**Перенос приложений MIDAS с одной СУБД на другую**

Александр Капустин

**Введение**

В данной статье рассматриваются проблемы, связанные с миграцией приложения MIDAS с одной СУБД на другую. Рассмотрим это на примере переноса приложения, описанного в статье Романа Игнатьева "MIDAS: практика применения". Приложение написано под Interbase 5.6 и использует компоненты IBX на сервере приложений для доступа к СУБД. Перепишем его таким образом, чтобы приложение смогло работать под управлением MSSQL Server 7.0 и MSSQL Server 2000 (при помощи небольших переделок скрипта можно добиться работы приложения под Sybase ASE 12.0). Следует также заметить, что переделке подвергнутся только скрипт СУБД и сервер приложений. Клиентская часть остается нетронутой, т.к. при использовании многозвенной архитектуры она абсолютно изолирована от деталей реализации серверной части.

Некоторые замечания к содержанию статьи.

Предполагается, что читателю уже знакомы (хотя бы в начальной стадии) синтаксис SQL (в приложении либо к Interbase, либо к MSSQL, а также общие принципы работы с БД из Delphi (в статье используются IBX&ADO, но это не единственно возможное решение).

Процесс переноса происходит уже после того, как приложение отлажено и стабильно работает (т.к. параллельная разработка для нескольких СУБД – немного другой случай, и его мы рассматривать не будем).

Хотелось бы сказать несколько слов о том, зачем это вообще нужно, что мы приобретаем, и что теряем (немного теории).

Приобретается в основном снижение стоимости программного продукта, ибо если у клиента уже установлена хотя бы одна из поддерживаемых вами СУБД, нет необходимости тратить большие деньги на закупку СУБД, нового сервера и настройку всего этого. Улучшаются возможности интеграции с существующими системами.

Но при этом мы перестаем использовать на 100% возможности какой-либо отдельной СУБД. Следует различать два случая. Первый случай, когда поддерживается совместимость со старыми версиями этой же СУБД (например, поддерживается линейка MSSQL 6.5 – MSSQL2000). В этом случае мы ограничены возможностями самой слабой версии, и не можем использовать нововведения. Второй случай, гораздо более тяжелый и вместе с тем интересный для рассмотрения, когда планируется совместимость между различными СУБД, например между MSSQL и Interbase. Должен сразу оговориться, что этот случай встречается гораздо реже, но если вы при проектировании приложения не будете учитывать эту возможность, то переход вызовет гораздо больше сложностей.

Следует отметить, что при переносе двухуровневого приложения проблем возникнет гораздо больше, т.к. большая часть бизнес-логики находится на сервере, и если синтаксис СУБД сильно отличается, возможности переноса сильно ограничиваются. В случае же трехуровневого приложения большинство задач, связанных с логикой, решает сервер приложений (хочется заметить, что это справедливо только для правильно спроектированного приложения).

Модификация структуры БД

К сожалению, перенос структуры БД "один-в-один" между различными СУБД практически невозможен. Здесь приведено описание некоторых проблем, с которыми придется столкнуться при переносе, а также возможные пути их решения. Лучше всего, если бы эти вещи были учтены изначально при проектировании исходного приложения, т.к. в этом случае объем работ при переносе приложения сокращается.

Добавление записей в таблицу

Для абстрагирования метода генерации уникальных идентификаторов для каждой записи можно вынести его в хранимую процедуру, которая будет возвращать ID новой записи. Это позволяет легко добавлять записи в подчиненную таблицу, не производя никаких дополнительных манипуляций. В дальнейшем вы можете возложить на эту процедуру, например, генерацию идентификаторов в заданном диапазоне, или обеспечить сквозную нумерацию. Для хранения идентификаторов проще всего иметь отдельную таблицу примерно следующего вида:

|  |
| --- |
| create table Seeds (  TableName varchar(30), --имя таблицы  ID int, --ID последней вставленной записи в данную таблицу  LowOffset int --нижняя граница диапазона  )  go |

При добавлении пользовательских таблиц необходимо не забывать вставлять в эту таблицу соответствующие записи. Ниже приведен пример SQL-запроса, делающего это:

|  |
| --- |
| insert into Seeds(TableName, ID, LowOffset, HiOffset)  values('MyCoolTable', 0, 0, 1000000)  go |

Текст процедуры в простейшем случае будет выглядеть так:

|  |
| --- |
| create procedure CLIENT\_ID  @TableName varchar(30),  @ID int output  as  update Seeds  set ID = ID + 1,  @ID = ID + LowOffset  where TableName = @TableName  go |

При обновлении (update) таблицы накладывается блокировка изменения, которая не позволит другому клиенту выполнить эту же процедуру одновременно с первым. Помимо вышеперечисленного такой подход упрощает жизнь при необходимости репликации данных между филиалами. Тогда в каждом филиале настраивается свой диапазон, и первичные ключи гарантированно не будут пересекаться.

Контроль целостности данных

При проектировании базы необходимо учесть следующие ограничения:

Каскадное изменение по foreign key появилось только в MSSQL2000. Так что если задаться целью сохранить совместимость с предыдущими версиями (а также с Sybase), каскадные изменения необходимо производить при помощи хранимых процедур (почему не использовать триггеры, сказано ниже).

Триггеры, отрабатывающие не после проверки всех ограничений целостности, а вместо действия, на которое их вызвали, также появились только в MSSQL2000. В более ранних версиях они просто не смогли бы отработать каскадное изменение при наличии foreign key. Также при написании триггеров следует учесть особенности реализации для каждой СУБД. Так, например, в Interbase триггер отрабатывает на каждую запись, а в MSSQL – на изменение, вставку или удаление записи. Как вариант, можно отказаться от поддержки целостности, основанной на foreign key, и реализовать ее полностью на триггерах.

Перенос скрипта

Здесь приведены основные трудности, с которыми можно столкнуться при переносе скрипта Interbase на MSSQL (должен еще раз повториться, что статья не претендует на полный и детальный разбор отличий между этими СУБД, да такой анализ и не может быть полностью корректным).

Соответствие встроенных типов

Основные различия, которые следует учитывать при переносе скрипта:

|  |  |  |
| --- | --- | --- |
| IB | MSSQL | Комментарий |
| char | char | -в MSSQL – не более 8000, в IB - не более 32767 char |
| varchar | varchar | -в MSSQL - не более 8000, в IB - не более 32767 char |
| blob | text, image |  |
| date | datetime, smalldatetime | (последний обрезает время до минут) |
|  |  |  |
|  | money, smallmoney | - в IB нет аналогов |
|  | bit | - в IB нет аналогов |

В MSSQL нет типов, представляющих только дату или только время, имеющихся в IB6.

Домены

В IB для создания доменов используется следующая конструкция:

|  |
| --- |
| create domain DCount numeric(15,4) default 1 not null; |

Для MSSQL это будет выглядеть следующим образом:

|  |
| --- |
| create default ONE as 1  go  exec sp\_addtype 'DCount', 'numeric(15,4)', 'NOT NULL'  go  exec sp\_bindefault 'ONE', 'DCount'  go |

Таблицы

Переносятся без проблем, следует только обратить внимание на замечания по контролю целостности данных (кстати, обратное преобразование будет затруднено, если вы будете использовать специфические для MSSQL типы (особенно для MSSQL2000)).

Хранимые процедуры

Перенос хранимых процедур – это наиболее трудоемкий процесс, т.к. придется переписывать все целиком. Но в правильно спроектированном трехзвенном приложении роль ХП должна быть сведена к минимуму. Основные трудности возникают при переводе ХП, возвращающих результирующий набор. Часть из них (не содержащие сложной бизнес-логики) может быть переведена в разряд представлений (view). Для остальных можно либо создавать временные таблицы на уровне соединения с СУБД, либо создавать постоянные таблицы и разграничивать данные в них по идентификатору подключения (SPID) (но тогда не забывайте их чистить :)). Если же вы решите ограничиться только MSSQL2000, то можете использовать тип "таблица" для возврата набора значений из процедуры. Рассмотрим несколько примеров перевода ХП. Процедура отчета о взаиморасчетах между клиентами:

|  |
| --- |
| create procedure REP\_INOUT(FROM\_DATE date, TO\_DATE date)  returns (FROM\_ID integer, FROM\_NAME varchar(180), TO\_ID integer,  TO\_NAME varchar(180), FULL\_SUM numeric(15,4))  as  begin  for select FROM\_ID, TO\_ID, sum(DOC\_SUM)  from DOC\_TITLE  where DOC\_DATE >= :FROM\_DATE and DOC\_DATE <= :TO\_DATE  group by FROM\_ID, TO\_ID  into :FROM\_ID, :TO\_ID, :FULL\_SUM  do begin  FROM\_NAME = NULL;  TO\_NAME = NULL;  select NAME  from client  where CLIENT\_ID = :FROM\_ID  into :FROM\_NAME;  select NAME  from client  where CLIENT\_ID = :TO\_ID  into :TO\_NAME;  if (FULL\_SUM is NULL) then  FULL\_SUM = 0;  suspend;  end  end  ^ |

Преобразуется в процедуру следующего вида:

|  |
| --- |
| create procedure rep\_inout @from\_date smalldatetime, @to\_date smalldatetime  as  select dt.from\_id, dt.to\_id, isnull(sum(dt.doc\_sum), 0) as full\_sum,  c.name as from\_name, c1.name as to\_name  from doc\_title dt,  client c,  client c1  where dt.doc\_date >= @from\_date  and dt.doc\_date <= @to\_date  and c.client\_id = dt.from\_id  and c1.client\_id = dt.to\_id  group by dt.from\_id, c.name, dt.to\_id, c1.name  go |

Следующий пример. Процедура выводит список документов и полные имена клиентов:

|  |
| --- |
| create procedure LIST\_DOC (FROM\_DATE date, TO\_DATE date)  returns (DOC\_ID integer, DOC\_NUM varchar(40), DOC\_DATE date,  FROM\_ID integer, TO\_ID integer, FROM\_NAME varchar(224),  TO\_NAME varchar(224), DOC\_SUM numeric(15,4))  as  begin  for select DOC\_ID, DOC\_NUM, DOC\_DATE, FROM\_ID, TO\_ID, DOC\_SUM  from DOC\_TITLE  where DOC\_DATE >= :FROM\_DATE and DOC\_DATE <= :TO\_DATE  into :DOC\_ID, :DOC\_NUM, :DOC\_DATE, :FROM\_ID, :TO\_ID, :DOC\_SUM  do begin  FROM\_NAME = NULL;  TO\_NAME = NULL;  execute procedure CLIENT\_FULL\_NAME (:FROM\_ID)  returning\_values :FROM\_NAME;  execute procedure CLIENT\_FULL\_NAME (:TO\_ID)  returning\_values :TO\_NAME;  suspend;  end  end  ^ |

На примере перевода данной процедуры покажем один из вариантов того, как можно свести к минимуму количество блокировок на часто используемой таблице.

Создадим для начала вспомогательную таблицу следующего вида:

|  |
| --- |
| create table pDoc\_List  (  SPID int, --идентификатор подключения  doc\_id int,  doc\_num varchar(40),  doc\_date smalldatetime,  from\_id int,  to\_id int,  doc\_sum DSum,  from\_name varchar(224),  to\_name varchar(224)  )  go |

В этой временной таблице мы будем хранить данные, отвечающие нашим критериям поиска. Для того, чтобы можно было отличить, какому клиенту предназначены данные, вводится столбец SPID, в котором мы будем хранить уникальный идентификатор подключения к БД (@@spid).

После того, как данные скопированы в эту таблицу, мы можем спокойно, никому не мешая, обрабатывать их так, как нам захочется. А клиенту (точнее, серверу приложений) остается только их выбрать из данной таблицы.

|  |
| --- |
| ПРИМЕЧАНИЕ  Надо отметить, что данный алгоритм применим только в тех случаях, где некритично, что между перечитыванием данных клиентом они могут измениться |

Вот код процедуры, заполняющей эту таблицу:

|  |
| --- |
| create proc list\_doc @from\_date datetime, @to\_date datetime  as  declare @from\_name varchar(224)  declare @to\_name varchar(224)  declare @from\_id int  declare @to\_id int  declare @doc\_id int  declare @doc\_num varchar(40)  declare @doc\_date datetime  declare @doc\_sum dsum  delete from pDoc\_List  where SPID = @@spid --очищаем временную таблицу от предыдущих данных  --вставляем нужные записи  insert into pDoc\_List(SPID,  doc\_id,  doc\_num,  doc\_date,  from\_id,  to\_id,  doc\_sum,  from\_name,  to\_name  )  select @@spid,  doc\_id,  doc\_num,  doc\_date,  from\_id,  to\_id,  doc\_sum,  '',  ''  from doc\_title  where doc\_date >= @from\_date and doc\_date <= @to\_date    --создаем наиболее быстрый курсор для обработки записей  declare list\_docs insensitive cursor for  select doc\_id, from\_id, to\_id  from pDoc\_List  where SPID = @@spid  for read only  open list\_docs  fetch next from list\_docs  into @doc\_id, @from\_id, @to\_id  while @@fetch\_status = 0  begin  select @from\_name = '', @to\_name = ''  exec client\_full\_name @from\_id, @from\_name output  exec client\_full\_name @to\_id, @to\_name output  --заполняем поля, которых нет в основной таблице  update pDoc\_List  set from\_name = @from\_name,  to\_name = @to\_name  where SPID = @@spid  and doc\_id = @doc\_id    fetch next from list\_docs  into @doc\_id, @from\_id, @to\_id  end  close list\_docs  deallocate list\_docs  go |

SQL-запрос, исполняемый на сервере приложений для передачи данных клиенту:

|  |
| --- |
| exec list\_doc @from\_date = :from\_date, @to\_date = :to\_date  select \* from pDoc\_List where SPID = @@spid |
| ПРИМЕЧАНИЕ  При написании ХП следует обратить особое внимание на одновременную работу нескольких пользователей. Необходимо минимизировать влияние "тяжелых" (отчетных) процедур на работу клиентов (один из вариантов был показан выше). | | |

Триггеры

Тут ничего сложного нет, необходимо только помнить, что триггеры в MSSQL запускаются только после действия (есть еще вместо (instead of), но это только в MSSQL2000).

Пример: предотвращение удаления клиента, если существуют документы с его участием (чтобы такой триггер не конфликтовал с ограничением ссылочной целостности, в MSSQL необходимо убрать foreign key с таблицы doc\_title на client)

|  |
| --- |
| create trigger CLIENT\_BEFORE\_DELETE for CLIENT  before delete  as  begin  if (exists (select \* from DOC\_TITLE  where FROM\_ID = OLD.CLIENT\_ID))  then  exception EX\_CLIENT\_IN\_DOC;  if (exists (select \* from DOC\_TITLE  where TO\_ID = OLD.CLIENT\_ID))  then  exception EX\_CLIENT\_IN\_DOC;  end  ^ |

Преобразуется в

|  |
| --- |
| create trigger CLIENT\_AFTER\_DELETE on CLIENT  for delete  as  if (exists (select d.CLIENT\_ID from  DOC\_TITLE dt, deleted d  where dt.FROM\_ID = d.CLIENT\_ID))  begin  --чтобы сообщение было видно на клиенте  raiserror ('Существует запись в документе', 16, 1)  --необходимо ручками откатить транзакцию  rollback transaction  end  if (exists (select d.CLIENT\_ID from  DOC\_TITLE dt, deleted d  where dt.TO\_ID = d.CLIENT\_ID))  begin  raiserror ('Существует запись в документе', 16, 1)  rollback transaction  end  go |

Модификация сервера приложений.

Здесь основная часть переработки связана с переходом от IBX (InterBase Express) к ADO (ActiveX Data Object). Основные вещи, на которые следует обратить внимание:

Реализация транзакций на клиенте – в IBX это отдельный компонент, в ADO такая функциональность предоставляется методами TADOConnection. Еще небольшая рекомендация – аккуратно подходите к выбору уровня изоляции транзакций (чем меньше уровень изоляции, тем быстрее будет работать приложение).

Чтобы клиент работал без переделки с разными источниками данных, необходимо, чтобы типы данных полей совпадали. Например, в IBX для numeric(15, 4) по умолчанию подставляется TFloatField, а в ADO – TBCDField. Это единственное отличие, которое мне встретилось при переносе (но это не значит, что их вообще нет). Проблема решилась ручной установкой данного типа поля в TCurrencyField.

Перевод sql-выражений из синтаксиса IB в MSSQL.

Отличия, связанные с различиями структуры БД. Например, если будет реализовано каскадное удаление с помощью ХП, то придется реализовывать эту логику внутри сервера, чтобы оставить клиента нетронутым.

В качестве примера приведем перевод одной из процедур сервера приложений:

|  |
| --- |
| //Описание того, что делает данная процедура, читайте в статье Игнатьева.  //Код, работающий с IBX  function TrdmDoc.ApplyChanges: WideString;  begin  lock;  try  FLastUpdateErrors := '';  if FState = osInactive then  raise Exception.Create('Документ не был создан либо открыт');  with cdsTitle do  begin  Edit;  FieldByName('DOC\_SUM').asCurrency := CalcSum;  Post;  end;  ibtDoc.StartTransaction; //ibtDoc – компонент транзакции  if FState = osInsert then  begin  if cdsTitle.ChangeCount > 0 then  cdsTitle.ApplyUpdates(-1);  if cdsBody.ChangeCount > 0 then  cdsBody.ApplyUpdates(-1);  end;  if FState = osUpdate then  begin  if cdsBody.ChangeCount > 0 then  cdsBody.ApplyUpdates(-1);  if cdsTitle.ChangeCount > 0 then  cdsTitle.ApplyUpdates(-1);  end;  Result := FLastUpdateErrors;  if Result = '' then  ibtDoc.Commit  else  begin  ibtDoc.Rollback;  end;  finally  ibtDoc.Active := False; //DefaultAction = Rollback  unlock;  end;  end;  //Код, работающий с ADO  function TrdmDoc.ApplyChanges: WideString;  begin  lock;  try  FLastUpdateErrors := '';  if FState = osInactive then  raise Exception.Create('Документ не был создан либо открыт');  with cdsTitle do  begin  Edit;  FieldByName('DOC\_SUM').asCurrency := CalcSum;  Post;  end;  adcDocs.BeginTrans; //явные транзакции задаются на уровне соединения  if FState = osInsert then //а не отдельным компонентом  begin  if cdsTitle.ChangeCount > 0 then  cdsTitle.ApplyUpdates(-1);  if cdsBody.ChangeCount > 0 then  cdsBody.ApplyUpdates(-1);  end;  if FState = osUpdate then  begin  if cdsBody.ChangeCount > 0 then  cdsBody.ApplyUpdates(-1);  if cdsTitle.ChangeCount > 0 then  cdsTitle.ApplyUpdates(-1);  end;  Result := FLastUpdateErrors;  if Result = '' then  adcDocs.CommitTrans  else  begin  adcDocs.RollbackTrans;  end;  finally  unlock;  end;  end; |

**Заключение**

Это всего лишь пример. В реальных приложениях следует более тщательно продумывать перенос приложений. Например, переписывать лучше не весь сервер приложений, а только зависимый от источника данных код, вынося его в отдельные модули.

Данная статья не претендует на полноту освещения данного вопроса, а также и автор при изложении подходов для решения проблем не претендует на роль "истины в последней инстанции". Здесь был изложен лишь минимум сведений, необходимый для решения поставленной задачи, а также некоторые размышления, которые могут помочь при решении схожих проблем.

Все вопросы, замечания, исправления, дополнения направляйте на kapusto@mail.ru

Хочу выразить признательность Игнатьеву Роману, Павлу Шмакову за советы, критику и настойчивость.