Unit Tests with PostgreSQL Backends

In order to easily stand up a Postgres database for unit and integration testing, we need a class that can encapsulate the effort, to an easy call surface.

Below we have a helper class that will manage the life-cycle of a test database, in PostgreSQL.

Given an admin user account and a set of table definitions to create, it will generate a test user, test database, and create the tables within it.

When done, it will handle removal of the test user and database.

Usage

Here are steps to perform, so that an instance of DbInstanceHelper_PostGres, will manage the life-cycle of a test database and test user.

Create an instance of the helper class, DbInstanceHelper_PostGres.
And, give it the Central Config name of the test postgres instance and test admin account:

```
var dbh = new DbInstanceHelper_PostGres();
dbh.Cfg_CentralConfig_DbCred_NameString = "PostGresTestAdmin";
```

For it to do useful work, we need to give it at least one table definition, that it will create in the test database:

```
// Create the test tabke...
string tablename = "tbl_Icons";
var tch = new TableDefinition(tablename, "postgres");

// Give it a pk...
var res1 = tch.Add_Pk_Column("Id", OGA.Postgres.DAL.Model.ePkColTypes.uuid);
if (res1 != 1)
    Assert.Fail("Wrong Value");

// Add a name column...
var res2 = tch.Add_String_Column("IconName", 50, true);
```

```
if (res2 != 1)
    Assert.Fail("Wrong Value");

// Add some other columns...
var res3 = tch.Add_Numeric_Column("Height", OGA.Postgres.DAL.Model.eNumericColTypes.integer, true);
if (res3 != 1)
    Assert.Fail("Wrong Value");

var res4 = tch.Add_Numeric_Column("Width", OGA.Postgres.DAL.Model.eNumericColTypes.integer, true);
if (res4 != 1)
    Assert.Fail("Wrong Value");

var res5 = tch.Add_String_Column("Path", 255, true);
if (res5 != 1)
    Assert.Fail("Wrong Value");
```

Pass the table definition to the helper:

```
var resadd = dbh.Add_TableDef(tch);
if (resadd != 1)
    Assert.Fail("Wrong Value");
```

> NOTE: Add_TableDef() can be called a number of times, if your test database needs to have multiple tables.

Once you've populated the helper with the list of desired tables, tell it to create the database/user/tables, with this:

```
var rescreate = dbh.Standup_Database();
if (rescreate != 1)
    Assert.Fail("Wrong Value");
```

On success, the helper contains properties, where you can read back, the created database name, test user's name and password.
You can use these in your unit and integration tests, to interact with the created tables as the created test user.

> NOTE: Retain a reference to the helper, so that you can leverage it to cleanup the test database and test user.

When Finished

At the end of your unit tests, make a call to the helper's Teardown_Database() call.

```
dbh.Teardown_Database();
```

This will delete the test database, and the test user.

And of course, be sure to dispose the helper instance, to cleanup its references, with this:

```
dbh?.Dispose();
```

> NOTE: For unit tests that may end, early, wrap your tests in a try-finally, to ensure the helper's dispose method gets called.

Here's a simple try-finally that creates the helper, uses it, and ensures its disposal:

```
DbInstanceHelper_PostGres dbh = null;
try
{
    // Create the instance...
    var dbh = new DbInstanceHelper_PostGres();
    dbh.Cfg_CentralConfig_DbCred_NameString = "PostGresTestAdmin";

    // Create a single table definition...
    TableDefinition tch = null;
    {
      ...
    }

    // Add the table def to the helper, so it can be created...
    var res = dbh.Add_TableDef(tch);

    // Tell the helper to create the database and tables...
    var rescreate = dbh.Standup_Database();
    if (rescreate != 1)
        Assert.Fail("Wrong Value");

// The test database is live, here and can be used in unit testing.
```

```
    // Now, teardown the database...
    dbh.Teardown_Database();
}
finally
{
    dbh?.Dispose();
}
```

Helper Class

Here's the current version of the helper class, DbInstanceHelper_PostGres:

> NOTE: It depends on OGA.Postgres.DAL and OGA.SharedKernel.Lib.

```
namespace MOVEELSEWHERE_SP.Helpers
{
    /// <summary>
    /// Used to manage lifecycle of a test database and associated test user.
    /// It is purposely generic in construction.
    /// See this page for usage: https://wiki.galaxydump.com/link/491
    /// </summary>
    public class DbInstanceHelper_PostGres : IDisposable
    {
        #region Private Fields

        static private string _classname = nameof(Postgres_Tools);

        static private volatile int _instancecounter;

        private bool disposedValue;

        private cPostGresDbConfig dbcfg;

        private Postgres_Tools _dbtool;

        private List<TableDefinition> _tables;

        #endregion
```

```
        #region Public Properties

        public int InstanceId { get; private set; }

        /// <summary>
        /// Exposes the username associated with the created test database.
        /// This will autopopulate with a test username when the database is created.
        /// </summary>
        public string TestUsername { get; private set; }

        /// <summary>
        /// Exposes the password for the test user.
        /// This will autopopulate when the database is created.
        /// </summary>
        public string TestUserPassword { get; private set; }

        /// <summary>
        /// Exposes the datbase name for the created test database.
        /// This will autopopulate with a test name when the database is created.
        /// </summary>
        public string TestDbName { get; private set; }

        /// <summary>
        /// Hostname of database server.
        /// Populated here, so a datacontext config instance can be composed without an independent call to
central config.
        /// </summary>
        public string Hostname { get; private set; }

        /// <summary>
        /// Set this to the name of the database creds to retrieve from central config.
        /// The creds required will be a user account with adequate privileges to create users and databases.
        /// By default, the cred name is set to: 'PostGresTestAdmin'.
        /// </summary>
        public string Cfg_CentralConfig_DbCred_NameString { get; set; } = "PostGresTestAdmin";

        #endregion


        #region ctor / dtor
```

```
public DbInstanceHelper_PostGres()
{
    // Create the database and username...
    this.TestDbName = ValueGenerators.GenerateTestDbName();
    this.TestUsername = ValueGenerators.GenerateTestUserName();
    this.TestUserPassword = ValueGenerators.GeneratePassword();

    this._tables = new List<TableDefinition>();

    dbcfg = null;
}

protected virtual void Dispose(bool disposing)
{
    if (!disposedValue)
    {
        if (disposing)
        {
            // TODO: dispose managed state (managed objects)

            this._dbtool.Dispose();

            dbcfg = null;
        }

        // TODO: free unmanaged resources (unmanaged objects) and override finalizer
        // TODO: set large fields to null
        disposedValue = true;
    }
}

// // TODO: override finalizer only if 'Dispose(bool disposing)' has code to free unmanaged resources
// ~DbInstanceHelper_PostGres()
// {
//     // Do not change this code. Put cleanup code in 'Dispose(bool disposing)' method
//     Dispose(disposing: false);
// }

public void Dispose()
```

```
{
    // Do not change this code. Put cleanup code in 'Dispose(bool disposing)' method
    Dispose(disposing: true);
    GC.SuppressFinalize(this);
}

#endregion



#region Public Methods

/// <summary>
/// Call this to include a table to the test database schema.
/// NOTE: This must be called before calling the standup method.
/// </summary>
/// <param name="tdef"></param>
/// <returns></returns>
public int Add_TableDef(TableDefinition tdef)
{
    if (disposedValue)
    {
        // Already disposed.
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Add_TableDef)} - " +
            $"Instance already disposed. Cannot use.");

        return -20;
    }

    if(tdef == null)
    {
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Add_TableDef)} - " +
            $"Given table definition not defined. Cannot use.");

        return -1;
    }

    if(string.IsNullOrEmpty(tdef.tablename.Trim()))
    {
```

```
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Add_TableDef)} - " +
                $"Given table has empty name. Cannot use.");

            return -2;
        }

        // Check if the table is already defined...
        var ct = this._tables.FirstOrDefault(t => (t.tablename ?? "") == (tdef.tablename ?? ""));
        if(ct != null)
        {
            // Table already exists.
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Add_TableDef)} - " +
                $"Given table already exists. Cannot use.");

            return -3;
        }

        // Add the table to the pending list...
        this._tables.Add(tdef);

        return 1;
    }

    /// <summary>
    /// This method will:
    ///     Verify the test admin creds work,
    ///     Create a user account that will own the test database,
    ///     Set the user's password,
    ///     Create the test database,
    ///     Set the database owner to the created user,
    ///     Create the set of tables, passed to Add_TableDef.
    /// Returns 1 for success.
    /// Returns -1 if the database already exists.
    /// -2 for access errors.
    /// </summary>
    /// <returns></returns>
    public int Standup_Database()
    {
```

```
bool success = false;

if (disposedValue)
{
    // Already disposed.
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Instance already disposed. Cannot use.");

    return -20;
}

// Wrap this in a try-finally, so we can teardown database and user on failure.
try
{
    // Setup the database tool instance...
    // This will retrieve the admin creds and setup the db tool instance that we will for management.
    var res1 = this.SetupDbTool();
    if (res1 != 1)
    {
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
            $"Failed to setup database management tool.");

        return -1;
    }

    // Verify connectivity...
    var resconn = this._dbtool.TestConnection();
    if(resconn != 1)
    {
        // Failed to connect.

        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
            $"Failed to connect with database engine.");

        return -2;
    }
```

```
// Check if the test user exists...
var resuserexists = this._dbtool.Does_Login_Exist(this.TestUsername);
if (resuserexists < 0)
{
    // Failed to connect.

    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to connect with database engine.");

    return -2;
}
else if(resuserexists == 1)
{
    // Username already exists.
    // We will use it below.
}
// User doesn't exist.

// Create the test user...
var resadduser = this._dbtool.CreateUser(this.TestUsername);
if(resadduser != 1)
{
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to create test username.");

    return -3;
}

// Make sure the user's password is known...
var rescp = this._dbtool.ChangeUserPassword(this.TestUsername, this.TestUserPassword);
if(rescp != 1)
{
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to set test user's password.");

    return -4;
}
```

```
// Check if the given database exists...
var respres = this._dbtool.Is_Database_Present(this.TestDbName);
if(respres < 0)
{
    // Failed to connect.

    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to connect to database engine.");

    return -5;
}
if(respres == 1)
{
    // Database already exists.

    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Database already exists. Won't standup again.");

    return -6;
}
// Database doesn't exist.
// We will create it below.

// Create the database...
var rescreate = this._dbtool.Create_Database(this.TestDbName);
if(rescreate != 1)
{
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to create database.");

    return -7;
}

// Set the owner...
var resowner = this._dbtool.ChangeDatabaseOwner(this.TestDbName, this.TestUsername);
if(resowner != 1)
```

```
{
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Failed to set database owner.");

    return -8;
}

// Before we add tables, we must switch our connection to the created database.
// Swap our connection to the created database...
{
    this._dbtool.Dispose();
    System.Threading.Thread.Sleep(500);
    this._dbtool = new Postgres_Tools();
    this._dbtool.Username = this.dbcfg.User;
    this._dbtool.Hostname = this.dbcfg.Host;
    this._dbtool.Password = this.dbcfg.Password;
    // Use the test database as our connection target...
    this._dbtool.Database = this.TestDbName;

    // Verify we can connect to the test database...
    var restdb = this._dbtool.TestConnection();
    if(restdb != 1)
    {
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
            $"Failed to change client connection to test database.");

        return -9;
    }
}

// Add each table...
foreach(var t in this._tables)
{
    // Add the current table...
    var resta = this._dbtool.Create_Table(t);
    if(resta != 1)
    {
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
```

```
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
                $"Failed to create table in database.");

            return -10;
        }
    }

    // Swap our connection back to the postgres database...
    // This will be needed when we teardown the test database.
    {
        this._dbtool.Dispose();
        System.Threading.Thread.Sleep(500);
        this._dbtool = new Postgres_Tools();
        this._dbtool.Username = this.dbcfg.User;
        this._dbtool.Hostname = this.dbcfg.Host;
        this._dbtool.Password = this.dbcfg.Password;
        this._dbtool.Database = this.dbcfg.Database;

        // Verify we can connect to the test database...
        var restdb = this._dbtool.TestConnection();
        if(restdb != 1)
        {
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
                $"Failed to change client connection to postgres database.");

            return -11;
        }
    }

    OGA.SharedKernel.Logging_Base.Logger_Ref?.Info(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
        $"Database stood up, with test user as owner.");

    // Set the success flag...
    success = true;
    return 1;
}
catch(Exception e)
{
```

```
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(e,
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Standup_Database)} - " +
                $"Exception caught while standing up test database.");

            return -12;
        }
        finally
        {
            // See if the above steps passed or not...
            if(!success)
            {
                // Failed to stand up everything required.

                // Teardown the database and user...
                this.Teardown_Database();
            }
        }
    }

    /// <summary>
    /// Drops the test database, test user, and such.
    /// NOTE: You still have to call Dispose() to get rid of the tool management instance.
    /// </summary>
    public void Teardown_Database()
    {
        if (disposedValue)
        {
            // Already disposed.
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
                $"Instance already disposed. Cannot use.");
        }

        try
        {
            if(this._dbtool == null)
            {
                // Tool not set.

                OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
```

```
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
        $"Database management tool is null.");

    return;
}

// Check that the database exists...
var res1 = this._dbtool.Is_Database_Present(this.TestDbName);
if(res1 == 0)
{
    OGA.SharedKernel.Logging_Base.Logger_Ref?.Info(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
        $"Database doesn't exist.");
}
else if(res1 == 1)
{
    // Database exists.
    // We will drop it.

    var res2 = this._dbtool.Drop_Database(this.TestDbName, true);
    if(res2 != 1)
    {
        // Failed to delete.

        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
            $"Failed to drop test database.");
    }
}
else
{
    // Access error.

    OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
        $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
        $"Access error while checking for test database.");
}

// Drop the test user...
var res3 = this._dbtool.DeleteUser(this.TestUsername);
```

```
        if(res3 != 1)
        {
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
                $"Error occurred while deleting test user.");
        }
    }
    catch(Exception e)
    {
        OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(e,
            $"{_classname}:{this.InstanceId.ToString()}:{nameof(Teardown_Database)} - " +
            $"Exception caught while tearing down test database.");
    }
}

#endregion


#region Private Methods

/// <summary>
/// This will retrieve test admin creds and create the management tool instance.
/// </summary>
/// <returns></returns>
private int SetupDbTool()
{
    try
    {
        // Retrieve database admin user creds...
        var rescreds = GetTestDatabaseUserCreds();
        if (rescreds != 1 || this.dbcfg == null)
            return -1;

        // Create the database tool...
        _dbtool = new Postgres_Tools();
        _dbtool.Username = this.dbcfg.User;
        _dbtool.Hostname = this.dbcfg.Host;
        _dbtool.Password = this.dbcfg.Password;
        _dbtool.Database = this.dbcfg.Database;
```

```
            this.Hostname = this.dbcfg.Host;

            return 1;
        }
        catch(Exception e)
        {
            OGA.SharedKernel.Logging_Base.Logger_Ref?.Error(e,
                $"{_classname}:{this.InstanceId.ToString()}:{nameof(SetupDbTool)} - " +
                $"Exception caught while setting up database management tool.");

            return -2;
        }
    }

    private int GetTestDatabaseUserCreds()
    {
        if (string.IsNullOrWhiteSpace(this.Cfg_CentralConfig_DbCred_NameString))
            return -1;

        var res =
MOVEELSEWHERE_SP.Helpers.CentralConfig.Get_Config_from_CentralConfig(this.Cfg_CentralConfig_DbCred_Nam
eString, out var config);
        if (res != 1)
            return -1;

        var cfg = Newtonsoft.Json.JsonConvert.DeserializeObject<cPostGresDbConfig>(config);
        if(cfg == null)
            return -1;

        this.dbcfg = cfg;
        return 1;
    }

    #endregion
  }
}
```

Revision #3
Created 13 October 2025 21:13:06 by glwhite
Updated 13 October 2025 22:14:43 by glwhite