

# Transaction Processing Performance Tool v.1.4

## Help Contents

[What is Transaction Processing Performance Tool](#)

[Command and Control](#) updated!

[How to start an experiment](#) updated!

[How to set Session Keywords or Terminal Keywords \(For TA use\)](#)

[How to edit Sybase server URL \(For TA use\)](#) New!

[How to set transaction isolation levels](#)

[How to set max rows per page](#)

[How to query the database](#) New!

[Important tips](#) New!

[Performance Measurement Report](#) updated!

[Database query Report](#) New!

[User Input File Requirements](#)

[History of TPPT](#) New!

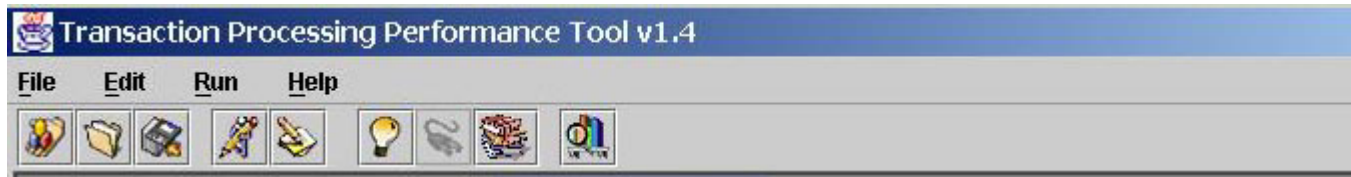
---

## [What is Transaction Processing Performance Tool?](#)

Transaction Processing Performance Tool is a facility designed to measure the performance of a database application built on Sybase. Through examining the effects of indices, locking disciplines, and deadlocks, this tool helps the user discover the bottlenecks of a database design and improve the overall user application for better performance.

## Command and Control

The control bar of Transaction Processing Performance Tool consists of two components, a menu bar and a tool bar. Each component controls various operations of the system:



### • **Menu Bar**

- File Menu

New Experiment:	Opens an <a href="#">Experiment</a> window and allows user to set up an experiment.
Open Experiment:	Opens an Experiment window with an existing experiment setting from an .exp file.
Save Experiment Setting:	Saves the current experiment setting in an .exp file.
Exit:	Closes the entire Performance Tool.

- Edit Menu

Set Isolation Levels:	Opens a <a href="#">Set Transaction Isolation Level</a> window and allows user to set isolation level for each transaction.
Set Max_Rows_Per_Page:	Opens a <a href="#">Set Max Rows Per Page</a> window and allows user to set a single max_rows_per_page value for all user defined tables.

- Run Menu

Start Experiment:	Starts executing an experiment.
Stop Experiment:	Stops experiment from execution.

- Help Menu

Provides information and explanation on how to use this Performance Tool.

## • Tool Bar



Opens an [Experiment](#) window and allows user to set up an experiment. Same as New Experiment from File menu.



Opens an Experiment window with an existing experiment setting from an .exp file. Same as Open Experiment from File menu.



Saves the current experiment setting in an .exp file. Same as Save Experiment Setting from File menu.



Opens a [Set Transaction Isolation Level](#) window and allows user to set isolation level for each transaction.  
Same as Set Isolation Level from Edit menu.



Opens a [Set Max Rows Per Page](#) window and allows user to set a single max\_rows\_per\_page value for all user defined tables.



Starts executing an experiment. Same as Start Experiment from Run menu.



Stops experiment from execution. Same as Stop Experiment from Run menu.



Runs a user-defined query to Sybase. Same as check Database status from Run menu.



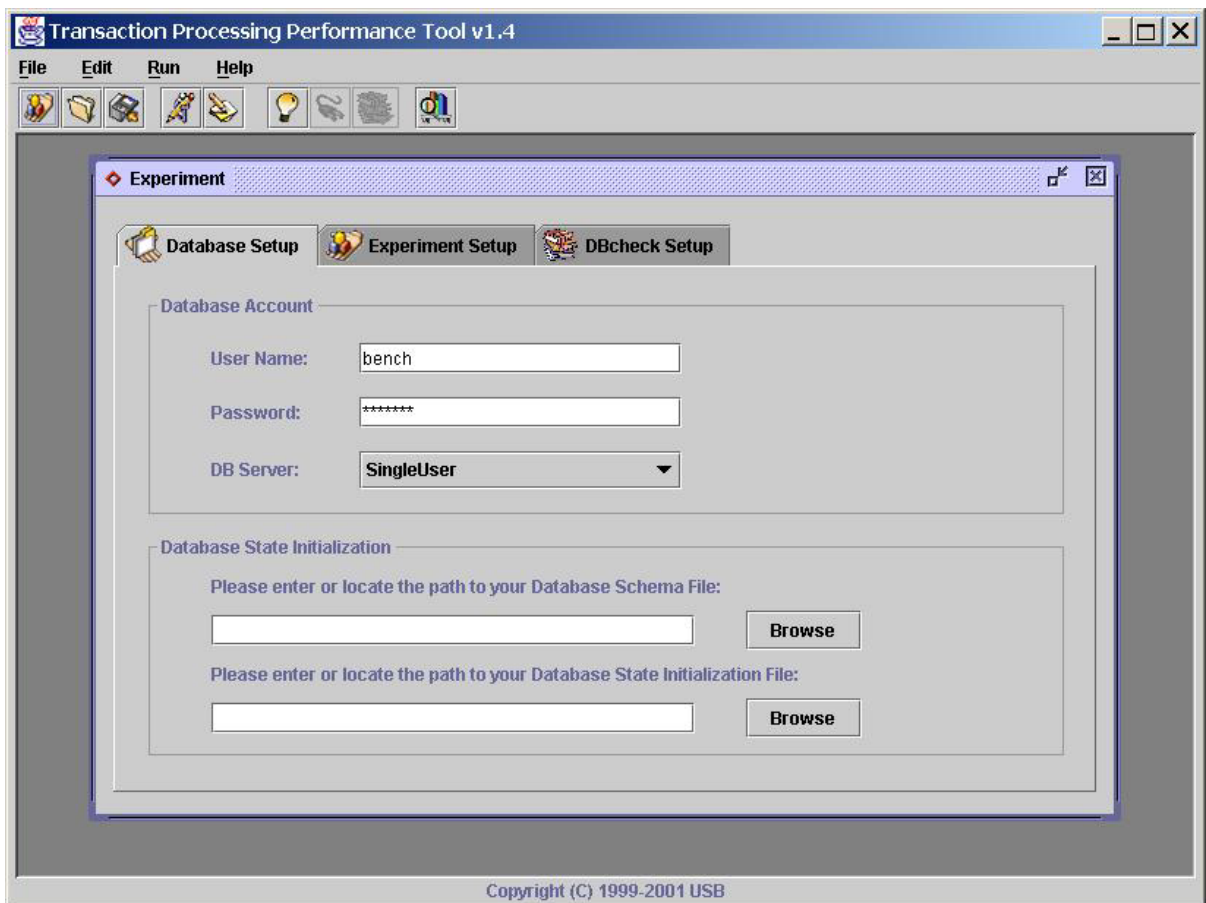
Provides information and explanation on how to use this Performance Tool. Same as How to use Performance Tool from Help menu.

## How to start an experiment?

To start an experiment, you can either create a new experiment, or open an existing experiment setting (suggested).

### • Create a new Experiment

1. Click **File** on the menu bar, then choose **New Experiment**. Or, click the  icon on the tool bar. An **Experiment** Window shall be displayed:



2. Click **DB Setup** to bring the Database Setup panel to the front. ( see the above image )
3. Fill in your database account name and password in the fields named **User Name** and **Password**.
4. Use the drop-down list next to **DB Server** to select a Sybase server that you would like to connect to. The default server is **sbntbench**.
5. Type in or use the **Browse** button to locate the path to your [Database Schema File](#) if it's missing.

6. Type in or use the **Browse** button to locate the path to your [Database Initialization File](#).

7. Click **Experiment Setup** to bring the Experiment Setup panel to the front:



8. In the field named **Number of Terminal**, fill in the total number of terminals that you would like to simulate for the experiment. Your input value must be greater than 0, otherwise an error message will be displayed.

9. In the field called **Think Time**, fill in the number of seconds that you want your experiment to be idle between any two transactions. The input value must be in the range [0, Total execution time].


10. Fill in the total number seconds that you would like to run the experiment in the field named **Execution Time**. The maximum execution time allowed is 60 seconds. Your input value must be in the range of (0, 60], or else an error message will be displayed.

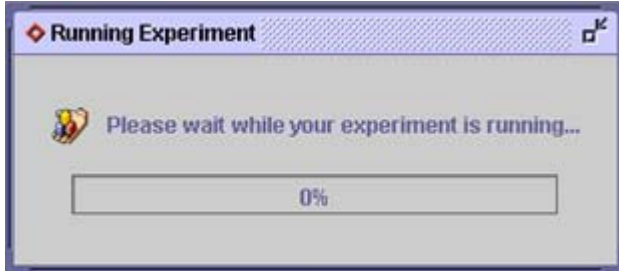
11. Type in or use the **Browse** buttons on the right side to locate the paths to your [Transaction Prototypes File](#), [Transaction Procedure Source File Directory](#), [Terminal Script File Directory](#), and [Transaction Argument File Directory](#).


( NOTE: The Transaction Prototypes File contains the signatures of all transaction procedures. Each file in the Transaction Procedure Source File Directory refers to the pre-compiled source file of one of the transactions listed in Transaction Prototypes File. )


12. [Set isolation levels](#) for transactions. (Optional) The default level for each of your transactions is 1, Read Committed.

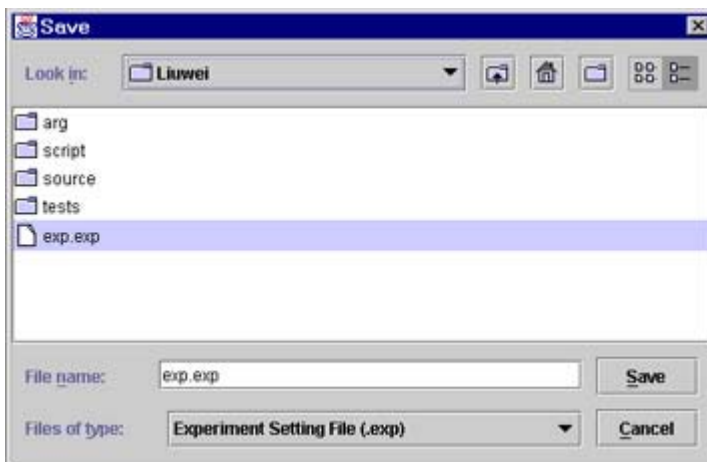
13. [Set max\\_rows\\_per\\_page value](#) for your tables. (Optional) The default max\_rows\_per\_page value is 0, which means Sybase will be responsible for managing the number of rows per page for your database tables.

14. To start running your experiment, click **Run** on the menu bar, then choose **Start Experiment**. Or simply click the  icon on the tool bar. An experiment progress bar will be displayed showing the percentage of the experiment TPPT has accomplished:



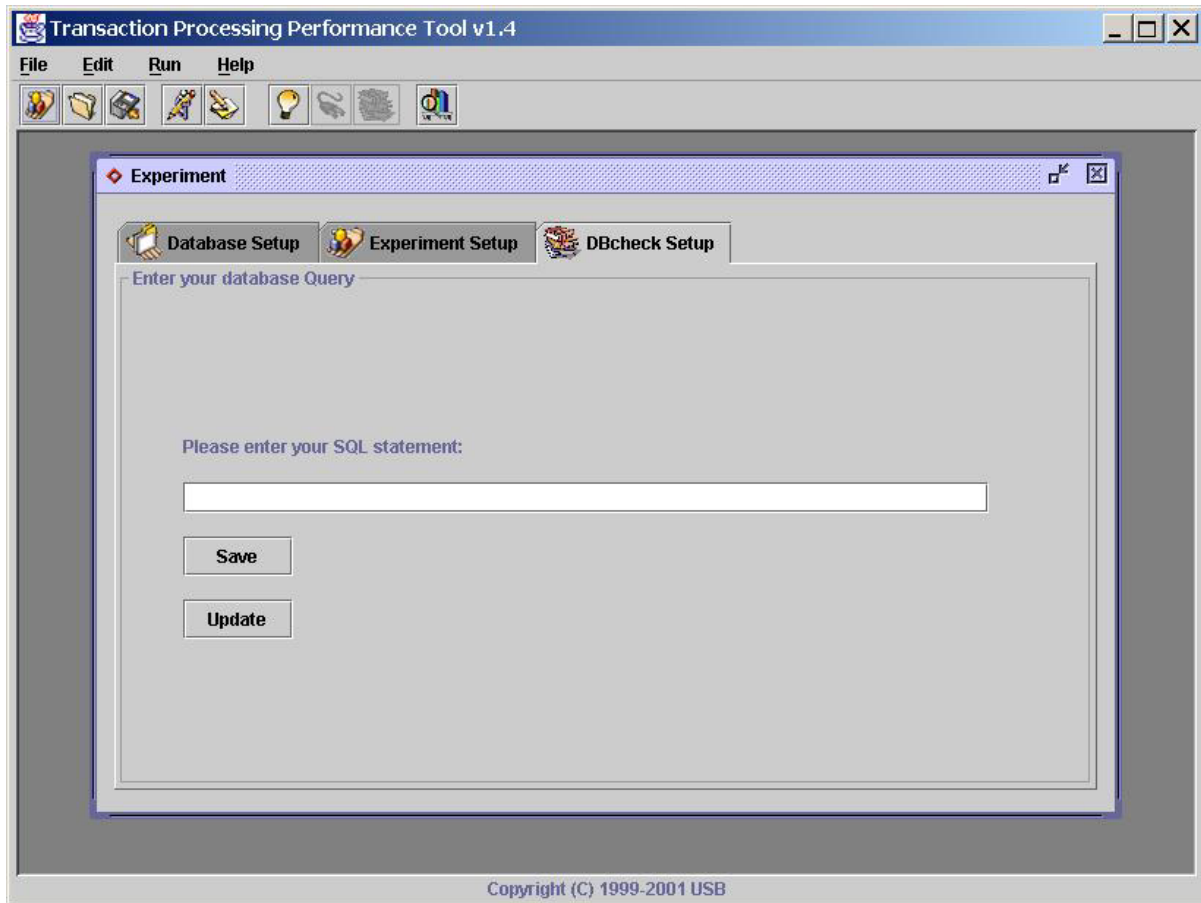
15. If you want to stop the experiment during its execution, click **Run** then choose **Stop Experiment**, or click the  icon. If you would like to start a new experiment, simply repeat the above steps.

16. If you want to save the current experiment setting in an .exp file, click **File** then choose **Save Experiment Setting**, or click the  icon. A file chooser window will appear. Type in the file name for your experiment setting file, for example, *myexperiment.exp*, then click **Save**.




( NOTE: Only the following information will be saved: Database Account User Name, Database Schema File path, Database Initialization File path, Transaction Prototypes File path, Transaction Source Files Directory path, Terminal Script Files Directory path, and Transaction Argument Files Directory path. [New in TPPTv1.4](#), all settings will be saved.)

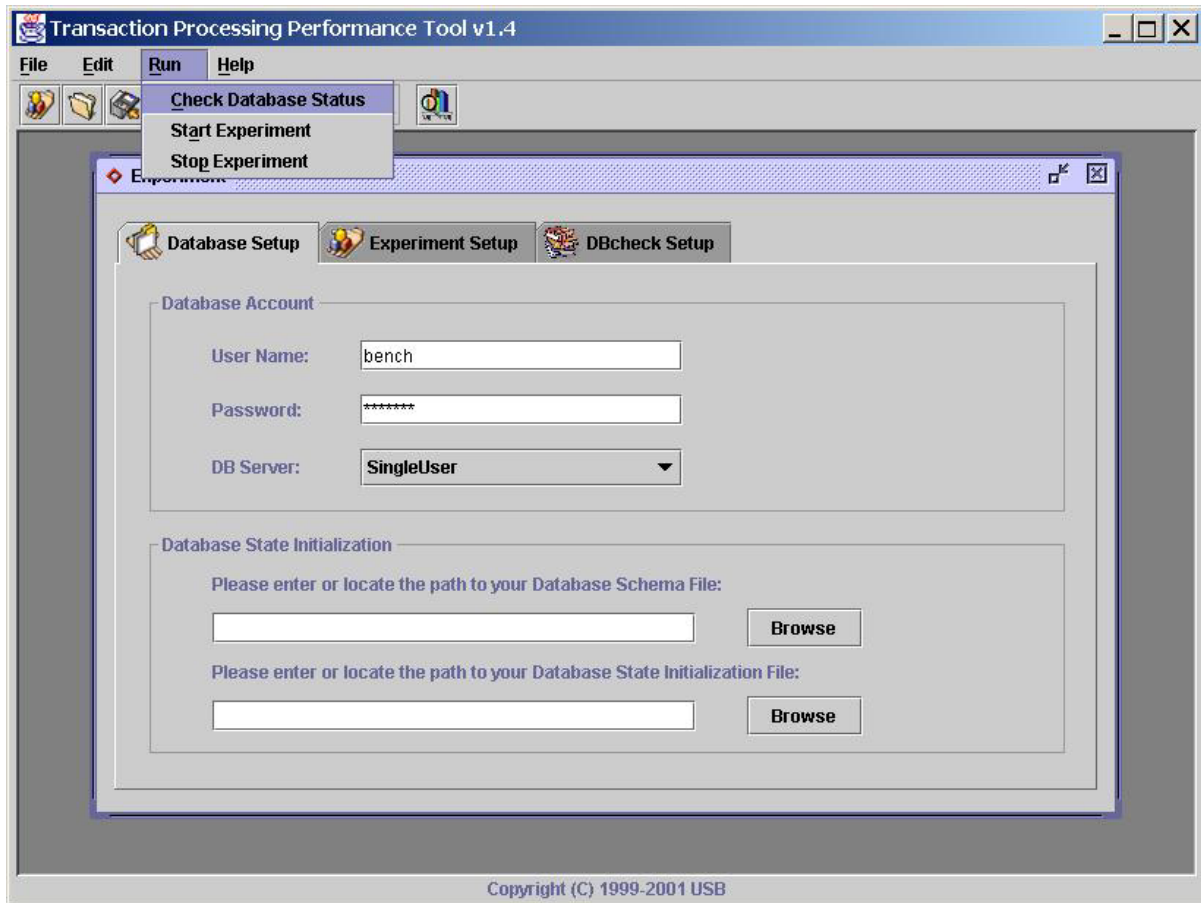
16. If you would like to query the database after one **successful** experiment, you may click on DBCheck Setup tab to bring the frame into foreground.




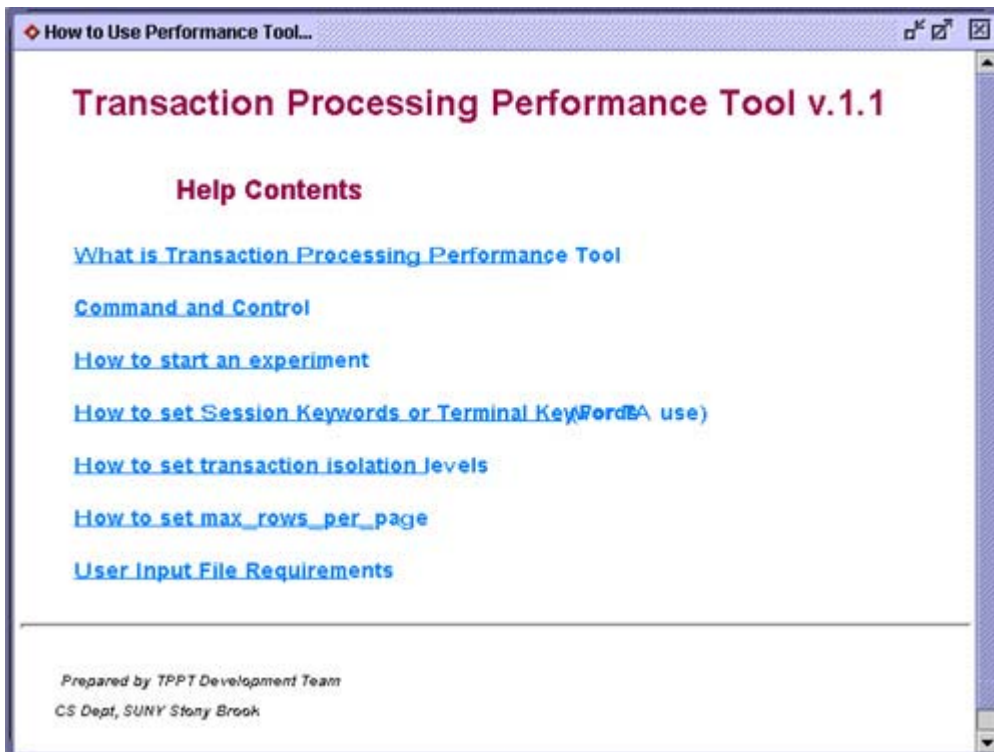
now you have to enter the SQL statement you wish to run (normally, it would be some kind of aggregate function on certain tables to check the general state of database). To inform TPPT that a new query has been entered, you should click on the **"Update"** button. A pop-up window should appear to inform you it's done.



To save the query together with all other current experimental settings, you can click on the **"Save"** button and follow the instructions on the file chooser. To run your query, you may either click on the  icon (which becomes activated after at least one **successful** experiment) or choose "check database status" from the drop-down menu under "Run".

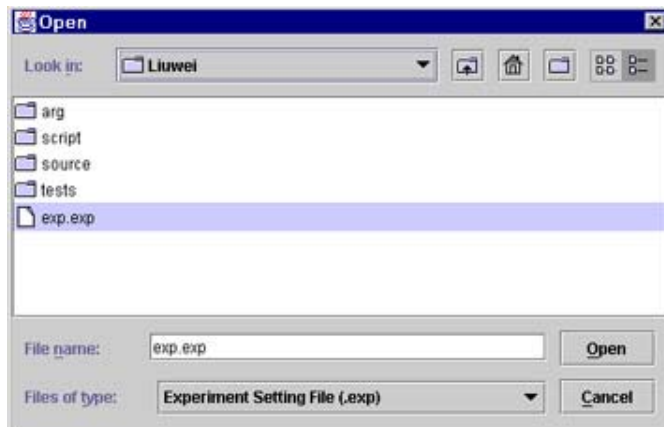


17. If you need help, click **Help** on the menu bar, then choose **How to use Performance Tool**, or simply click the  icon on the tool bar. A help menu browser window will be displayed. To find information on any topic, click on its link.



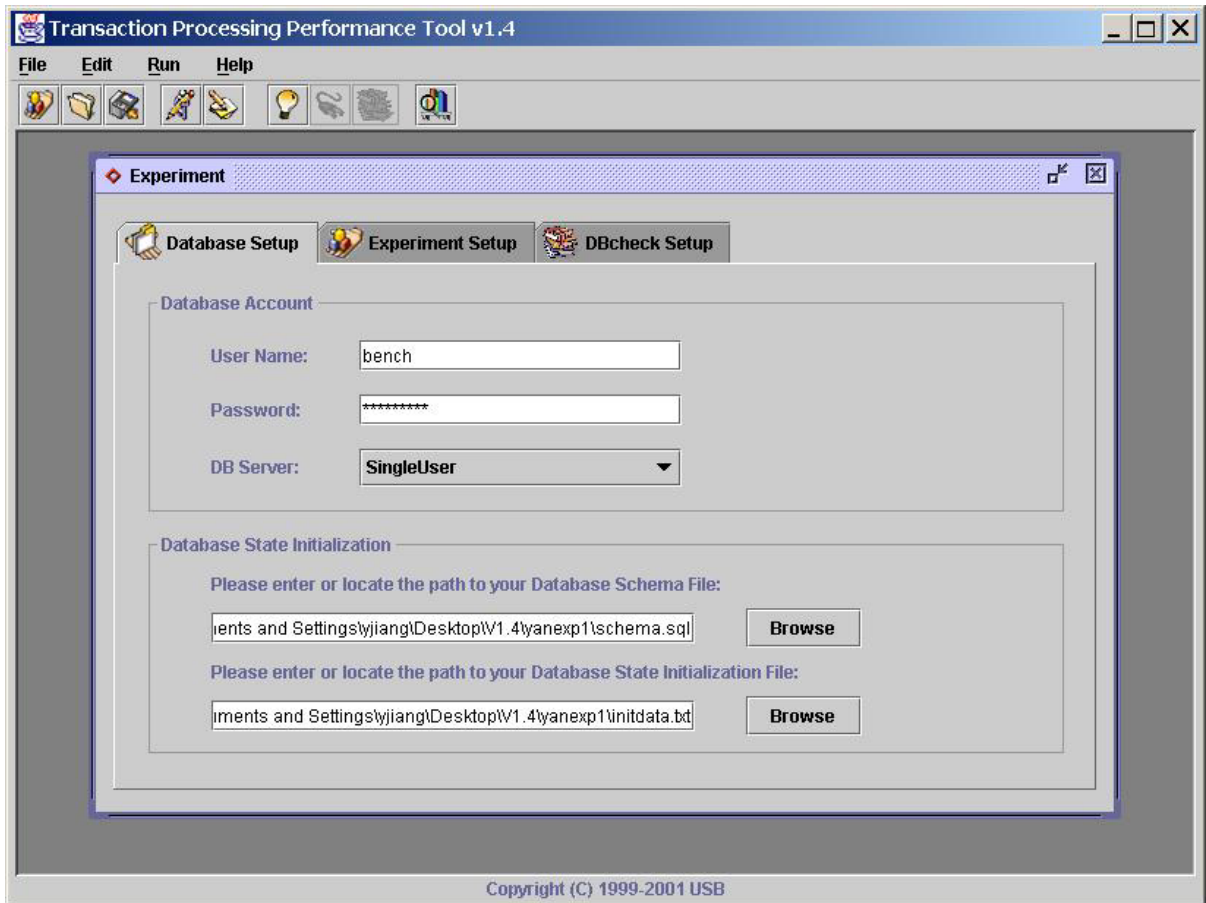
## • Open an Existing Experiment

1. Click **File** on the menu bar, then choose **Open Experiment**. Or, click the  icon on the tool bar. A file chooser window shall be displayed:



2. Select the appropriate experiment file (i.e. experiment1.exp), and click **Open**.

An **Experiment** Window shall be displayed with the experiment setting from the selected file:



**New in TPPTv1.4:** if you don't want to make any modifications to experiment setting, now you can start running performance tests right away by clicking on the "Run Experiment" button or select from the top menu.

---

## [How to set Session Keywords or Terminal Keywords? \( For TA Use\)](#)

Two types of keywords might be involved in an experiment: Session Keywords and Terminal Keywords.

**Session Keywords** are transaction parameters whose value must remain the same within one transaction session on a terminal. Imagine a Student Registration System with 3 terminals running concurrently; in front of each terminal there are several students waiting on line, thus when each of these students gets his turn to execute a number of transactions, the student is conducting one transaction session. During one such transaction session ( for example, between Login and Logout ), the student's user ID and password must represent the same person and their values must remain the same, hence *Student ID* and *Password* may be considered as Session Keywords.

**Terminal Keywords** are transaction parameters whose value must not appear on different terminals at the same moment during the experiment. For example, the registrar of a school who monitors the Student Registration System may disallow any student to logon to more than one terminal at the same time. Thus, in such system, no two terminals may encounter the same *Student ID* and *Student Password* arguments simultaneously. Therefore, *Student ID* and *Password* may be defined as Terminal keywords.

In theory, a transaction parameter could be a Session Keyword, a Terminal Keyword, neither, or both. In the example described above, *Student ID* and *Password* are both Session Keywords and Terminal Keywords. However, if the registrar does allow a student to use more than one terminal at the same time, then these two parameters may be just Session Keywords, but not Terminal Keywords. If for some reason the registrar does not allow students to register for the same course on different terminals at the same time, then the *Course Number* may be a Terminal Keyword, not a Session Keyword, but this is very rare.

Transaction Processing Performance Tool allows users to specify any Session Keywords or Terminal Keywords that are needed for an experiment in the **Transaction Prototypes File**. For detailed information on how to set up these keywords, click [here](#).

---

## User Input File Requirements

### Transaction Prototypes File ( for TA use )

Transaction Prototypes File locates each transaction procedure source file and keywords that will be used in an experiment. The user must provide the path to this file through the user interface before an experiment can be executed.

#### ● File Name Restriction

None

#### ● Content Convention

The Transaction Prototypes File consists of two sections -- prototype section and [keyword](#) section; the two sections are separated by a line with the mark '%%' :

prototype definitions

%%

keyword definitions

The "prototype definitions" section specifies what transaction procedures will be used for an experiment. Each procedure is represented by its signature written in the format:

```
ReturnType TransactionName ( ParameterType1 ParameterName1, ParameterType2  
ParameterName2, ... ) ;
```

Each signature should correspond to the signature in the corresponding Transaction Procedure Source files.

The "keyword definitions" section specifies any session or terminal keywords that will be involved in an experiment. Each keyword must correspond to a parameter name defined in one of the transaction prototypes.

To define session keywords, write the definition in the format of:

```
Session Keywords:  
keyword1
```

*keyword2*

:  
:

Similarly, terminal keywords must be defined in the format of :

**Terminal Keywords:**

*keyword1*

*keyword2*

:  
:

The order of session keyword definition and terminal keyword definition is interchangeable.

● **Sample File**

```
int Login ( Connection con, String user, String passwd ) ;  
int AddCourse ( Connection con, String user, String course ) ;  
int DropCourse ( Connection con, String user, String course ) ;
```

%%

Session Keywords:

user

passwd

Terminal Keywords:

user

passwd

course

---

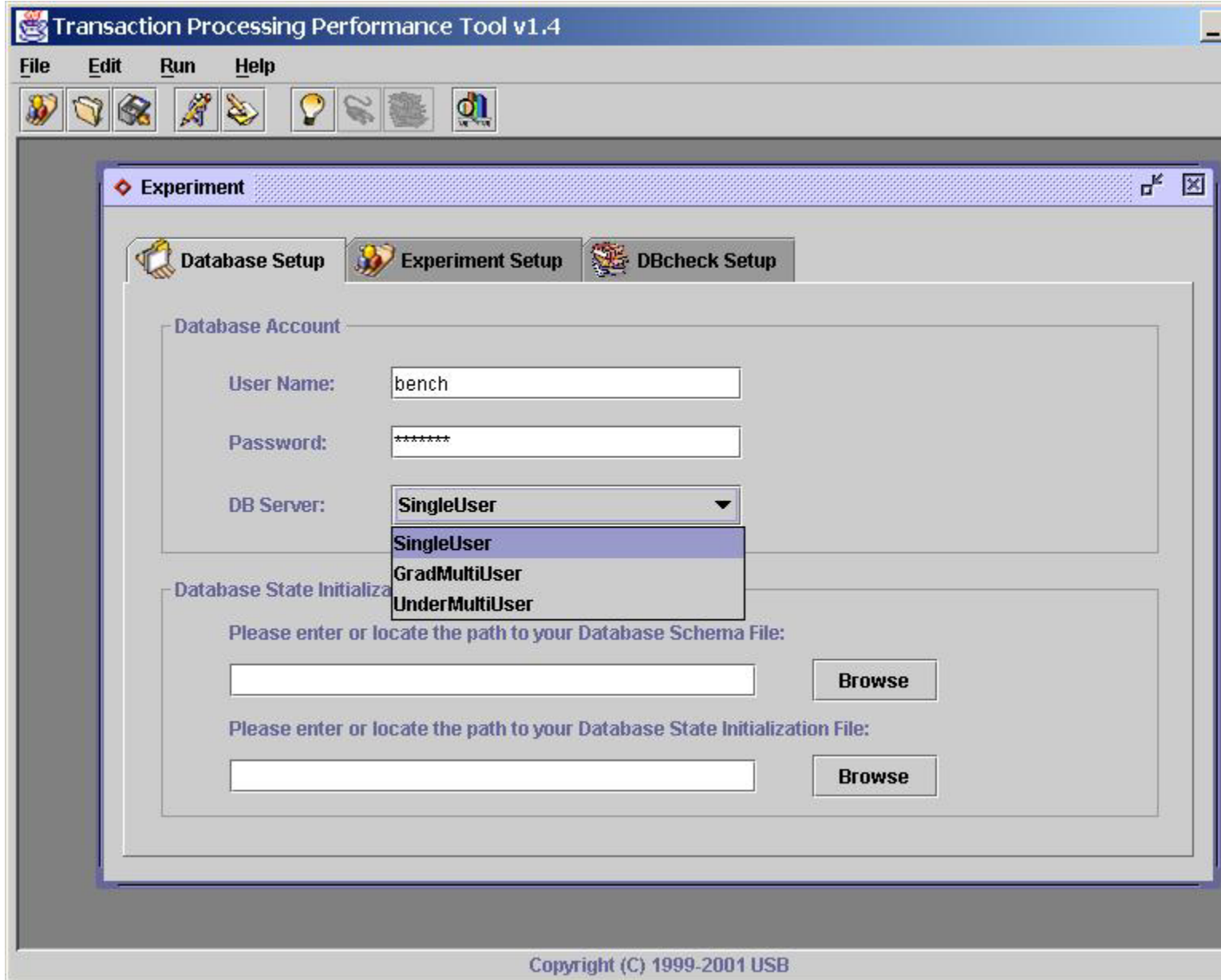
## [How to edit Sybase server URL? \( For TA Use\)](#)

In order to run TPPT in multi-user mode, a .txt file named "[serverNames.txt](#)" must be present in the same directory where TPPT class files are located. When TPPT is started without such file present, it will run experiments in single-user mode. However, the server file can be later added into the directory to activate [multi-user mode](#), once multi-user is on, this file is no longer editable until user quits running TPPT. The format for entering URL's for Sybase server is explained as comments in the server file..

---

```
GradMultiUserServer*jdbc:sybase:Tds:sbgrad.cs.sunysb.edu:5000  
UnderMultiUserServer*jdbc:sybase:Tds:sbntdbm.translab.cs.sunysb.edu:5000  
END  
//this is a sample file for server names,  
//enter the server url addresses after the '*' symbol accordingly above,  
//since '*' is a token, it can't appear elsewhere.  
//everything must be entered before END  
//only the first two correctly parsed urls are used and the order of grad  
//and under servers should NOT be changed.
```


```
//Don't leave a blank line
//GradMultiUserServer*jdbc:sybase:Tds:sbgrad.cs.sunysb.edu:5000
//UnderMultiUserServer*jdbc:sybase:Tds:sbntdbm.translab.cs.sunysb.edu:5000
```

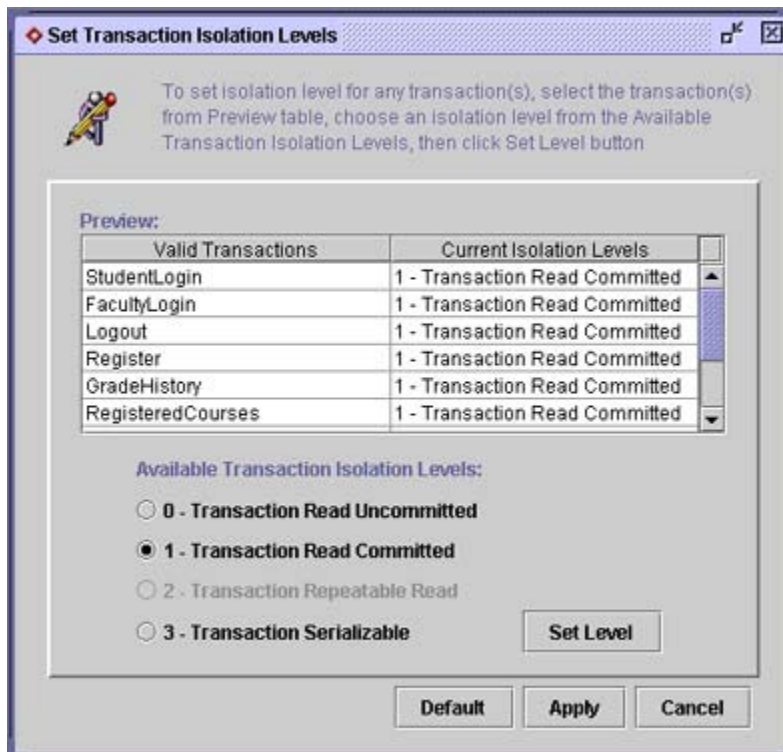


## [How to set transaction isolation levels?](#)

Transaction isolation level is an important factor that affects the performance of a database application. To prevent erroneous user input, you must enter or locate the path to your [Transaction Prototypes File](#) through [Experiment Setup Window](#) before you are allowed to set transaction isolation levels.

To set isolation levels for different transactions, first click **Edit** on the [menu bar](#), then choose **Set**

**Isolation Levels**. Or, click the  icon on the [tool bar](#). A **Set Transaction Isolation Levels** window shall be displayed:



(You may also open this window by clicking the Set Isolation Level button in [Experiment Setup](#) panel of an [Experiment](#) window. )

The default isolation level for each transaction is level 1, Transaction Read Committed. To change levels for any of your transaction, please use one of the selection modes below:

- **Single Selection Mode**

To set isolation level for a transaction, first highlight the transaction from **Preview Table**, then select your desired isolation level from the **Available Transaction Isolation Levels** list, click **Set Level**. You shall see that the isolation level for that transaction is changed in the preview table. Click **Apply** to confirm your changes, or click **Cancel** to abandon all your changes.

- **Multi Selection Mode**


You may also set one isolation level for a group of transactions in one snap! First highlight all the transactions whose level you would like to change from the **preview table**, then select the desired isolation level from the **Available Transaction Isolation Levels** list, click the **Set Level** button. You shall see that the isolation levels for those transactions are changed in the preview table. Click the **Apply** button to confirm your changes, or click the **Cancel** button to abandon all your changes.

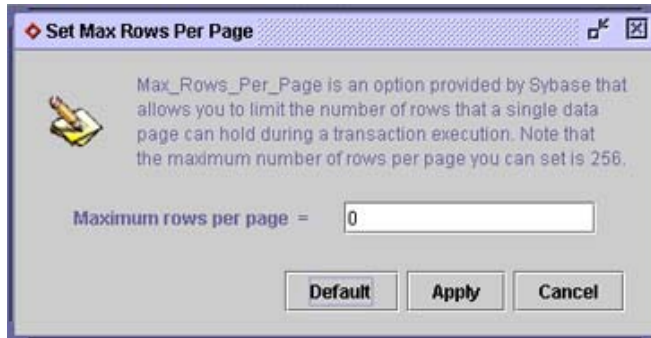
- **Set Isolation Levels back to Default**

If you need to abandon all your changes and set isolation levels back to default for all transactions, click **Default** button, then click the **Apply** button to confirm your changes.

## [How to set max rows per page?](#)

To set max\_rows\_per\_page value for all your tables, first click **Edit** on the [menu bar](#), then choose **Set**

**Max\_Rows\_Per\_Page**, or, simply click the  icon on the [tool bar](#). A **Set Max\_Rows\_Per\_Page** window shall be displayed:



(You may also open this window by clicking the Set Max\_Rows\_Per\_Page button in [Experiment Setup](#) panel of an [Experiment](#) window. )

### • **User Defined Max\_Rows\_Per\_Page Value**

To set max\_rows\_per\_page to your desired value, move the cursor to the text field appearing next to the line "Maximum rows per page = ", then type in a positive integer value that you would like to set. Click the **Apply** button to confirm your changes, or click the **Cancel** button to abandon all your changes. Note that the largest value you can set is **256** based on the requirement of Sybase.

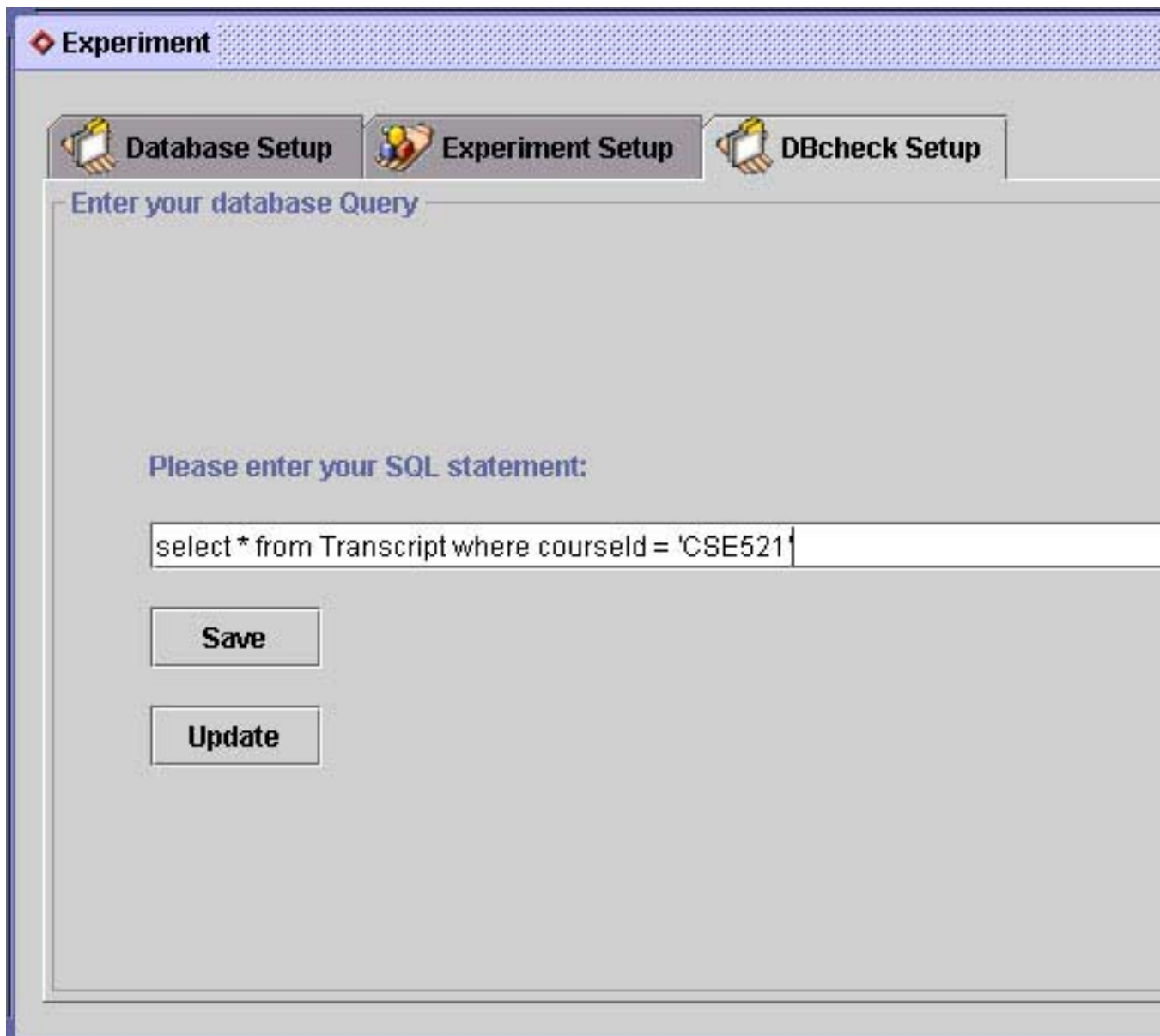
### • **Default Max\_Rows\_Per\_Page Value**


The default value for max\_rows\_per\_page is 0, which means that Sybase will manage the number of rows in a data page for you. To set max\_rows\_per\_page to default value, click the **Default** button, then click **Apply** to activate your change.

---

## [How to query the database?](#)

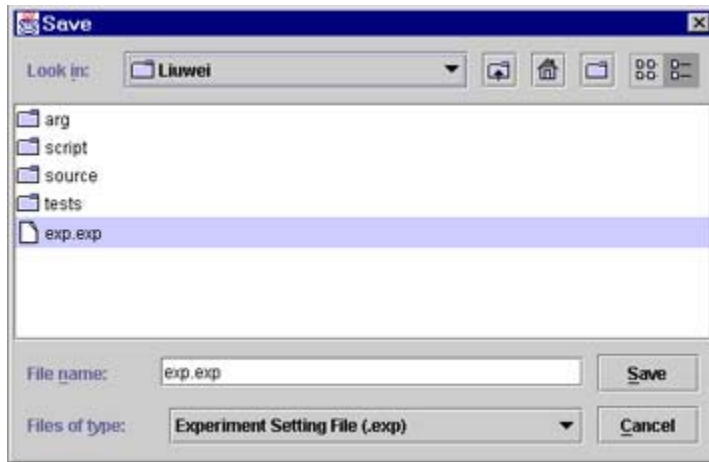
Firstly, make sure your last run of experiment was successful, in which case, the response time frame should have shown up. Secondly, bring up the experiment setup window. Then click on **DBCheck Setup** tab on the top of [Experiment frame](#), a tabbed pane shall be displayed:



then enter your database query in plain SQL statement, try to make your query succinct in both the query itself and the results, as TPPT will only display the first 100 tuples returned from any query. Make sure your SQL statement is correct, then simply click the  icon on the [tool bar](#) to start your query. Or, you may choose "Check Database Status" from the submenu under "Edit" on the tool menu.

- **Save your Database query**

To save your database query, click on the "save" button on **checkDB Setup** pane. A file chooser will come up which you can use to save the current experimental setting including database query.



### • Update your Database query

If you just entered a new query or made a change to your existing database query, you may "inform" the **TPPT tool** by pressing the "**Update**" button. If successful, a pop-up window should appear.



---

### **Tips** updated regularly

1. If you get an error while running an experiment which says "Experiment aborted due to errors in transaction \*\*\*\*\* class". Then there is possibly an error in your transaction file, which you should edit and recompile. However, to test your new class files, you must quit the current session of TPPT and restart a new one, as the loading of user source files is not run-time.
  2. TPPT requires user transaction classes return certain values so it can determine the execution status of each transaction, namely, -5 is returned when an sql Deadlock exception occurred, 0 is returned when transaction ran to completion, -2 is returned when transaction got a logic failure, -1 is returned when a java or sql exception (non-deadlock) occurred. Make sure your transactions return these values accordingly. See more about [how to write user-defined procedure files](#).
-

## Performance Measurement Report

At the end of a successful experiment, TPPT generates and displays a detailed performance measurement report on the screen.

Transaction	Result Type	Total Time	Occurrence	Avg Respon	Isolation level
StudentLogin	Success	1902.0	8	237.75	1
StudentLogin	Deadlocked	0.0	0	NA	1
StudentLogin	Logic Fail	0.0	0	NA	1
FacultyLogin	Success	411.0	2	205.50	8
FacultyLogin	Deadlocked	0.0	0	NA	8
FacultyLogin	Logic Fail	0.0	0	NA	8
Logout	Success	0.0	5	0.00	8
Logout	Deadlocked	0.0	0	NA	8

Overall Average Response Time (ms): 482.80  
Average Throughput (transactions / sec): 3.43  
Total number of deadlocks occurred: 0  
Experiment Duration (seconds): 11.37

Save Report Close Report

Copyright (C) 1999-2001 USDB

The report includes two group of statistics: Individual Transaction Performance and Overall Experiment Performance.

**Individual Transaction Performance** data are shown in a table format in the upper half of the report window. Relevant information include:

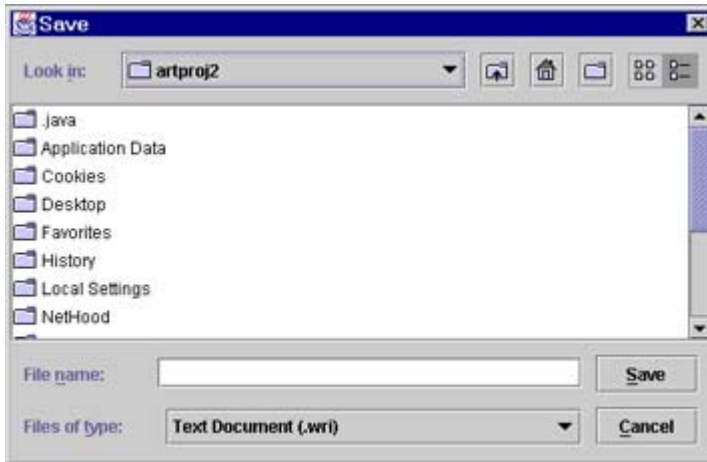
- **Transaction** : the name of a particular transaction type
- **Total Exec Time** : the total amount of time used for executing a particular transaction type ( measured in milliseconds by Sybase server )
- **Trans. Committed** : the total number of transactions committed successfully, excluding any transaction encountered logic errors
- **Avg Response Time** : the average response time of a particular transaction type (Avg Response Time = Total Exec Time / Trans. Committed). **New in version 1.3**, the response time is divided into a collection of three categories: **Success**, **Deadlocked** and **Logic failure**. The **Success** category includes time spent for transactions that ran into completion, **Deadlocked** combines time wasted when a transaction was aborted by Sybase due to deadlock with some other transactions, **Logic failure** applies to those transactions that terminates in the middle of its execution flow due to non-ideal results from logic checks set by user.

For example, the above report shows that for the transaction type 'Register', there were a total of 8 transactions committed successfully during 190ms of total execution time, so on average each Register transaction takes 23.75ms to complete.

**Overall Experiment Performance** statistics are shown in the lower half of the report window. Important data are:

- **Overall Average Response Time** : the average response time of all transaction types in milliseconds
- **Average Throughput** : the number of transactions committed per second
- **Total number of deadlocks occurred** : the total number of deadlocks occurred during the experiment

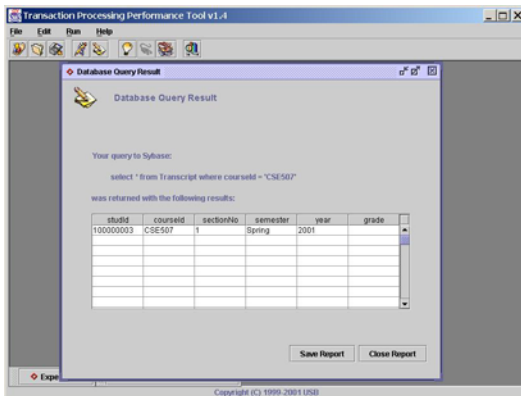
You may save the performance measurement report by clicking on the **Save Report** button. A file chooser will be shown (see below), which allows you to input the file name for your report. All report files will be saved in text format with an extension ".wri".



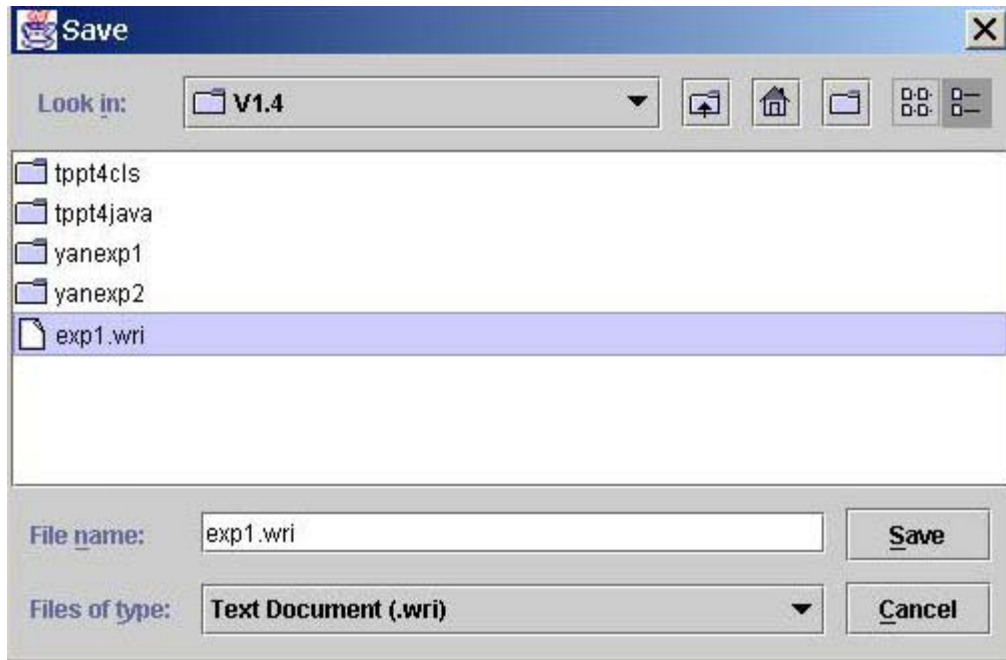
---

## Database Query Report

At the end of a successful query, TPPT generates and displays tuples returned by Sybase in a frame.



You can attach the query results to the end of a [Performance Measurement Report](#) you just saved



(by choosing an existing file in the file chooser, **note**, you can attach multiple query reports to one single performance report) or save them in a new file (by giving a new file name in the file chooser). The interface for saving report is similar to that for saving [Performance Measurement Report](#).

### Limitations:

1. User query shall not generate a result with more than **20** columns.
2. User query shall not generate a result with more than **100** rows.
3. The overflow in 1. and 2. will be discarded by TPPT before displaying your query result.
4. Only the first 80 characters of user database query will be shown in the final query report frame.

---

## [User Input File Requirements](#)

In order for an experiment to run properly, you must provide the following resource files:

- [Database Schema File](#)
- [Database Initialization File](#)
- [Transaction Prototypes File \( for TA use \)](#)
- [Transaction Procedure Source Files](#)
- [Terminal Script Files \( for TA use \)](#)
- [Transaction Argument Files \( for TA use \)](#)

Each of these files must follow its predefined requirement and convention. To view the requirement for a particular file type, click on the above hyperlink.

## User Input File Requirements

### Database Schema File

Database Schema File is used to set up the schema of a database, including tables, triggers, store procedures, indices, etc. By modifying this file, a user can execute experiments with different schema, and examine how different database design can affect the performance of an application. The user must provide the path to this file through the user interface before an experiment can be executed.

#### • File Name Restriction

None

#### • Content Convention

This file should contain all SQL statements needed to create tables, triggers, indices, store procedures, constraints and any other valid statements for setting up the database. Unlike the Database Initialization File, each statement in this file need not be written on one line. To guarantee the correctness and efficiency during database setup process, each SQL statement or store procedure is required to end with the statement "go" (see example below).

**NOTE:** Do NOT use double quotes " in this file, use single quotes ' only, otherwise error will occur in database setup process.

#### • Sample File

```
CREATE TABLE Student (  
    sid          char(9) NOT NULL,  
    tname       varchar(20) NULL,  
    sname       varchar(50) NOT NULL,  
    spasswd     varchar(20) NOT NULL,  
    sstatus     char(1) NOT NULL  
                CHECK (sstatus IN ('S')),  
    svalidity   char(1) NOT NULL  
)  
go  
  
CREATE INDEX MyIndex ON Student  
( tname  
)  
go  
  
ALTER TABLE Student  
    ADD PRIMARY KEY (sid)  
go  
  
CREATE PROC MyStoredProcedure @x int, @result int output AS  
    print 'Hello world!'  
    return 0  
go
```

## User Input File Requirements

### Database Initialization File

Database Initialization File is used for initializing the state of a database. By starting each experiment in the same initial database state, the results of different experiments can be meaningfully compared. The user must provide the path to this file through user interface before an experiment can be executed.

#### • File Name Restriction

None

#### • Content Convention

This file should only contain SQL INSERT statements. All statements must be Sybase compatible and each statement must be written on one line.

It is highly recommended that you add the statement "go" after a block of INSERT statements (see example below).

NOTE: You may also create stored procedures in Database Schema File to perform data insertion, then you may write stored procedure calls in this file.

#### • Sample File

```
INSERT INTO Students (sid, password) VALUES ('000000000', 'aaaaa')
INSERT INTO Students (sid, password) VALUES ('111111111', 'bbbb')
INSERT INTO Students (sid, password) VALUES ('222222222', 'ccccc')
INSERT INTO Students (sid, password) VALUES ('333333333', 'dddd')
go
```

```
INSERT INTO Courses (cid, cname) VALUES ('CSE100', 'Intro to CS')
INSERT INTO Courses (cid, cname) VALUES ('CSE200', 'Advanced CS')
go
```

```
INSERT INTO Registered (sid, cid) VALUES ('111111111', 'CSE100')
INSERT INTO Registered (sid, cid) VALUES ('222222222', 'CSE200')
INSERT INTO Registered (sid, cid) VALUES ('333333333', 'CSE100')
go
```

---

## User Input File Requirements

### Transaction Prototypes File ( for TA use )

Transaction Prototypes File locates each transaction procedure source file and keywords that will be used in an experiment. The user must provide the path to this file through the user interface before an experiment can be executed.

#### • File Name Restriction

None

#### • Content Convention

The Transaction Prototypes File consists of two sections -- prototype section and [keyword](#) section; the two sections are separated by a line with the mark '%%' :

prototype definitions

%%

keyword definitions

The "prototype definitions" section specifies what transaction procedures will be used for an experiment. Each procedure is represented by its signature written in the format:

```
ReturnType TransactionName ( ParameterType1 ParameterName1, ParameterType2  
ParameterName2, ... ) ;
```

Each signature should correspond to the signature in the corresponding [Transaction Procedure Source files](#).

The "keyword definitions" section specifies any session or terminal keywords that will be involved in an experiment. Each keyword must correspond to a parameter name defined in one of the transaction prototypes.

To define session keywords, write the definition in the format of:

**Session Keywords:**

```
keyword1  
keyword2  
:  
:
```

Similarly, terminal keywords must be defined in the format of :

**Terminal Keywords:**

```
keyword1  
keyword2  
:  
:
```

The order of session keyword definition and terminal keyword definition is interchangeable.

## • Sample File

```
int Login ( Connection con, String user, String passwd ) ;  
int AddCourse ( Connection con, String user, String course ) ;  
int DropCourse ( Connection con, String user, String course ) ;
```

%%

Session Keywords:

```
user  
passwd
```

Terminal Keywords:

```
user  
passwd  
course
```

---

## User Input File Requirements

### Transaction Procedure Source Files

Transaction Procedure Source Files is a collection of source files written by the database application programmer ( the student ). Each file defines one database transaction using the programming language, Java. Thus, within a source file there should exist the definition of the "transaction class" and the "transaction procedure" as a method of the class. The user must provide the path to the directory ( Transaction Source Files Directory ) that contains all the **pre-compiled** transaction files.

#### • File Name Restriction

*TransactionName* + ".Class.java"

For example, if a "transaction procedure" is called "Login", then the "transaction class" name should be "LoginClass", and its file name should be "LoginClass.java". ( After compilation, the file name will be "LoginClass.class". )

#### • Content Convention

A "transaction class" and its "transaction procedure" must follow the following convention:

1. Transaction class must be declared as a **public** class.
2. Transaction class name must correspond to transaction procedure name defined later in the file. For example, if the transaction class name is "LoginClass", then the transaction procedure name must be "Login".
3. Transaction class procedure must be declared as **public**.
4. Transaction procedure may have arbitrary number of parameters. The first parameter of the procedure must be of **java.sql.Connection** type, which can be used to create or execute SQL statments. Each parameter must be of java Object type, such as String, Integer, Float, Double, etc.
5. The return type of the transaction procedure must be **int**, indicating different exit states:

**0** -- If transaction completes succesfully

**-2** -- Normal transaction failure due to integrity constraints or pre-condition failure, etc. ( For example, for the transaction called *Register*, you probably need to check if the student has taken all prerequisite courses, if not, transaction should not proceed. Thus, in such situation, the transaction should return the value 1.

**-5** -- If a deadlock occurred. Your transaction should detect the deadlock in the clause "... catch (SQLException e) { ... }" and return the value -5 if a deadlock is caught.

**-1** -- Transaction failed due to SQLExceptions other than deadlock. ( For example, your SQL statement contains syntax error. )

#### • Sample File ( LoginClass.java )

```
import java.sql.*;
```

```

public class LoginClass {

    /* Member variable declaration if needed */

    ... ..

    public int Login ( Connection con, String user, String password ) {

        /* code for the transaction ... */

    }

    /* Other procedures that might be used by the procedure Login */

    ....

}

```

---

## User Input File Requirements

### Terminal Script Files ( for TA use )

A Transaction Script File defines a sequence of transactions that will be executed on one simulated terminal. If a terminal finishes executing all the transactions in its corresponding script file before the experiment execution time runs out, it loops back to the beginning of the file and re-executes the script.

The user must provide a script file for each terminal he would like to simulate. The path to the directory that contains all script files must be provided by the user before an experiment can be executed.

#### ● File Name Restriction

"Terminal" + *TerminalNumber* + ".scp"

In order to run an experiment, the user must specify the total number of terminals he would like to simulate. The first terminal gets the terminal number "0", the second gets the terminal number "1", and so on. If, for example, the total number of terminals is 10, then the script files must be named "Terminal0.scp", "Terminal1.scp", "Terminal2.scp", ... and "Terminal9.scp".

#### ● Content Convention

A Transaction Script File consists of several transaction sessions. Each transaction session contains a number of transactions that will be simulated to be executed by one client of the user database application. Transactions are represented using the transactions procedure names defined in [Transaction Prototypes File](#). Transaction sessions are separated by a line with the sign ">>" . For example,

```

>>
... transaction session 1...

>>
... transaction session 2...

>>
... transaction session 3...

```

... ..

### • **Sample File**

In a Student Registration System, for instance, there might be several students waiting on line to use a terminal for registering courses. Thus, a typical transaction script file representing transactions executed by these students on a terminal may look as follows:

(File name: Terminal0.scp)

```
>>
Login
AddCourse
AddCourse
DropCourse
Logout

>>
Login
AddCourse
AddCourse
Logout

>>
Login
SearchCourseInfo
ViewGrades
Logout
```

... ..

---

## User Input File Requirements

### **Transaction Argument Files ( for TA use )**

A Transaction Argument File contains argument values for a particular parameter type that will be passed to a transaction procedure that requires an argument of that type during the execution of a transaction. Each parameter declared in the Transaction Prototype File corresponds to a Transaction Argument File, except for the first parameter with parameter type "Connection". The user must provide the path to the directory ( Transaction Argument Files Directory ) that contains all the transaction argument files before an experiment begins.

### • **File Name Restriction**

*TransactionProcedureParameterName* + ".arg"

The name of a Transaction Argument File must agree with the parameter name declared in Transaction Prototype File. For example, if the Transaction Prototype File contains the following transaction signature:

```
int Login (Connection con, String user, String password ) ;
```

then the user must provide the argument file "user.arg" for the parameter "user", and the argument file "password.arg" for the parameter "password". If the same parameter appears in more than one transaction prototype, only one argument file for that parameter is required.

Note: No argument file is needed for the first parameter "con" of type "Connection" in the above transaction prototype.

### • Content Convention

One argument value per line.

### • Sample Files

( File name: user.arg )

Amy  
Cathy  
Helen  
Brian  
John

( File name: courses.arg )

CSE100  
ESE320  
EGL210  
BIO151  
MAT309

---

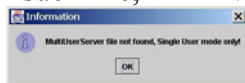
## The history of TPPT®

TPPT has been in the works for over two years. The latest version v1.4 is the closure to the MS Projects of several people under the tutorship of Prof. Arthur Bernstein: Helen Yi Ren and Sue Jie built the first version of TPPT in 1999 in which the multi-user and single-user modes were run separately. Yang Wang and Yan Zhang later contributed to some improvement in the Synchronization part of TPPT (single-user mode). From Fall 2000 through 2001, TPPT underwent several transformations by Jiang Yin which includes: the combination of multi-user mode and single-user mode (v1.2); the separate tabulation of response times for different scenarios in the execution of a transaction (v1.3); the addition of a new interface which allows user to query into database after each successful experiment (v1.4). For more information on the modifications to the original tool, please read on.

### • v1.2 from v1.1:

1. Combined multi-user mode with single-user mode, a .txt file named "[serverNames.txt](#)" which contains URL's for different servers is expected to exist in the same directory as the class files for running TPPT. In the absence of such file, TPPT will start running in single-user mode and inform the user through a

pop-up window.



2. Expanded the functionality of "save experiment setting" to cover all user input except for isolation level and maxRowPerPage.

### • v1.3 from v1.2:

1. The response time for each transaction is collected categorically by the outcome of each transaction. The execution of a transaction can be in one of the four following cases:

- success: the transaction ran to completion, no SQL or java exception occurred.
- logic failure: the transaction which usually consists of several SQL statements, however it quit prematurely because the result from some SQL statement(s) doesn't warrant the continuation of the rest of the transaction.
- deadlocked: if transactions requiring access to the same system resources are executed at the same time, they might deadlock each other, it is then Sybase's choice to allow only one of them to go through and abort all other transactions. The aborted transactions will then be re-run twice, at which time they, if still haven't gone through, will be aborted.
- exception: needless to say, if this happens, the current experiment should be aborted.

2. Use the drop-down list to allow user to choose from different Sybase server.

● **v1.4 from v1.3:**

1. Save experiment now stores all information about an experiment, however, it's the user's responsibility to keep track of changes made into user input files, e.g., prototype file, scripts, etc.
2. Prototype file now can be loaded during run-time.
3. A new interface was introduced to allow user query into database after at least one successful experiment, if the query runs without problem, the results will be displayed and saved should user wish so.
4. Each script will be run once and only once, instead of being looped.
5. Synchronization in single-mode now employs stored procedure to assure exclusiveness while one user is running experiment, eliminating a possible bug in earlier versions.