

Компоненты IBExpress

 visual-t.ru/ibexpress.html

IBX для Lazarus 1.8 [\[Скачать\]](#)

Тестовое приложение с IBX для Lazarus [\[Скачать\]](#)

Это модифицированная версия IBX, основные изменения - это поддержка 2х транзакций пишущей и читающей, добавлены дополнительные свойства, а т.ж. изменен доступ к API, TIBDatabase загружает клиентскую библиотеку в момент соединения с базой и освобождает при отсоединении, благодаря этому можно работать из одного приложения с серверами разных версий. Лицензия: Interbase Public Licence

Пример использования возвращаемых значений

Пример использования EXECUTE BLOCK

Использование отдельных транзакций

Сортировка TIBCustomDataSet

Дополнительные свойства TIBCustomDataSet

Пример использования возвращаемых значений из Insert, Update запросов, из EXECUTE BLOCK и EXECUTE PROCEDURE

Возвращаемые значения в IBX главным образом используются для механизма получения первичного ключа, который генерится на стороне сервера при вставке записи. В этом случае не нужно заботиться о генерации ключа на стороне клиента. Если имена возвращаемых параметров из TIBDataSet.QInsert, TIBDataSet.QModify совпадают с именами полей, то значения параметров подставляются в набор данных автоматически после выполнения метода TIBDataSet.Post. Пример:

TIBDataSet1.InsertSQL.Text:

```
INSERT INTO PEOPLE (LASTNAME, FIRSTNAME, THIRDNAME) VALUES  
(:LASTNAME, :FIRSTNAME, :THIRDNAME) RETURNING PEOPLE_ID
```

Таблица PEOPLE триггер BEFORE INSERT

```

CREATE OR ALTER TRIGGER PEOPLE_BI FOR PEOPLE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
IF (NEW.PEOPLE_ID IS NULL) THEN
NEW.PEOPLE_ID = GEN_ID(GEN_PEOPLE_ID,1);
END

```

```

IBDataSet1.Insert;
IBDataSet1.FN('LASTNAME').AsString := 'Иванов';
IBDataSet1.FN('FIRSTNAME').AsString := 'Иван';
IBDataSet1.FN('THIRDNAME').AsString := 'Иванович';
IBDataSet1.Post;

```

После выполнения метода Post сработает SQL запрос IBDataSet1.InsertSQL он в свою очередь производит вставку записи и возвращает значение первичного ключа, которое генерится в триггере BEFORE INSERT. После получения параметра 'PEOPLE_ID' из IBDataSet1.QInsert, значение будет подставлено в набор данных IBDataSet1, после этого автоматически срабатывает Refresh по первичному ключу.

Возвращаемые значение можно получать из EXECUTE BLOCK и EXECUTE PROCEDURE, в этом случае процедура или блок будут выглядеть примерно так:

```

EXECUTE BLOCK (LASTNAME VARCHAR(30) = :LASTNAME, FIRSTNAME
VARCHAR(30) = :FIRSTNAME, THIRDNAME VARCHAR(30) = :THIRDNAME)
RETURNS (PEOPLE_ID BIGINT)
AS
BEGIN
IF ((LASTNAME IS NULL) OR (LASTNAME = '')) THEN EXCEPTION ER_ERROR
'Не указана фамилия';
INSERT INTO PEOPLE (LASTNAME, FIRSTNAME, THIRDNAME) VALUES
(:LASTNAME, :FIRSTNAME, :THIRDNAME) RETURNING PEOPLE_ID INTO
:PEOPLE_ID;
SUSPEND;
END

```

Если объявлены возвращаемые значения в EXECUTE BLOCK, то команда SUSPEND обязательна. По умолчанию, компоненты могут принять только одну строку с возвращаемыми параметрами, если запрос из EXECUTE BLOCK возвращает более одной строки, возникнет исключение.

В случае с хранимой процедурой, оператор SUSPEND не нужен.

Если используется TIBQuery или TIBStoredProc, то для работы с возвращаемыми параметрами предусмотрены свойства:

TIBQuery.OutParams: TParams
TIBQuery.OutParamsCount: Integer
TIBQuery.OutParamByName(ParamName: string): TParam
TIBStoredProc.OutParamsCount: Integer
TIBStoredProc.OutParamByName(ParamName: string): TParam
TIBStoredProc.Params: TParams

У TIBStoredProc.Params указывает на список возвращаемые параметры, у TIBQuery для работы с возвращаемыми параметрами предусмотрен TIBQuery.OutParams.

Использование EXECUTE BLOCK

Инструкция EXECUTE BLOCK поддерживается в TIBDataSet.InsertSQL, TIBDataSet.ModifySQL, TIBDataSet.DeleteSQL, в TIBQuery, в TIBStoredProc.

IBQuery1.SQL:

```
EXECUTE BLOCK (INPARAM1 INTEGER = :INP1)
RETURNS (OUTPARAM VARCHAR(50))
AS
BEGIN
  OUTPARAM = 'Выход ' || INPARAM1;
SUSPEND;
END
```

Если объявлены возвращаемые значения в EXECUTE BLOCK, то команда SUSPEND обязательна. По умолчанию, компоненты могут принять только одну строку с возвращаемыми параметрами, если запрос из EXECUTE BLOCK возвращает более одной строки, возникнет исключение. Если EXECUTE BLOCK требуется использовать для получения набора данных, то для TIBSQL в котором используется запрос, нужно установить TIBSQL.SelectOnBlock := True, в TIBDataSet для QSelect и QRefresh свойство SelectOnBlock установлено в True по умолчанию, поэтому для TIBDataSet в SelectSQL EXECUTE BLOCK может возвращать любое количество записей

Пример выполнения вышеприведенного SQL запроса с EXECUTE BLOCK через TIBQuery:

```
IBQuery1.PN('INP1').AsInteger := 100;
IBQuery1.ExecSQL;
if IBQuery1.OutParamCount > 0 then
  ShowMessage(IBQuery1.OutParamByName('OUTPARAM').AsString);
```

Инструкция EXECUTE BLOCK поддерживается в компоненте TIBStoredProc, через свойство SQL, так-же как и в TIBQuery.

Использование отдельных транзакций

Компонент TIBDataSet использует для своей работы 2 транзакции: Transaction и UpdateTransaction. Смысл использования отдельных транзакций - максимально исключить возможность получения во время работы мертвых блокировок (DEAD LOCK). Исследования по использованию оптимальных параметров отдельных транзакций проводила компания Devrice на FIBPlus, я могу лишь подтвердить, что используя их рекомендации, у меня проблем не было.

Запросы из TIBDataSet.QSelect (и QRefresh в большинстве случаев) работают через читающую транзакцию (Transaction). Для читающей транзакции могу рекомендовать параметры (в редакторе компонента TIBTransaction соответствует пункту ReadCommitted)

read, read_committed, rec_version, nowait

С этими параметрами транзакция стартует в режиме ReadOnly и может жить весьма долго не расходуя ресурсы сервера. Но есть одно но, если используются текстовые блобы и используется операция конкатенации текстовых блобов, это может привести к чрезмерному потреблению ресурсов даже в режиме ReadOnly, поэтому при работе с блобами, лучше использовать отдельную транзакцию, которую не стоит держать долго открытой и закрывать сразу после получения данных с сервера.

Модифицирующие запросы из QInsert, QModify, QDelete выполняются через UpdateTransaction. Для пишущей транзакции могу рекомендовать параметры (в редакторе компонента TIBTransaction соответствует пункту FBWrite):

write, wait, no_rec_version, read_committed, lock_timeout=10

В TIBDataSet, после метода Post автоматически стартует Refresh, выполняя SQL запрос из TIBDataSet.RefreshSQL. При выполнении этого запроса, QRefresh сам выбирает себе транзакцию. В случае, если на момент старта Refresh запроса UpdateTransaction активна, такое может быть при AutoCommit = False, т.е. ручное управление транзакциями, то предполагается, раз транзакция модифицирующего запроса не подтверждена, то изменения сделанные запросом будут видны лишь через модифицирующую транзакцию, поэтому Refresh пройдет через UpdateTransaction, во всех других случаях будет использована Transaction.

Обычно в приложении для TIBDataSet с AutoCommit = True достаточно использовать 1 читающую и 1 пишущую транзакцию. Читающую транзакцию можно открыть при старте приложения вручную (в этом случае AutoCommit для неё работать не будет) и закрывать её то-же руками, когда приложение завершает работу, в этом случае, когда наборы данных приходится много раз закрывать и открывать не будут сбрасываться значения уже установленных параметров в TIBDataSet. Но если вы используете блобы, лучше сделать для них отдельную читающую транзакцию, время жизни которой лучше ограничить временем

обращения к серверу для выполнения читающего запроса, по причине описанной выше.

У компонента TIBTransaction установка свойства DefaultDatabase обязательно. Если Вы создаете проект с использованием IBX, в котором подключение к базе проходит нормально, а датасет не открывается, у транзакций проверьте свойство DefaultDatabase.

Сортировка TIBCustomDataSet

Для сортировки данных в TIBCustomDataSet введено public свойство OrderFields. Это свойство не осуществляет сортировку локального кэша, а изменяет или добавляет SQL инструкцию ORDER BY к SELECT запросу. Можно использовать ASC и DESC после имен полей, несколько значений перечисляются через запятую.

```
IBDataSet.PN('GOD').AsInteger := 2015;  
IBDataSet.Open;  
IBDataSet1.OrderFields := 'GROUPNAME, SUBGROUPNAME DESC';
```

Дополнительные свойства TIBCustomDataSet

TIBCustomDataSet.AutoCancel: Boolean - работает совместно с AutoCommit и если AutoCommit := True и AutoCancel := True и когда в наборе данных выполняется отмена вставки или редактирования строки TIBDataSet.Cancel, то происходит проверка UpdateTransaction и если транзакция активна, такое могло произойти если во время операции TIBDataSet.Post произошла ошибка, то транзакция автоматически завершается.

TIBCustomDataSet.DefFormats - свойство в виде структуры которая задает форматы для отображения дат и чисел.

TIBCustomDataSet.DefValueFormServer: Boolean - если True то перед вставкой новой записи извлекает значения по умолчанию для полей заданные при создании таблицы и подставляет их в новую запись.

TIBCustomDataSet.DetailConditions - параметры задают поведение для связки Master - Detail. Если назначен набор данных Master через свойство DataSource то:

TIBCustomDataSet.DetailConditions.dcForceMasterPost - если True, то перед методом Post проверяет набор данных Master и если он в режиме dsInsert или dsEdit, то выполняет ему Post.

TIBCustomDataSet.DetailConditions.dcForceMasterRefresh - если True, то после метода Post вызывает у набора данных Master.Refresh. Метод обработки события Refresh у мастера переделан и если после Master.Refresh строка не изменилась (параметры по которым происходит связка с Detail), то набор данных Detail не переоткрывается.

TIBCustomDataSet.DetailConditions.dcForceOpenClose - если True, то набор данных автоматически открывается и закрывается при открытии и закрытии набора данных Master

TIBCustomDataSet.DetailConditions.FieldsFormats - позволяет задать персональные форматы отображения для отдельных полей

TIBCustomDataSet.DetailConditions.EditFormats - позволяет задать персональные форматы редактирования для отдельных полей