Wt - Support #7373

SEH Exception on Wt:DBO rollback

12/19/2019 10:02 AM - Michael Tornack

Status:	New	**Start date:**	12/19/2019
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Hello,

I'm trying to implement a SQL storage class with your Wt::Dbo framework. In general it is working fine but with the rollback scenario I have problems.

I have function which accepts a lambda (fun). All database actions are performed within this function. If an error occurs in this lambda function I want to gracefully roll back and throw a SqlException.

Storage class

void Storage::execute(std::function<void (dbo::Session *session)> fun)

{

dbo::Transaction transaction{m_session};

bool exceptionCaught = false;

std::string errorMessage = "";

try {

fun(&m_session);

transaction.commit();

} catch (const std::exception& e) {

errorMessage = e.what();

exceptionCaught = true;

}

transaction.rollback();

if(exceptionCaught){

throw SqlException(errorMessage);

}

}

In my tests, the exception (std::exception& e) is caught. But when I try to commit new transactions in the database I get an exception (error: SEH exception with code 0xc0000005). Furthermore I noticed that the warning "1 Dirty Object" is shown. Can you give me a hint what is wrong with my setup?

The unit test looks like that btw:

Unit Test

```
TEST_F(TestStorage, RollBack){

auto storage = Storage::getInstance();

auto funInsert = [](dbo::Session *session){

auto user = std::make_uniquePersistence::User();

user->userName = "Joe";

user->id = 1;

session->add(std::move(user));

};

auto funFailed = [](dbo::Session *session){

dbo::ptrPersistence::User joe = session->findPersistence::User().where("user_name = ?").bind("Joe");

ASSERT_EQ(joe.get()->userName, "Joe");

ASSERT_EQ(joe.get()->id, 1);

joe.modify()->userName = "Jane";

// Check if username was changed

dbo::ptrPersistence::User jane = session->findPersistence::User().where("user_name = ?").bind("Jane");

ASSERT_EQ(jane.get()->userName, "Jane");

// Insert User with same id, this should trigger a unique constraint failure

auto user = std::make_uniquePersistence::User();

user->userName = "SameId";

user->id = 1;

session->add(std::move(user));

};

auto funFind = [](dbo::Session *session){

dbo::ptrPersistence::User joe = session->findPersistence::User().where("user_name = ?").bind("Joe");

ASSERT_EQ(joe.get()->userName, "Joe");

ASSERT_EQ(joe.get()->id, 1);

};

ASSERT_NO_THROW(storage->execute(funInsert));

ASSERT_THROW(storage->execute(funFailed), SqlException);

ASSERT_NO_THROW(storage->execute(funFind));

}
```

History

#1 - 12/19/2019 12:16 PM - Wim Dumon

Hello,

The intented use of Transaction to implement what you want is this:

```
void Storage::execute(std::function<void (dbo::Session *session)> fun)
{
    try {
        dbo::Transaction transaction{m_session};
        fun(&m_session);
    } catch (const std::exception& e) {
        throw SqlException(e.what());
    }
```

At first sight, I can't see something wrong with your code, it's just more verbose that it has to be. The behaviour you describe could be a bug. What backend are you using?

BR,

Wim.

#2 - 12/19/2019 12:24 PM - Michael Tornack

Hello,

I am using the Sqlite3 backend.

Your code snippet was aswell my first try. But with that I got aswell the exception. I refactored it after reading the example at (line 1484 onwards):

https://github.com/emweb/wt/blob/master/test/dbo/DboTest.C

Best regards,

#3 - 01/02/2020 12:19 PM - Roel Standaert

I think you mean to do m_session.discardUnflushed() rather than a rollback. The rollback will be performed automatically when the transaction goes out of scope.

#4 - 01/08/2020 08:58 AM - Michael Tornack

It seems that this solves my issue.

Thanks for your feedback.

I post my working code in case it is needed for others.

```
void Storage::execute(std::function<void (dbo::Session *session)> fun)
{
    dbo::Transaction transaction{m_session};
    bool exceptionCaught = false;
    std::string errorMessage = "";

    try {
        fun(&m_session);
        transaction.commit();
    } catch (const std::exception& e) {
        errorMessage = e.what();
        exceptionCaught = true;
    }

    if(exceptionCaught){
        m_session.discardUnflushed();
        throw SqlException(errorMessage);
    }
}
```

Still I think this behaviour should have be already done within your rollback scenario (as indicated by Wim Dumont).

Can you give me a heads up if in future you change something in the rollback of the Sqlite3 backend?

Best regards and thanks again

#5 - 01/08/2020 10:31 AM - Koen Deforche

Michael,

We do not actually rollback changes you made to in-memory objects ourselves since the intent of the transaction is to bring the database persisted state in sync with the in-memory state. If that fails, that should not necessarily rollback the in-memory state (Wt::Dbo did not make the changes to the

in-memory state, so it will not automatically undo them either on transaction failure). For example, it could be that you can recover from the failure in some other way and then retry the transaction commit.