performance. However, in many cases, the use of nested queries is necessary and cannot be replaced by a JOIN operation.

I. Nested queries that can be expressed using JOIN operations:

Example 1: (Library DB Query A) How many copies of the book titled the lost tribe are owned by the library branch whose name is “Sharptown”?

Single Block Query Using Join:

```
SELECT    No_Of_Copies
          FROM      BOOK_COPIES, BOOK, LIBRARY_BRANCH
          WHERE    BOOK_COPIES.BranchId = LIBRARY_BRANCH.BranchId    AND
                    BOOK_COPIES.BookId = BOOK.BookId              AND
                    BOOK.Title = "The Lost Tribe"     AND
                    LIBRARY_BRANCH.BranchName = "Sharpstown";
```

Using Nested Queries:

```
   SELECT    No_Of_Copies
   FROM          BOOK_COPIES
     WHERE             BranchID    IN
                        (SELECT    BranchID    from LIBRARY_BRANCH      WHERE
                                   LIBRARY_BRANCH.BranchName = "Sharpstown")
       AND                      BookID IN
                                                    (SELECT    BookID from    BOOK    WHERE
                                             BOOK.Title = "The Lost Tribe" ) ;
```

Performance considerations: The nested queries in this example involves simpler and faster operations. Each subquery will be executed once and then a simple select operation will be performed. On the other hands, the operations using join require Cartesian products of three tables and have to evaluate 2 join conditions and 2 selection conditions. Nested queries in this example also save internal temporary memory space for holding Cartesian join results.

===

Rule of thumb:

1) ***Correlated queries*** *where the inner query references some attribute of a relation declared in the outer query and use the" =" or IN operators.*
2) *Conversely, if the attributes in the projection operation of a single block query that joins several tables are from only one table, this query can always be translated into a nested query.*

===

Example 2: see Query 12 and Query 12A

Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

Single Block query using JOIN operation

```
select A.fname, A.lname
from employee A, dependent B
where A.ssn = B.essn    and
                  A.sex = B.sex and A.fname = B.dependent_name;
```

Correlated Query:

```
select    A.fname, A.lname
from    employee A
where A.ssn    IN (SELECT essn
                                             FROM dependent
                                             WHERE    essn = A.ssn    and    dependent_name = A.fname and sex = A.sex);
```

Computer Procedures:

Conceptually, think of this query as stepping through EMPLOYEE table one row at a time, and then executing the inner query each time. The first row has A.fname = "John" and A.sex = "M" so that the inner query becomes **SELECT Essn FROM dependent where essn = 12345678, dependent_name = "John" and sex = "M"**; The first run of the subquery returns nothing so it continues to proceed to the next tuple and executes the inner query again with the values of A.SSN, A.fname and A.sex for the second row, and so on for all rows of EMPLOYEE.

The term *correlated subquery* is used because its value depends on a variable (or variables) that receives its value from an outer query (e.g., A.SSN, A.fname, A.sex in this example; they are called **correlation variables**.). A correlated subquery thus cannot be evaluated once and for all. It must be evaluated repeatedly -- once for each value of the variable received from the outer query. This is different from non-correlated subqueries explained below.

Non-correlated Subquery:

A non-correlated subquery needs to be evaluated only once. For example:

Query EMP-NQ2: find an employee that has the highest salary of the company.

SELECT fname, lname, bdate
FROM EMPLOYEE
WHERE salary = **(SELECT max (salary) FROM Employee)**;

Here inner query returns a value: 55000. The inner query will be executed first and only *once* and then the entire query becomes
SELECT fname, lname, bdate
FROM EMPLOYEE WHERE **salary = 55000**;

II. Nested Queries that cannot be directly translated into Join Operations

Rule of thumb:

1) Unknown selection criteria: WHERE clause examines unknown value.

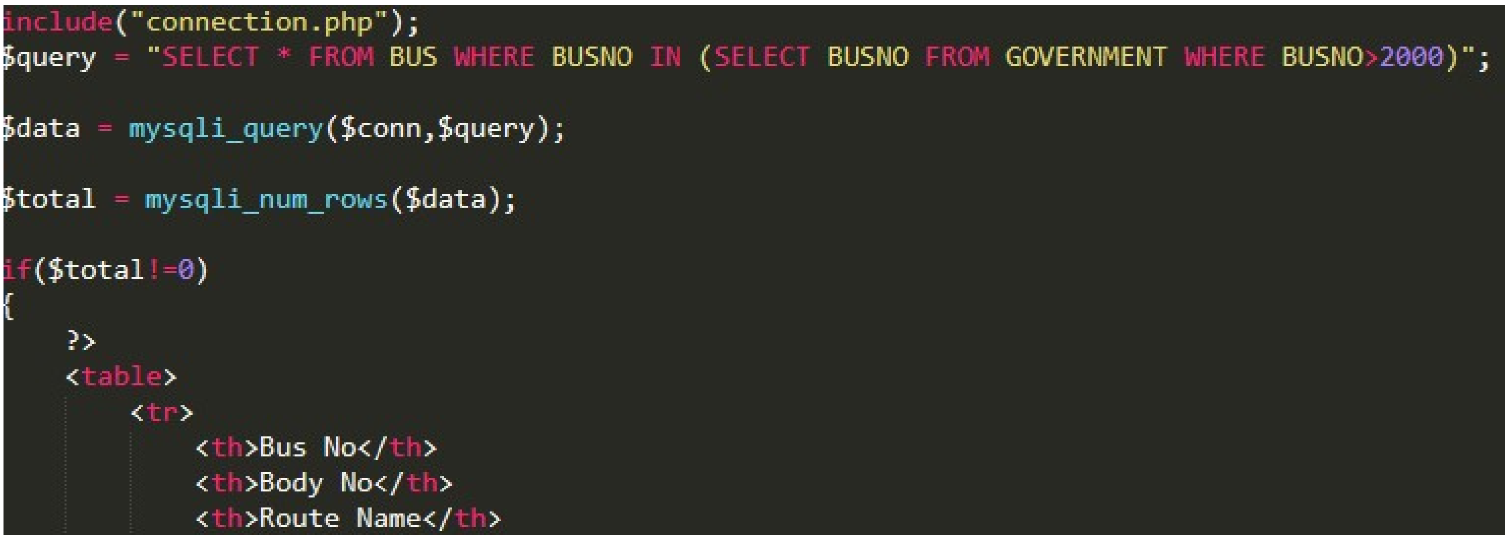
For example shown above (Query EMP-NQ2): find everybody in a department which has an employee that has the highest salary of the company.

Another example in section 7.2.5. finds employees who has salary higher than the highest salary in Department 5.

SELECT ssn, salary, dno from Employee where salary > (SELECT max (salary) from employee where dno = 5);

2) Relational **set** operations such as Division or other comparison that involves EXISTS, NOT EXISTS, > , etc. (This may involve using paradox SET operation operators, such as NO, ONLY, EXACTLY and EVERY.)

3) Outer Join that involves Null value operations. This is the equivalent of using NOT EXISTS. (See *SQL solution for queries on Library DB*: query C and C').


```
include("connection.php");
$query = "SELECT * FROM BUS WHERE BUSNO IN (SELECT BUSNO FROM GOVERNMENT WHERE BUSNO>2000)";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);

if($total!=0)
{
    ?>
    <table>
        <tr>
            <th>Bus No</th>
            <th>Body No</th>
            <th>Route Name</th>
```

SELECT * FROM BUS WHERE BUSNO IN (SELECT BUSNO FROM GOVERNMENT WHERE BUSNO>2000)

SET OPERATIONS

SET OPERATIONS and VIEWS

The Set operator combines the result of 2 queries into a single result.The following are the operators:

· **Union**
· **Union all**
· **Intersect**
· **Minus**

Rules:
· ***The queries which are related by the set operators should have a same number of column and column definition.***
· ***Such query should not contain a type of long.***
· ***Labels under which the result is displayed are those from the first select statement.***

Union:

Returns all distinct rows selected by both the queries
Syntax:
Query1 Union Query2;

Exp: SELECT * FROM table1 UNION SELECT * FROM table2;

Union all:

Returns all rows selected by either query including the duplicates.
Syntax:
Query1 Union all Query2;

Exp: SELECT * FROM table1 UNION ALL SELECT * FROM table2;

Intersect

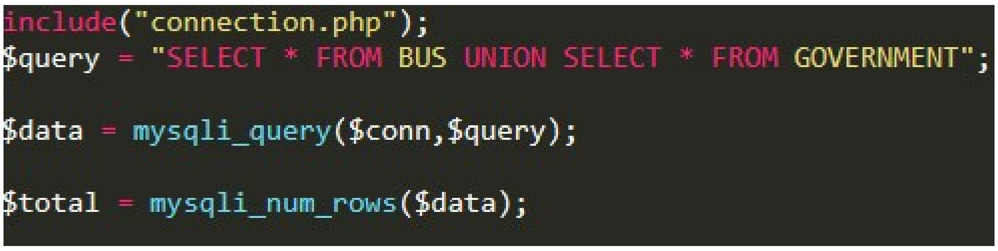
Returns rows selected that are common to both queries.
Syntax:
Query1 Intersect Query2;

Exp: SELECT * FROM table1 INTERSECT SELECT * FROM table2;

Minus

Returns all distinct rows selected by the first query and are not by the second
Syntax:
Query1 minus Query2;

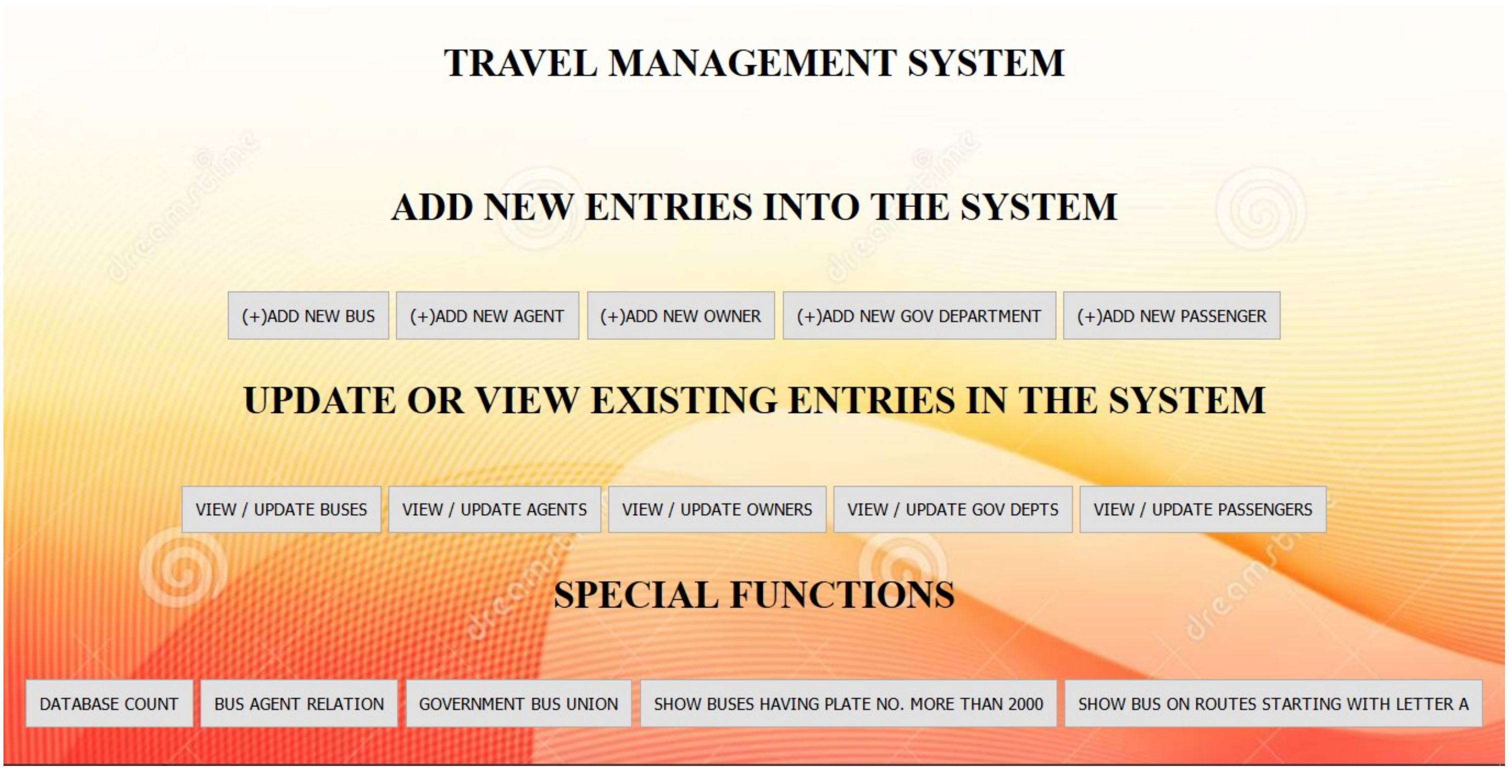
Exp: SELECT * FROM table1 MINUS SELECT * FROM table2;



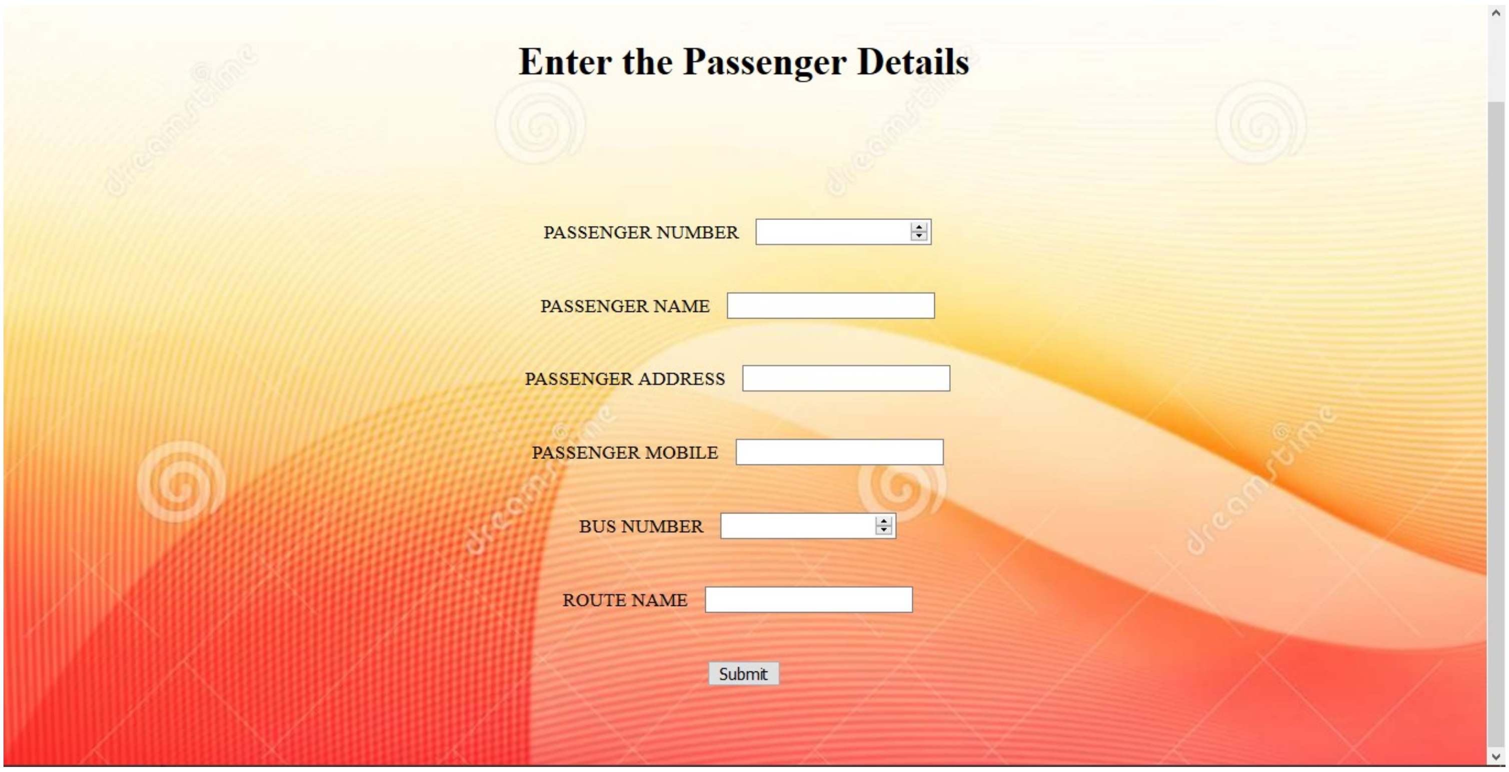
```php
include("connection.php");
$query = "SELECT * FROM BUS UNION SELECT * FROM GOVERNMENT";

$data = mysqli_query($conn,$query);

$total = mysqli_num_rows($data);
```

SELECT * FROM BUS UNION SELECT * FROM GOVERNMENT

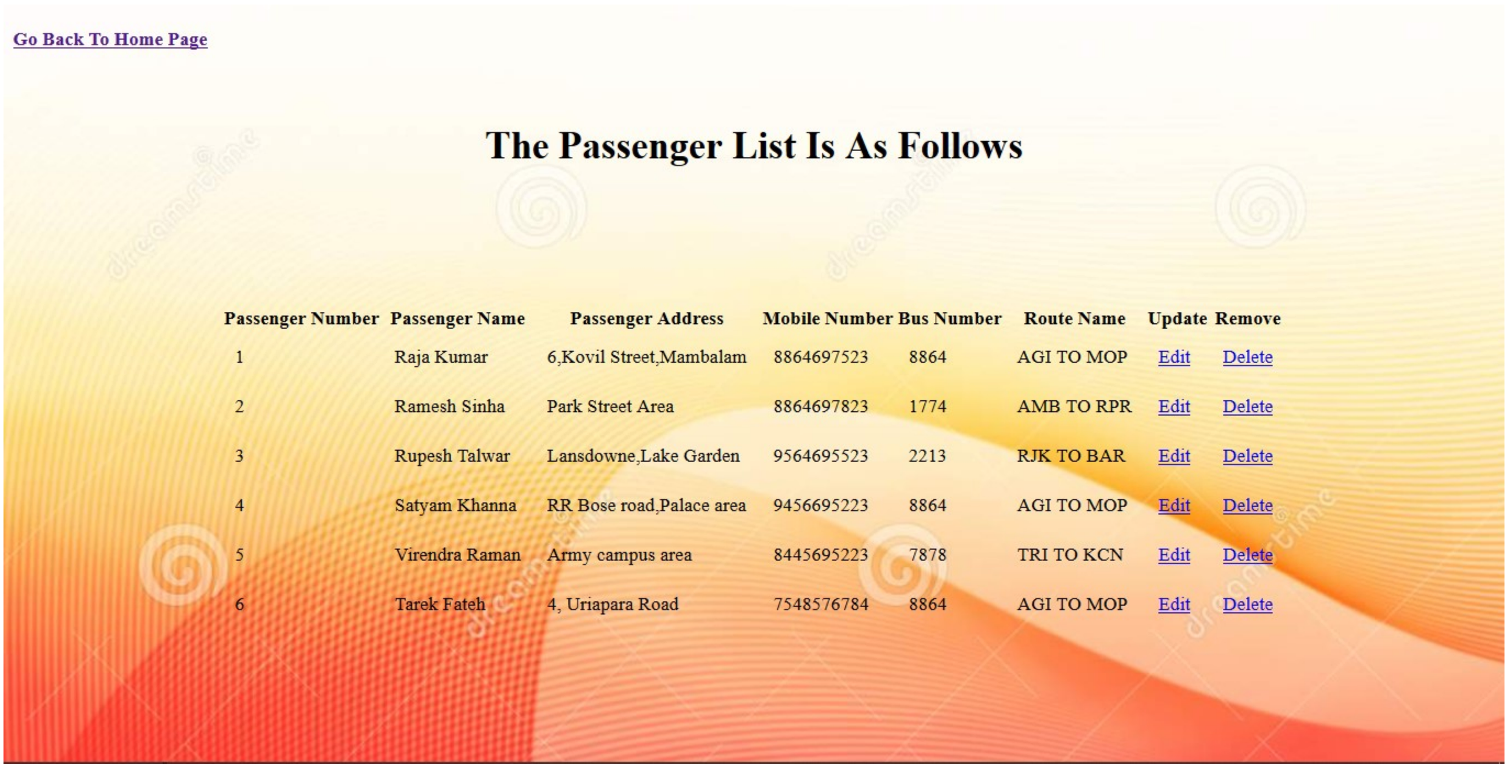
ABOUT GUI



HOME PAGE



INSERT PAGE

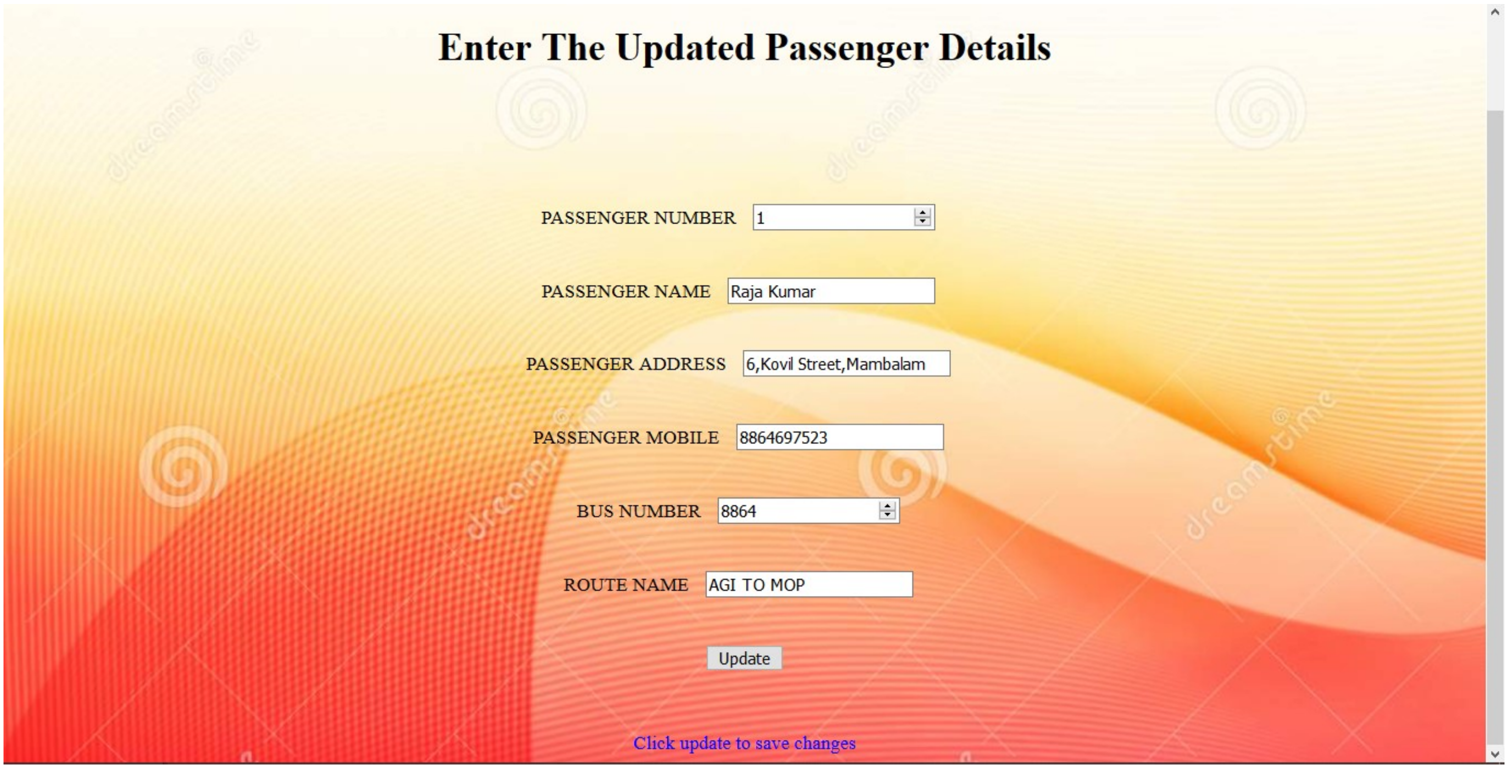


Go Back To Home Page

The Passenger List Is As Follows

Passenger Number	Passenger Name	Passenger Address	Mobile Number	Bus Number	Route Name	Update	Remove
1	Raja Kumar	6,Kovil Street,Mambalam	8864697523	8864	AGI TO MOP	Edit	Delete
2	Ramesh Sinha	Park Street Area	8864697823	1774	AMB TO RPR	Edit	Delete
3	Rupesh Talwar	Lansdowne,Lake Garden	9564695523	2213	RJK TO BAR	Edit	Delete
4	Satyam Khanna	RR Bose road,Palace area	9456695223	8864	AGI TO MOP	Edit	Delete
5	Virendra Raman	Army campus area	8445695223	7878	TRI TO KCN	Edit	Delete
6	Tarek Fateh	4, Uriapara Road	7548576784	8864	AGI TO MOP	Edit	Delete

VIEW/UPDATE PAGE

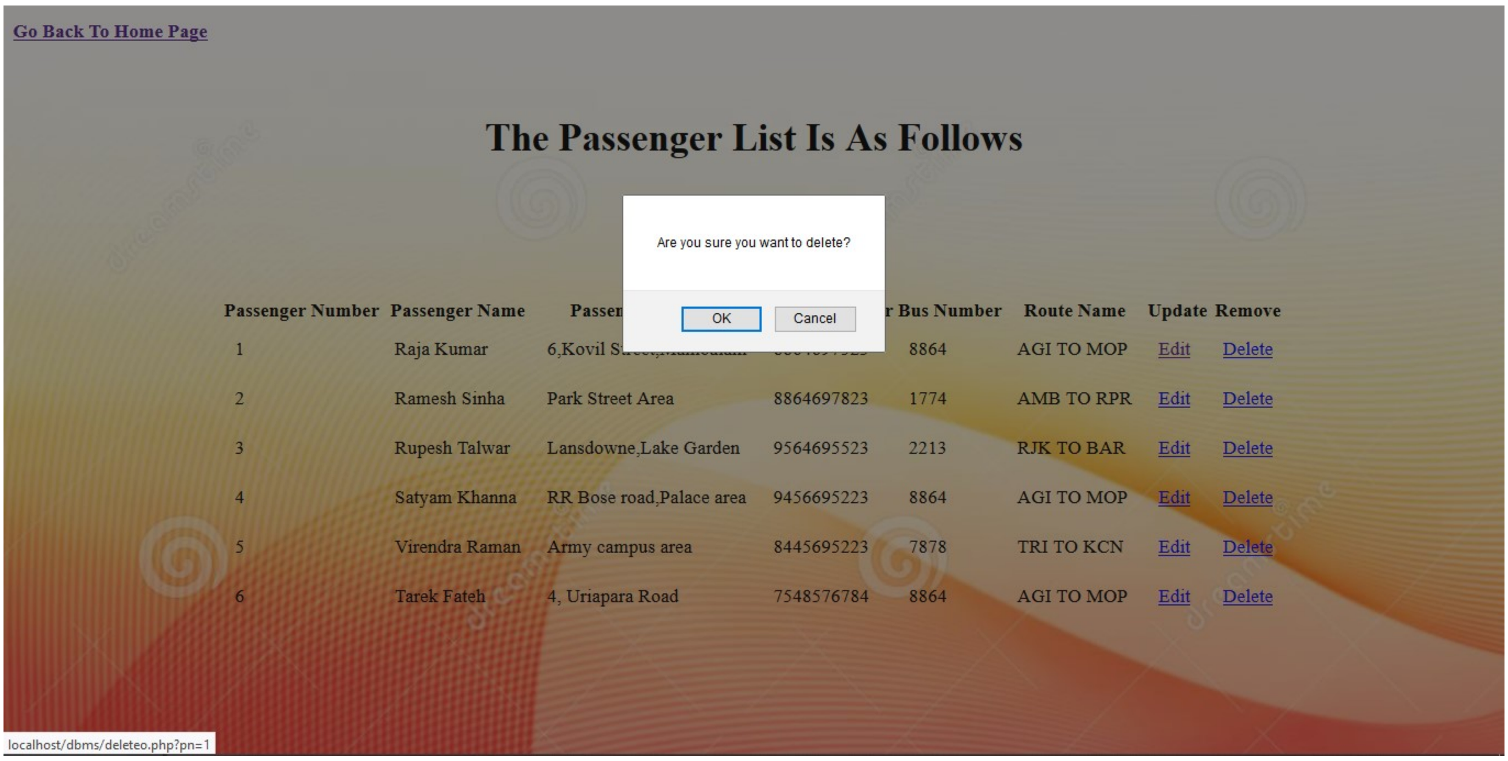


Enter The Updated Passenger Details

PASSENGER NUMBER	1
PASSENGER NAME	Raja Kumar
PASSENGER ADDRESS	6,Kovil Street,Mambalam
PASSENGER MOBILE	8864697523
BUS NUMBER	8864
ROUTE NAME	AGI TO MOP

Click update to save changes

EDIT PAGE



Go Back To Home Page

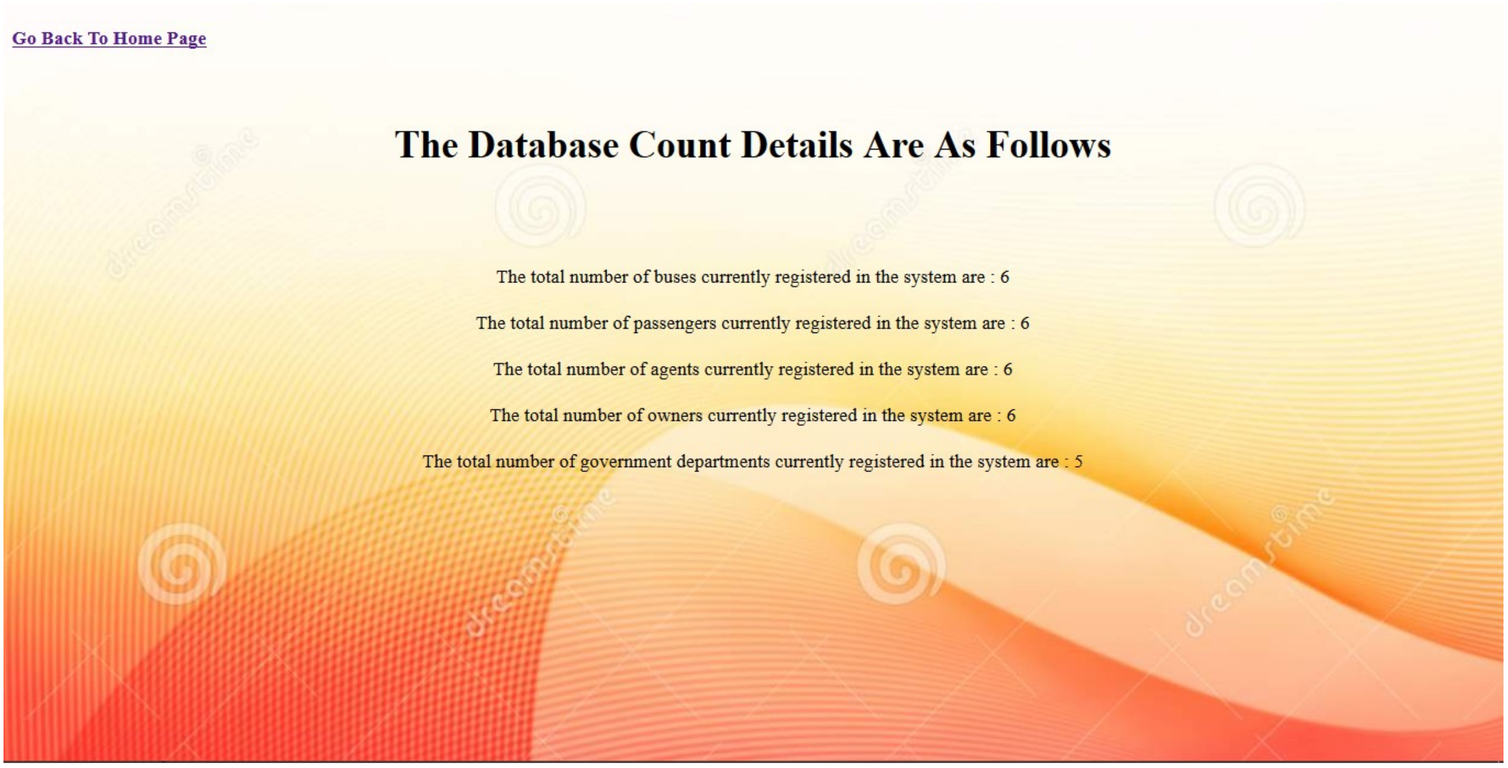
The Passenger List Is As Follows

Are you sure you want to delete?

Passenger Number	Passenger Name	Passen		r Bus Number	Route Name	Update	Remove
1	Raja Kumar	6,Kovil S		8864	AGI TO MOP	Edit	Delete
2	Ramesh Sinha	Park Street Area	8864697823	1774	AMB TO RPR	Edit	Delete
3	Rupesh Talwar	Lansdowne,Lake Garden	9564695523	2213	RJK TO BAR	Edit	Delete
4	Satyam Khanna	RR Bose road,Palace area	9456695223	8864	AGI TO MOP	Edit	Delete
5	Virendra Raman	Army campus area	8445695223	7878	TRI TO KCN	Edit	Delete
6	Tarek Fateh	4, Uriapara Road	7548576784	8864	AGI TO MOP	Edit	Delete

localhost/dbms/deleteo.php?pn=1

DELETE POPUP



Go Back To Home Page

The Database Count Details Are As Follows

The total number of buses currently registered in the system are : 6

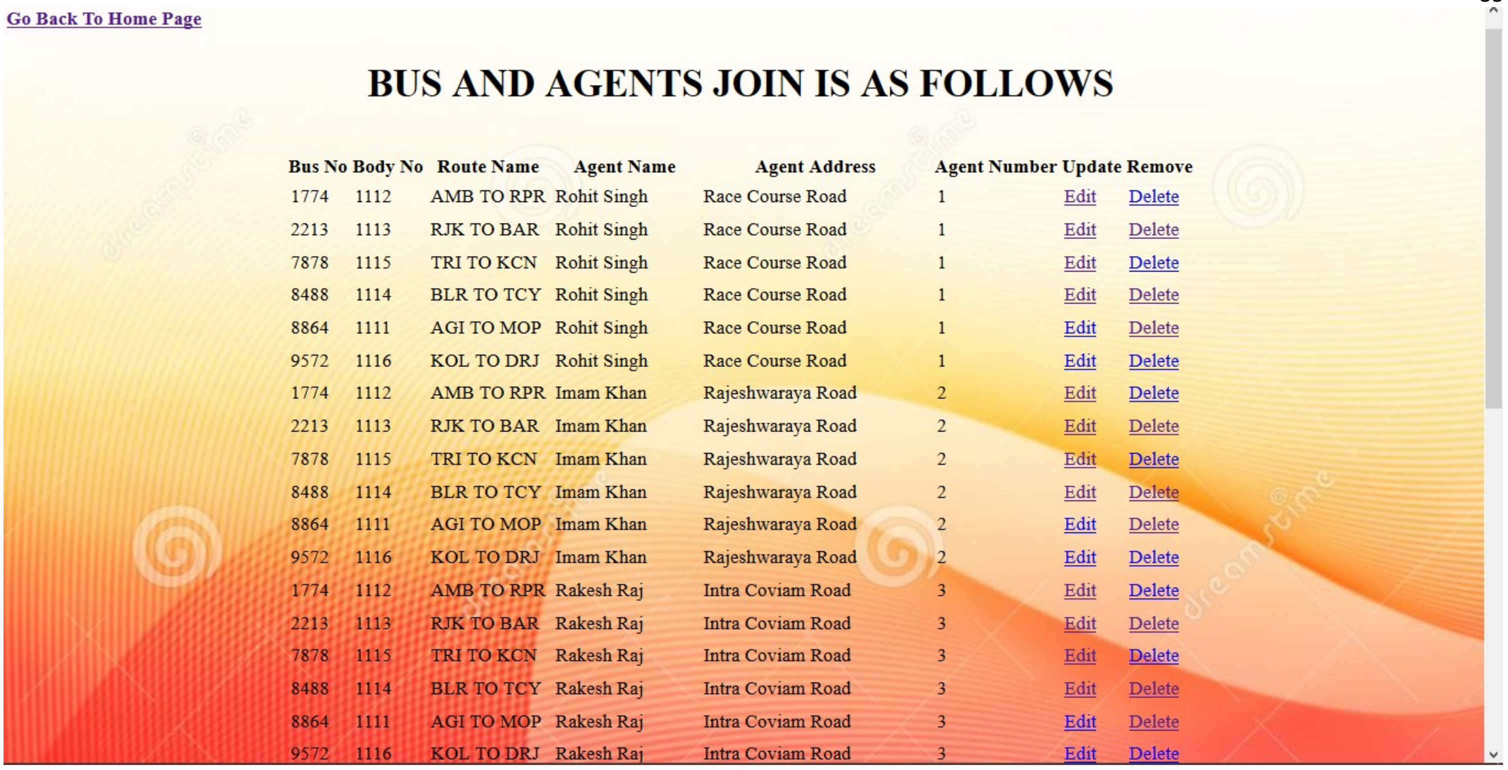
The total number of passengers currently registered in the system are : 6

The total number of agents currently registered in the system are : 6

The total number of owners currently registered in the system are : 6

The total number of government departments currently registered in the system are : 5

DATABASE COUNT

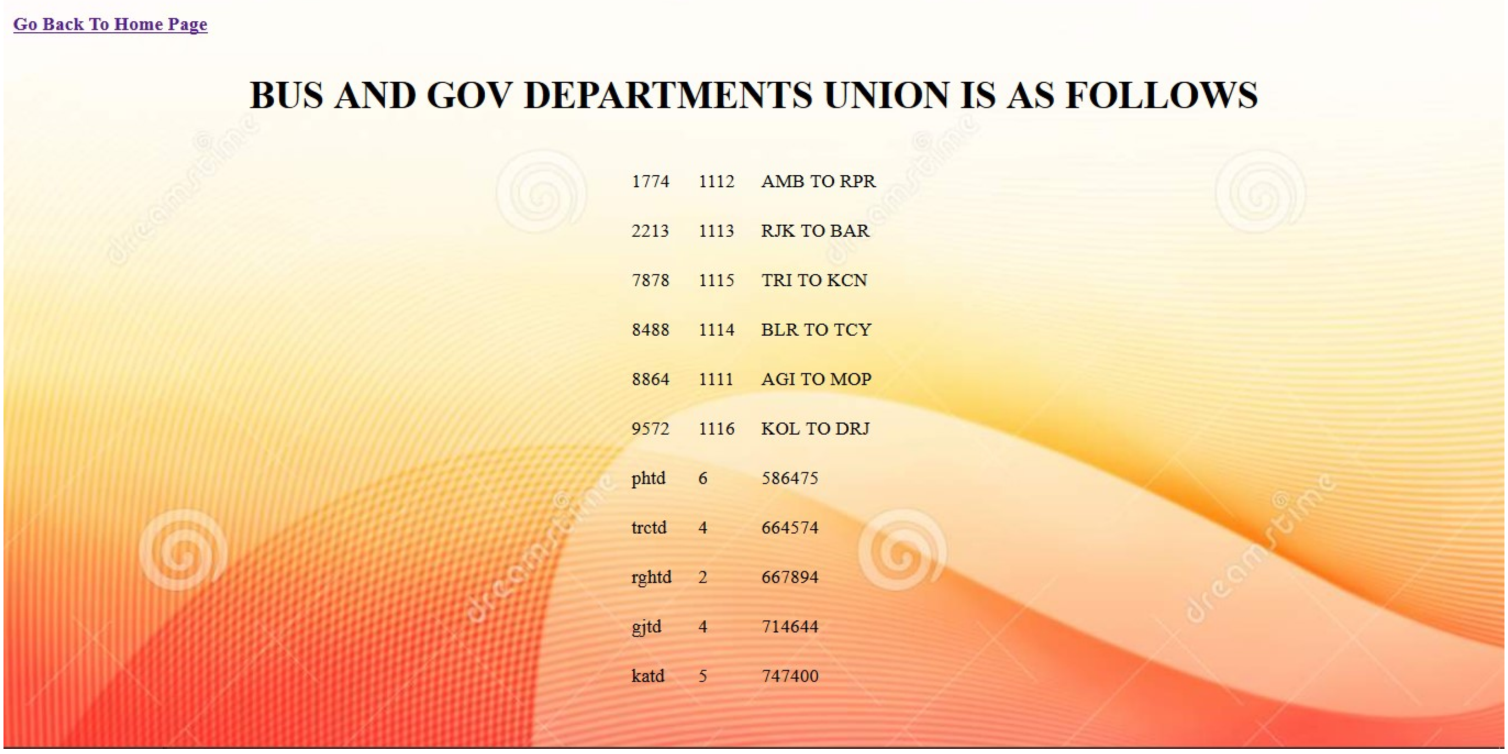


Go Back To Home Page

BUS AND AGENTS JOIN IS AS FOLLOWS

Bus No	Body No	Route Name	Agent Name	Agent Address	Agent Number	Update	Remove
1774	1112	AMB TO RPR	Rohit Singh	Race Course Road	1	Edit	Delete
2213	1113	RJK TO BAR	Rohit Singh	Race Course Road	1	Edit	Delete
7878	1115	TRI TO KCN	Rohit Singh	Race Course Road	1	Edit	Delete
8488	1114	BLR TO TCY	Rohit Singh	Race Course Road	1	Edit	Delete
8864	1111	AGI TO MOP	Rohit Singh	Race Course Road	1	Edit	Delete
9572	1116	KOL TO DRJ	Rohit Singh	Race Course Road	1	Edit	Delete
1774	1112	AMB TO RPR	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
2213	1113	RJK TO BAR	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
7878	1115	TRI TO KCN	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
8488	1114	BLR TO TCY	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
8864	1111	AGI TO MOP	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
9572	1116	KOL TO DRJ	Imam Khan	Rajeshwaraya Road	2	Edit	Delete
1774	1112	AMB TO RPR	Rakesh Raj	Intra Coviam Road	3	Edit	Delete
2213	1113	RJK TO BAR	Rakesh Raj	Intra Coviam Road	3	Edit	Delete
7878	1115	TRI TO KCN	Rakesh Raj	Intra Coviam Road	3	Edit	Delete
8488	1114	BLR TO TCY	Rakesh Raj	Intra Coviam Road	3	Edit	Delete
8864	1111	AGI TO MOP	Rakesh Raj	Intra Coviam Road	3	Edit	Delete
9572	1116	KOL TO DRJ	Rakesh Raj	Intra Coviam Road	3	Edit	Delete

JOINS PAGE

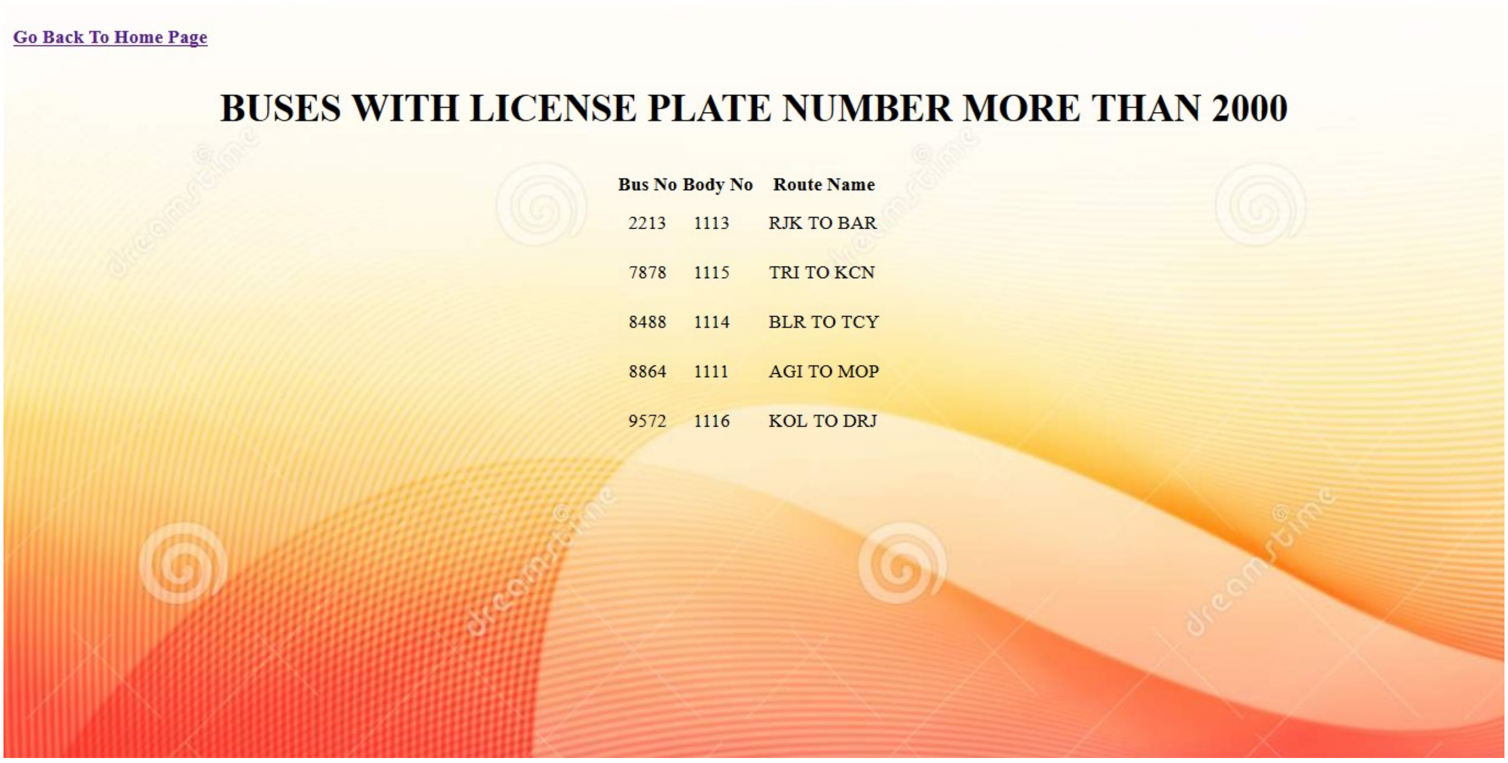


Go Back To Home Page

BUS AND GOV DEPARTMENTS UNION IS AS FOLLOWS

1774	1112	AMB TO RPR
2213	1113	RJK TO BAR
7878	1115	TRI TO KCN
8488	1114	BLR TO TCY
8864	1111	AGI TO MOP
9572	1116	KOL TO DRJ
phtd	6	586475
trctd	4	664574
rghtd	2	667894
gjtd	4	714644
katd	5	747400

UNION PAGE

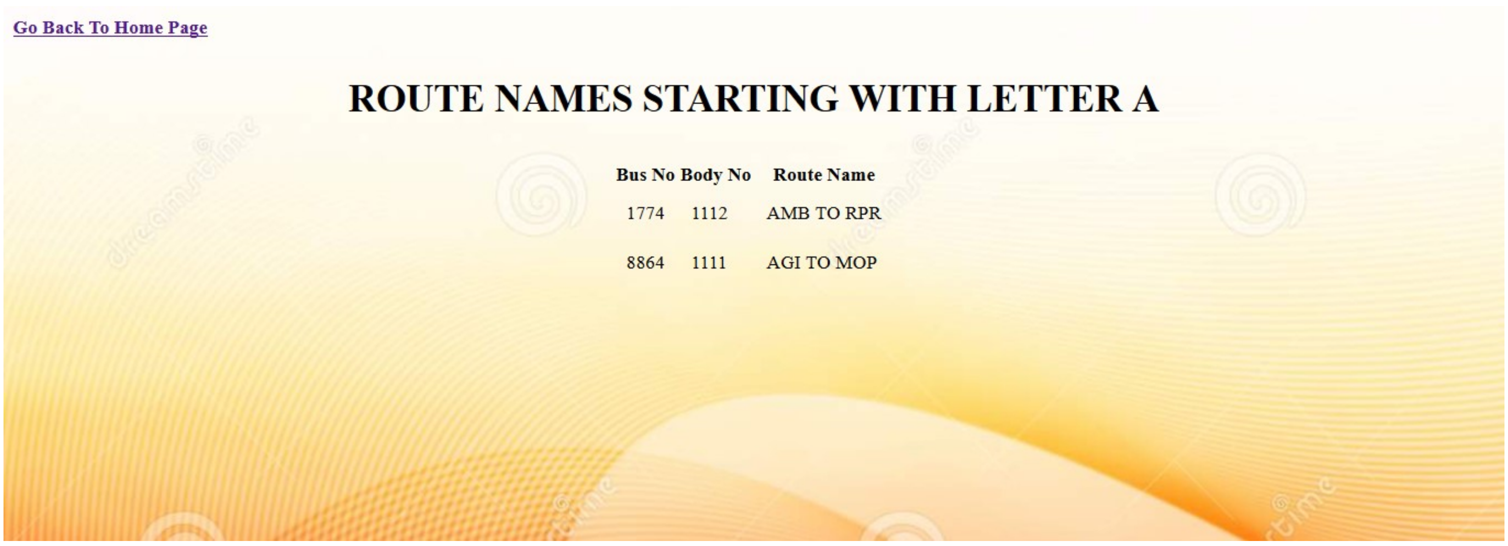


Go Back To Home Page

BUSES WITH LICENSE PLATE NUMBER MORE THAN 2000

Bus No	Body No	Route Name
2213	1113	RJK TO BAR
7878	1115	TRI TO KCN
8488	1114	BLR TO TCY
8864	1111	AGI TO MOP
9572	1116	KOL TO DRJ

SUBQUERY PAGE



Go Back To Home Page

ROUTE NAMES STARTING WITH LETTER A

Bus No	Body No	Route Name
1774	1112	AMB TO RPR
8864	1111	AGI TO MOP

INBUILT FUNCTION PAGE

PL/SQL CONDITIONAL ITERATIVE STATEMENTS

```
DECLARE
sales NUMBER(8,2) := 10100;
quota NUMBER(8,2) := 10000;
bonus NUMBER(6,2);
emp_id NUMBER(6) := 120;
BEGIN
IF sales > (quota + 200) THEN
bonus := (sales - quota)/4;
UPDATE employees SET salary = salary + bonus WHERE employee_id = emp_id;
END IF;
END;
/
```

```
DECLARE
grade CHAR(1);
BEGIN
grade := 'B';
CASE grade
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
END;
/
```

```
DECLARE
p NUMBER := 0;
BEGIN
FOR k IN 1..500 LOOP -- calculate pi with 500 terms
p := p + ( ( (-1) ** (k + 1) ) / ((2 * k) - 1) );
END LOOP;
p := 4 * p;
DBMS_OUTPUT.PUT_LINE( 'pi is approximately : ' || p ); -- print result
END;
/
```

```
DECLARE
p VARCHAR2(30);
n PLS_INTEGER := 37; -- test any integer > 2 for prime
BEGIN
FOR j in 2..ROUND(SQRT(n)) LOOP
IF n MOD j = 0 THEN -- test for prime
p := ' is not a prime number'; -- not a prime number
GOTO print_now;
END IF;
END LOOP;
p := ' is a prime number';
<<print_now>>
DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

PL/SQL PROCEDURES AND FUNCTIONS

```
DECLARE
    a number;
    b number;
    c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
    IF x < y THEN
        z:= x;
    ELSE
        z:= y;
    END IF;
END;
BEGIN
    a:= 23;
    b:= 45;
    findMin(a, b, c);
    dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
/

DECLARE
    a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
    a:= 23;
    squareNum(a);
    dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
    total number(2) := 0;
BEGIN
    SELECT count(*) into total
    FROM customers;

    RETURN total;
END;
/
```

PL/SQL CURSORS

```
DECLARE
    total_rows number(2);
BEGIN
    UPDATE customers
    SET salary = salary + 500;
    IF sql%notfound THEN
        dbms_output.put_line('no customers selected');
    ELSIF sql%found THEN
        total_rows := sql%rowcount;
        dbms_output.put_line( total_rows || ' customers selected ');
    END IF;
END;
/
```

```
DECLARE
    c_id customers.id%type;
    c_name customerS.No.ame%type;
    c_addr customers.address%type;
    CURSOR c_customers is
        SELECT id, name, address FROM customers;
BEGIN
    OPEN c_customers;
    LOOP
    FETCH c_customers into c_id, c_name, c_addr;
        EXIT WHEN c_customers%notfound;
        dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
    END LOOP;
    CLOSE c_customers;
END;
/
```