Wt - Feature #6130

Impossible to access entity relation of an entity returned from a function | Dbo load(): no active transaction

11/13/2017 04:13 PM - Oleg Artenii

Status:	New	**Start date:**	11/13/2017
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:	4.0.0		

Description

main.cpp

```
#include <iostream>
#include <memory>
#include <Wt/Dbo/Dbo.h>
#include <Wt/Dbo/backend/Sqlite3.h>

using namespace std;
namespace dbo = Wt::Dbo;

class EntityA
{
public:
    string name {"Jef"};

    template<class Action>
    void persist(Action& a)
    {
        dbo::field(a, name, "name");
    }
};

class EntityB
{
public:
    string name {"Jef Jr."};
    dbo::ptr<EntityA> parent;

    template<class Action>
    void persist(Action& a)
    {
        dbo::field(a, name, "name");
        dbo::belongsTo(a, parent, "parent");
    }
};

dbo::ptr<EntityB> getAnyB(dbo::Session& session)
{
    dbo::Transaction t{session};

    return session.find<EntityB>().limit(1);
}

int main()
{
    dbo::Session session;

    session.setConnection([](){
        auto sqlite3 = make_unique<dbo::backend::Sqlite3>(":memory:");
        sqlite3->setProperty("show-queries", "true");
```

```
        return move(sqlite3);
    }());

    session.mapClass<EntityA>("entity_a");
    session.mapClass<EntityB>("entity_b");

    session.createTables();

    {
        dbo::Transaction t{session};

        auto b = make_unique<EntityB>();
        b->parent = session.add(make_unique<EntityA>());
        session.add(move(b));
    }

    //dbo::Transaction t{session}; // This "fixes" the problem, but it's VERY inconvenient
    cout << getAnyB(session)->parent->name; // Exception: Dbo load(): no active transaction

    return 0;
}
```

Output

```
begin transaction
create table "entity_a" (
  "id" integer primary key autoincrement,
  "version" integer not null,
  "name" text not null
)
create table "entity_b" (
  "id" integer primary key autoincrement,
  "version" integer not null,
  "name" text not null,
  "parent_id" bigint,
  constraint "fk_entity_b_parent" foreign key ("parent_id") references "entity_a" ("id") deferrabl
e initially deferred
)
commit transaction
begin transaction
insert into "entity_a" ("version", "name") values (?, ?)
insert into "entity_b" ("version", "name", "parent_id") values (?, ?, ?)
commit transaction
begin transaction
select "id", "version", "name", "parent_id" from "entity_b"  limit ?
commit transaction
terminate called after throwing an instance of 'Wt::Dbo::Exception'
  what():  Dbo load(): no active transaction
```

The instance is returned from a function and has relations lazy loaded when they are accessed, when it happens Dbo tries to execute a query and requires a transaction that doesn't exist anymore.

I suggest to create automatically a transaction if there is no an active one (especially for lazy loaded SELECT queries).

History

#1 - 11/13/2017 04:28 PM - Oleg Artenii

Or a way to tell find() to load all relations at once so the object can be safely used outside any transaction.

#2 - 11/15/2017 12:18 PM - Roel Standaert

- Tracker changed from Bug to Feature

- Priority changed from Urgent to Normal

You can always force load the parent using Session::load() when retrieving the entity.

Perhaps we could make it possible to transitively load related objects. I think we'll have to think very carefully about things like automatically creating transactions.

#3 - 11/15/2017 03:14 PM - Oleg Artenii

Any ETA?

So I should use Session::load() like this?

```
dbo::ptr<EntityB> getAnyB(dbo::Session& session)
{
    dbo::Transaction t{session};

    auto entity = session.find<EntityB>().limit(1).resultValue();

    session.load<EntityA>(entity->parent.id());

    return entity;
}
```

#4 - 11/15/2017 03:16 PM - Oleg Artenii

Is Session::load() recursive? If parent will also have some relations, will they be loaded or I need to call Session::load() for all nested relations?

#5 - 11/15/2017 04:32 PM - Roel Standaert

Yes, you can use Session::load() like that. Session::load() is not recursive.

#6 - 11/30/2017 10:55 AM - Oleg Artenii

I will "fix" this by encapsulating Wt::Dbo::ptr and the **getter** will **do the query and return std::unique_ptr**.

```
#include <iostream>
#include <memory>
#include <Wt/Dbo/Dbo.h>
#include <Wt/Dbo/backend/Sqlite3.h>

using namespace std;
namespace dbo = Wt::Dbo;

class EntityA
{
public:
    string name {"Jef"};

    template<class Action>
    void persist(Action& a)
    {
        dbo::field(a, name, "name");
    }
};

class EntityB
{
public:
    string name {"Jef Jr."};

    static shared_ptr<dbo::Session> session;
    static void setSession(shared_ptr<dbo::Session> s)
    {
        session = s;
    }

    template<class Action>
    void persist(Action& a)
    {
        dbo::field(a, name, "name");
        dbo::belongsTo(a, parent, "parent");
    }

    unique_ptr<EntityA> getParent() const
    {
        dbo::Transaction t{*session};
        return make_unique<EntityA>(*parent);
```

```
    }

    void setParent(dbo::ptr<EntityA> p)
    {
        parent = p;
    }

    void setParent(unique_ptr<EntityA> p)
    {
        parent.reset(move(p));
    }

private:
    dbo::ptr<EntityA> parent {nullptr};
};

shared_ptr<dbo::Session> EntityB::session = nullptr;

dbo::ptr<EntityB> getAnyB(shared_ptr<dbo::Session> session)
{
    dbo::Transaction t{*session};

    return session->find<EntityB>().limit(1);
}

int main()
{
    auto session = make_shared<dbo::Session>();

    session->setConnection([](){
        auto sqlite3 = make_unique<dbo::backend::Sqlite3>(":memory:");
        sqlite3->setProperty("show-queries", "true");

        return move(sqlite3);
    }());

    session->mapClass<EntityA>("entity_a");
    session->mapClass<EntityB>("entity_b");

    session->createTables();

    EntityB::setSession(session);

    {
        dbo::Transaction t{*session};

        auto b = make_unique<EntityB>();
        b->setParent(
            session->add(make_unique<EntityA>())
        );
        session->add(move(b));
    }

    cout << getAnyB(session)->getParent()->name;

    return 0;
}
```

DIFF https://www.diffchecker.com/LBdETr82

```
begin transaction
create table "entity_a" (
  "id" integer primary key autoincrement,
  "version" integer not null,
  "name" text not null
)
create table "entity_b" (
  "id" integer primary key autoincrement,
  "version" integer not null,
  "name" text not null,
  "parent_id" bigint,
  constraint "fk_entity_b_parent" foreign key ("parent_id") references "entity_a" ("id") deferrable initially
deferred
)
commit transaction
```

```
begin transaction
insert into "entity_a" ("version", "name") values (?, ?)
insert into "entity_b" ("version", "name", "parent_id") values (?, ?, ?)
commit transaction
begin transaction
select "id", "version", "name", "parent_id" from "entity_b"  limit ?
commit transaction
begin transaction
select "version", "name" from "entity_a" where "id" = ?
commit transaction
Jef
```

Benefits:

- No more bothering to create a Transaction before accessing a ptr member.
- No more bothering to call manually Session::load() in Model to populate all first level and nested ptr **
- The users of Entity will not know if it uses Wt::Dbo or any other ORM because getters will return std::unique_ptr

Disadvantages

- Every Entity must have access to Session instance. As static variable, I can't send it in constructor because of this https://redmine.emweb.be/issues/6076
- Entity getters will create a Transaction before accessing ptr to make sure the app doesn't fail if ptr is not loaded
- Entity getters will create a copy of ptr as unique_ptr and return it.

#7 - 11/30/2017 11:13 AM - Oleg Artenii

- ~~Every Entity must have access to Session instance~~
- ~~Entity getters will create a Transaction before accessing ptr~~

In my real app this is not the case because in Entity getter I will just call the model of EntityA to load and return the instance. So there will be no queries or transactions in Entities, all these will be in Models, and Entities will call Models. This breaks the Wt::Dbo::ptr sync feature but the benefits wins.

#8 - 12/26/2017 01:07 PM - Oleg Artenii

This issue is not urgent anymore.

We migrate from Wt::Dbo to libpqxx because we refactored the database layer and Wt::Dbo now is not used directly but it's deep in the abstraction, so all it's features are not so attractive anymore.

P.S. This issue with transactions is not present in libpqxx:

> Result objects can be kept around for as long as they are needed, **completely separate from the connections and transactions that originated them**. You can access the rows in a result using standard iterators, or more like an array using numerical indexes.
>
> http://pqxx.org/development/libpqxx/wiki/WikiStart