Глава 17

Уровень базы данных

В этой главе

- Введение
- Временные таблицы
- Суррогатные ключи
- Альтернативные ключи
- Связи между таблицами
- Наследование таблиц
- Единица работы
- Инфраструктура контроля даты вступления в силу
- Поддержка запросов полнотекстового поиска
- Прикладной программный интерфейс QueryFilter
- Разделы данных

Введение

Среда времени выполнения Microsoft Dynamics AX 2012 предоставляет набор надежных возможностей по работе с базами данных, которые намного упрощают создание ERP-приложения. В Microsoft Dynamics AX 2012 добавлена поддержка многих новых мощных возможностей по работе с базами данных; в этой главе мы подробно расскажем о некоторых из них. Вы познакомитесь с этими возможностями, узнаете, как использовать их в приложении и там, где это необходимо, а также о том, как каждая из них работает.

Многие возможности базы данных, такие как оптимистичное параллельное управление (optimistic concurrency control, OCC), поддержка транзакций и система выполнения запросов, были доступны в Microsoft Dynamics AX на протяжении нескольких выпусков продукта. Более подробную информацию об этих и других возможностях работы с базами данных в Microsoft Dynamics AX вы можете найти в разделе «Database» в Microsoft

Dynamics AX 2012 SDK по адресу: *http://msdn.microsoft.com/en-us/library/aa588039.aspx*. Вы также можете обратиться к предыдущим изданиям этой книги, содержащим полезную информацию о функциональности работы с базами данных, которая все еще применима к Microsoft Dynamics AX 2012.

Временные таблицы

По умолчанию любая таблица, определенная в AOT, отображается в связке 1-к-1 на постоянную таблицу в используемой реляционной базе данных. Microsoft Dynamics AX также поддерживает функциональность временных таблиц. В предыдущих версиях программы была возможность создавать временные таблицы типа *InMemory,* которые отображались на таблицу, использующую архитектуру индексно-последовательного метода доступа (indexed sequential access method, ISAM) на основе файла и доступную только на время работы сервера AOS или клиента. Microsoft Dynamics AX 2012 предоставляет новый тип временных таблиц, хранящихся в базе TempDB в Microsoft SQL Server.

Временные таблицы типа *InMemory*

Файл ISAM, представляющий временную таблицу типа *InMemory*, содержит данные и все индексы, которые определены на этой таблице в AOT. За счет того, что работа с небольшими наборами данных обычно быстрее, чем с большими, среда времени выполнения Microsoft Dynamics AX отслеживает размер каждой временной таблицы типа *InMemory*. Если этот размер не превышает 128 кбайт, временная таблица остается в оперативной памяти. Если же размер превышает 128 кбайт, то временная таблица записывается в физический файл ISAM. Переключение с оперативной памяти на физический файл существенно сказывается на производительности. Когда данные перебрасываются из памяти в физический файл, то создается файл с названием наподобие *$tmp<8 цифр>.$$$*. Вы можете отслеживать превышение порогового размера по тому, когда появляется этот файл.

Хотя временные таблицы типа *InMemory* не связаны с таблицами в реляционной базе данных, все операторы манипуляции данными (DML-операторы) в X++ корректно работают для этого типа таблиц. Однако среда времени выполнения Microsoft Dynamics AX некоторые операторы выполняет в режиме понижения до работы на уровне записей, потому что файлы ISAM предлагают более ограниченный уровень функциональности, чем реляционные базы данных.

Использование временных таблиц типа *InMemory*

Когда вы объявляете табличную переменную для временной таблицы типа *InMemory*, таблица не будет содержать записей, поэтому для работы с временной таблицей необходимо предварительно вставить в нее записи. Сама временная таблица типа *InMemory* и все ее записи будут утеряны, когда в памяти не останется объявленных табличных переменных, указывающих на этот временный набор данных. Для временной таблицы типа *InMemory* память и файловое пространство не выделяются до тех пор, пока не будет вставлена первая запись, при этом таблица размещается на том уровне, на котором была вставлена первая запись. Например, если первая запись была создана на уровне сервера, то память выделяется на сервере приложения и при необходимости файл также создается на сервере.

> **Важно.** Используйте временные таблицы с осторожностью, уделяя внимание тому, чтобы они не стали причиной увеличения числа циклов приема-передачи между клиентом и сервером, поскольку это ведет к падению производительности. Более подробную информацию вы можете найти в главе 13.

Объявленная табличная переменная временной таблицы содержит указатель на набор данных. Если используются две табличные переменные временной таблицы, то они по умолчанию указывают на два различных набора данных, даже если тип таблицы у них один и тот же. Чтобы проиллюстрировать это, код X++ в следующем примере использует временную таблицу *TmpLedgerTable*, определенную в Microsoft Dynamics AX 2012. Таблица содержит четыре поля: *AccountName*, *AccountNum*, *CompanyId* и *LedgerDimension*. Поля *AccountNum* и *CompanyId* являются частью уникального индекса *AccountNumIdx*, как показано на рис. 17-1.

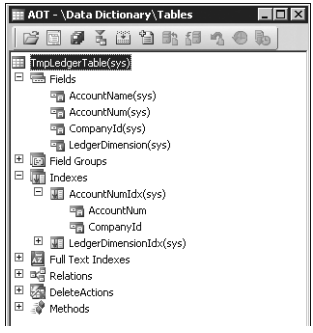


Рис. 17–1. Временная таблица *TmpLedgerTable*

В следующем коде X++ показаны два буфера записи одного и того же типа и то, как одна и та же запись может быть вставлена в оба буфера записей. Поскольку эти буферы указывают на два различных набора данных, то ошибки «запись уже существует» не произойдет. Сбой случился бы, если бы оба буфера записей указывали на один и тот же временный набор данных или же отображались на реальную таблицу в базе данных.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable1;
    TmpLedgerTable tmpLedgerTable2;

    tmpLedgerTable1.CompanyId = 'dat';
    tmpledgerTable1.AccountNum = '1000';
    tmpLedgerTable1.AccountName = 'Name';
    tmpLedgerTable1.insert(); // вставка в набор данных буфера tmpLedgerTable1.

    tmpLedgerTable2.CompanyId = 'dat';
    tmpledgerTable2.AccountNum = '1000';
    tmpLedgerTable2.AccountName = 'Name';
    tmpLedgerTable2.insert(); // вставка в набор данных буфера tmpLedgerTable2.
}
```

Чтобы табличные буферы использовали один и тот же временный набор данных, необходимо вызвать метод табличного буфера *setTmpData*, как показано в следующем примере кода X++. Здесь метод *setTmpData* вызывается на втором буфере, и ему передается в качестве параметра первый табличный буфер.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable1;
    TmpLedgerTable tmpLedgerTable2;

    tmpLedgerTable2.setTmpData(tmpLedgerTable1);

    tmpLedgerTable1.CompanyId = 'dat';
    tmpledgerTable1.AccountNum = '1000';
    tmpLedgerTable1.AccountName = 'Name';
    tmpLedgerTable1.insert(); // вставка в общий набор данных.

    tmpLedgerTable2.CompanyId = 'dat';
    tmpledgerTable2.AccountNum = '1000';
    tmpLedgerTable2.AccountName = 'Name';
```

```
    tmpLedgerTable2.insert(); // вставка приведет к ошибке из-за нарушения уникальности
индекса.
}
```

Предыдущий пример кода X++ при второй вставке приведет к ошибке «запись уже существует», поскольку оба табличных буфера указывают на один и тот же набор данных. Вы получите похожее поведение, если вместо вызова метода *setTmpData* просто присвоите второй табличный буфер первому, как показано в следующем примере.

```
tmpLedgerTable2 = tmpLedgerTable1;
```

Однако в данном случае переменные будут указывать на один и тот же объект, а это означает, что они будут использовать один и тот же набор данных.

Когда необходимо использовать метод *data* для копирования данных из одного табличного буфера в другой, в случае, когда оба буфера указывают на один и тот же набор данных, вы должны написать код, как показано в следующем примере.

```
tmpLedgerTable2.data(tmpLedgerTable1);
```

Предупреждение. Связь между двумя табличными буферами и одним набором данных будет утеряна, если просто написать *tmpLedgerTable2 = tmpLedgerTable1.data()*. В этом случае табличный буфер *tmpLedgerTable2* будет указывать на новый табличный буфер, который связан уже с другим набором данных.

Как упоминалось ранее, если ни один табличный буфер не указывает на набор данных, то записи во временной таблице теряются, выделенная память освобождается и файл удаляется. Такое поведение иллюстрируется в следующем примере кода X++, где одна и та же запись вставляется дважды, используя один и тот же табличный буфер. Однако поскольку табличному буферу между двумя вставками присваивается *null*, то первый набор данных теряется и вторая вставка не приводит к нарушению уникальности индекса, так как новая запись вставляется уже в новый набор данных.

```
static void TmpLedgerTable(Args _args)
{
    TmpLedgerTable tmpLedgerTable;
```

```
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // вставка в первый набор данных.

    tmpLedgerTable = null; // выделенная память освобождается и файл удаляется.

    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // вставка в новый набор данных.
}
```

Обратите внимание, что ни в одном из примеров работы с временными таблицами типа *InMemory* не используются операторы *ttsbegin*, *ttscommit* и *ttsabort*. Эти операторы влияют только на обычные таблицы, хранящиеся в реляционной базе данных. К примеру, следующий код X++ добавляет данные во временную таблицу типа *InMemory*. Из-за типа таблицы значение поля *accountNum* будет выведено в Infolog несмотря на то, что был выполнен оператор *ttsabort*.

```
static void TmpLedgerTableAbort(Args _args)
{
    TmpLedgerTable tmpLedgerTable;

    ttsbegin;
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // вставка в таблицу.
    ttsabort;

    while select tmpLedgerTable
    {
        info(tmpLedgerTable.AccountNum);
    }
}
```

Чтобы успешно отменить операции вставки в таблицу в предыдущем примере, необходимо вызывать методы *ttsbegin* и *ttsabort* на самом временном табличном буфере, как показано ниже.

```
static void TmpLedgerTableAbort(Args _args)
{
    TmpLedgerTable tmpLedgerTable;

    tmpLedgerTable.ttsbegin();
    tmpLedgerTable.CompanyId = 'dat';
    tmpledgerTable.AccountNum = '1000';
    tmpLedgerTable.AccountName = 'Name';
    tmpLedgerTable.insert(); // вставка в таблицу.
    tmpLedgerTable.ttsabort();

    while select tmpLedgerTable
    {
        info(tmpLedgerTable.AccountNum);
    }
}
```

В случае работы с несколькими временными табличными буферами необходимо вызывать метод *ttsbegin*, *ttscommit* и *ttsabort* на каждом табличном буфере, потому что никакой связи между отдельными временными наборами данных нет.

Рекомендации по работе с временными таблицами типа *InMemory*

При работе с временными таблицами типа *InMemory* не забывайте о следующих особенностях.

- Когда исключения генерируются и затем перехватываются за пределами области действия транзакции, то, если оператор *ttsabort* уже был вызван средой времени выполнения Microsoft Dynamics AX, данные временных таблиц назад не откатываются. При работе с временными наборами данных убедитесь, что вы понимаете, как наборы данных используются внутри и за пределами области действия транзакций.
- Методы-триггеры временных таблиц ведут себя почти так же, как и триггеры обычных таблиц, за некоторыми исключениями. Когда вызываются методы *insert*, *update* и *delete* временного табличного буфера, они не вызывают методы логирования в журнал базы данных или генерации оповещений класса *Application*, даже если для таблицы настроено использование журнала базы данных или функциональность оповещений.

Примечание. В общем случае нужно настроить логирование или оповещения для ваших временных таблиц типа *InMemory*. Однако поскольку обычные таблицы могут быть сделаны временными, то логирование и оповещения уже могут быть настроены для этих таблиц.

- Действия при удалении записей также не работают для временных таблиц типа *InMemory*. И хотя вы можете настроить действия при удалении для временных таблиц, среда времени выполнения Microsoft Dynamics AX не пытается их исполнять.
- Microsoft Dynamics AX позволяет вам настраивать трассировку операторов Transact-SQL либо из приложения Windows-клиента Microsoft Dynamics AX, либо из конфигурационной утилиты Microsoft Dynamics AX, либо из конфигурационной утилиты сервера Microsoft Dynamics AX. Однако трассировка операторов Transact-SQL работает лишь в случае, если они отсылаются реляционной базе данных. С помощью этих инструментов вы не можете выполнять трассировку манипуляций с данными временных таблиц типа *InMemory*. Однако вы можете использовать для этого Microsoft Dynamics AX Trace Parser. Более подробную информацию можно найти в разделе «Trace Parser в Microsoft Dynamics AX» в главе 13.
- Для определения того, работает ли табличный буфер с временным набором данных, вы можете вызвать его метод *isTmp*, который возвращает значение *true* или *false* в зависимости от того, связан табличный буфер с временной таблицей или нет.

Временные таблицы типа *TempDB*

Код приложения Microsoft Dynamics AX часто использует временные таблицы для хранения промежуточных данных. Из-за этого в некоторых случаях при работе операций на основе наборов приходится выполнять объединения временных таблиц с обычными. Однако временные таблицы типа *InMemory* предоставляют ограниченную поддержку объединений. Эти объединения выполняются уровнем доступа к данным на сервере AOS, что обеспечивает производительность, далекую от идеальной. Кроме того, как упоминалось ранее, операции на основе наборов для временных таблиц типа *InMemory* всегда понижаются до операций на основе отдельных записей. В Microsoft Dynamics AX были добавлены временные табли-

цы типа *TempDB*, чтобы обеспечить для этих сценариев высокопроизводительное решение. За счет того, что эти временные таблицы хранятся в базе данных SQL Server, становится возможным использование операций баз данных, таких как объединение.

Временные таблицы типа *TempDB* используют те же конструкции языка X++, что и временные таблицы типа *InMemory*. Ключевое отличие состоит в том, что они хранятся в базе данных *TempDB* на SQL Server. В следующем примере кода показано использование временной таблицы типа *TempDB*.

```
void select2Instances()
{
    TmpDBTable1 dbTmp1;
    TmpDBTable1 dbTmp2;

    dbTmp1.Field1 = 1;
    dbTmp1.Field2 = 'Первый';
    dbTmp1.insert();

    dbTmp2.Field1 = 2;
    dbTmp2.Field2 = 'Второй';
    dbTmp2.insert();
    info("Первый экземпляр.");
    while select * from dbTmp1
    {
        info(strfmt("%1 - %2", dbTmp1.Field1, dbTmp1.Field2));
    }
    info("Второй экземпляр.");
    while select * from dbTmp1
    {
        info(strfmt("%1 - %2", dbTmp2.Field1, dbTmp2.Field2));
    }
}
```

Пример использует временную таблицу *TmpDBTable1*, содержащую два поля. Свойство *TableType* таблицы *TmpDBTable1* установлено в *TempDB*. Аналогично *InMemory* временные таблицы типа *TempDB* создаются только при вставке первой записи в табличный буфер. Чтобы увидеть данные временной таблицы, вставьте точку останова перед первым оператором *select* в коде X++и затем откройте SQL Server Management Studio (SSMS), чтобы посмотреть, какие таблицы созданы в системной базе данных *TempDB*. Для каждого экземпляра табличного буфера временной таблицы

существует соответствующая таблица в базе данных с названием в таком формате: *t<идентификатор_таблицы>_GUID*. В приведенном примере идентификатор таблицы *TmpDBTable1* равен *101420*. Значит в базе данных *TempDB* созданы две таблицы, одна из которых называется *t101420_E448847EACA4482997F4CD8BCAAAE0CE*. После завершения выполнения метода и удаления табличных буферов из памяти эти две временные таблицы будут усечены. Среда времени выполнения Microsoft Dynamics AX использует пул для отслеживания таких временных таблиц в базе данных *TempDB* и повторно использует один из этих экземпляров таблицы, когда в коде X++ вновь создается табличный буфер типа *TmpDBTable1*.

Создание временных таблиц

Временные таблицы можно создать следующими способами:

- во время разработки приложения за счет установки свойств в метаданных;
- во время конфигурирования за счет включения лицензированных модулей или конфигурационных ключей;
- во время выполнения приложения за счет явного указания в коде X++.

Эти методы описаны в следующих разделах.

Время разработки приложения

Чтобы указать, что такая-то таблица является временной, вы должны установить соответствующее значение свойства *TableType* для нее в AOT. По умолчанию свойство *TableType* установлено в значение *Regular*. Для создания временной таблицы выберите одно из других значений: *InMemory* или *TempDB*, как показано на рис. 17-2. Временные таблицы создаются в памяти и при необходимости становятся связаными с файлами на диске либо для них создается таблица в базе данных *TempDB*.

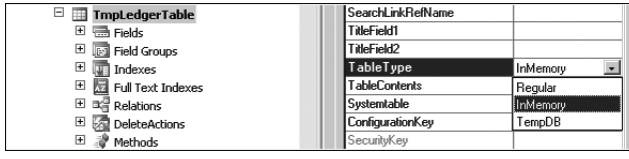


Рис. 17–2. Пометка таблицы как временной при разработке приложения

Совет. У таблиц, которые вы при разработке приложения определяете как временные, *Tmp* в названии должно быть где-то в середине, а не в начале или в конце названия, например: *InventCostTmpTransBreakdown*. Это улучшает читаемость кода X++ в случае явного использования временных таблиц. В предыдущих версиях Microsoft Dynamics AX рекомендовалось использовать *Tmp* в качестве префикса – вот почему названия определенного числа таблиц все еще следуют этому соглашению по именованию.

Время конфигурирования

Когда вы создаете таблицу в AOT, то можете привязать к ней конфигурационный ключ, указав его в свойстве *ConfigurationKey* таблицы; оно относится к категории свойств *Data* таблицы.

Когда среда времени выполнения Microsoft Dynamics AX синхронизирует таблицы с базой данных, она синхронизирует таблицы для всех модулей и конфигурационных ключей независимо от того, включены ли они, за исключением конфигурационного ключа *SysDeletedObjects*. Относится таблица к лицензированному модулю или включенному конфигурационному ключу, зависит от ее свойства *ConfigurationKey*. Если конфигурационный ключ отключен, то таблица становится отключенной и начинает вести себя как временная таблица типа *TempDB*. Таким образом, когда среда времени выполнения Microsoft Dynamics AX выполняет код X++, который использует отключенную таблицу, никаких ошибок времени выполнения не возникает. Более подробную информацию о конфигурационном ключе *SysDeletedObjects* вы можете найти в статье «Best Practices: Tables» по адресу: *http://msdn.microsoft.com/en-us/library/aa876262.aspx*.

Примечание. Отключена таблица или нет, не влияет на те таблицы, которые изначально созданы как временные. Таблица остается временной несмотря на то, что ее конфигурационный ключ выключен, поэтому вы можете полагаться на одинаковое ее поведение независимо от настройки конфигурационного ключа.

Время выполнения

С помощью кода X++ вы можете превратить обычную таблицу во временную типа *InMemory*, вызвав на табличном буфере метод *setTmp*. С этого

момента табличный буфер будет рассматриваться, как если бы свойство таблицы *TableType* было установлено в значение *InMemory*.

Примечание. Вы не можете объявить табличный буфер для временной таблицы и сделать ее постоянной отчасти из-за того, что связанной таблицы нет в реляционной базе данных.

Следующий код X++ демонстрирует использование метода *setTmp*. Здесь объявляются два табличных буфера одного и того же типа, но один из них – временный, и все записи из базы данных вставляются во временную версию таблицы. Таким образом, временный табличный буфер указывает на набор данных, содержащий полную копию всех записей из базы данных, относящихся к текущей компании.

```
static void TmpCustTable(Args _args)
{
    CustTable custTable;
    CustTable custTableTmp;

    custTableTmp.setTmp();
    ttsbegin;
    while select custTable
    {
        custTableTmp.data(custTable);
        custTableTmp.doInsert();
    }
    ttscommit;
}
```

Обратите внимание, что код **X++, приведенный выше, использует ме**тод *doInsert* для вставки записей во временную таблицу. Это предотвращает выполнение перекрытого метода *insert*. Метод *insert* вставляет записи в другие таблицы, которые не переключаются автоматически в режим временных таблиц только от того, что табличный буфер *custTable* является временным.

Внимание. Следует очень осторожно делать временные табличные буферы из обычных таблиц, потому что непреднамеренно может быть выполнена прикладная логика в перекрытых методах, которая манипулирует данными в обычных таблицах. Это может

произойти, если временный табличный буфер используется на форме и движок форм вызывает методы, обращающиеся к базе данных.

Суррогатные ключи

Существенное изменение, касающееся ключевых полей таблиц, связано с появлением в Microsoft Dynamics AX 2012 суррогатных ключей. Суррогатный ключ – это генерируемый системой первичный ключ, состоящий из одного поля и не несущий какой-либо смысловой нагрузки в предметной области. Его значение привязано к инсталляции Microsoft Dynamics AX, в которой оно генерируется, поэтому значение суррогатного ключа, сгенерированное в одной инсталляции, не может использоваться в другой.

Естественным выбором на роль суррогатного ключа в Microsoft Dynamics AX является поле *RecId*. Однако в Microsoft Dynamics AX 2009 и более ранних версиях индекс по *RecId* все еще включает поле *DataAreaId* для таблиц, чьи данные хранятся в разрезе компаний. В Microsoft Dynamics AX 2012 индекс по *RecId* более не содержит поле *DataAreaId* и становится уникальным ключом, состоящим из одного поля.

Поддержка суррогатных ключей по умолчанию включена, когда вы создаете новую таблицу в AOT. Чтобы использовать в таблице шаблон суррогатного ключа, установите значение свойства *PrimaryIndex* в *SurrogateKey*, как показано на рис. 17-3.

Примечание. Значение свойства *SurrogateKey* недоступно для таблиц, созданных в более ранних версиях Microsoft Dynamics AX. Если вы хотите использовать суррогатный ключ для уже существующей таблицы, то должны пересоздать ее в AOT.

Определение суррогатного ключа на таблице в Microsoft Dynamics AX дает несколько преимуществ. Первое – это производительность. Когда для объединения двух таблиц в запросе используются суррогатный ключ и внешний, который на него ссылается, производительность заметно улучшается по сравнению с объединениями, использующими другие типы данных. Это преимущество еще более заметно при сравнении с Microsoft Dynamics AX 2009 и более ранними версиями, потому что в них ядро автоматически добавляет поле *DataAreaId* в любой индекс таблицы, чьи дан-

ные хранятся в разрезе компаний. Определить такие таблицы вы можете по значению свойства *SaveDataPerCompany*. Без суррогатных ключей объединение должно быть основано как минимум на двух полях, одним из которых должно быть *DataAreaId*.

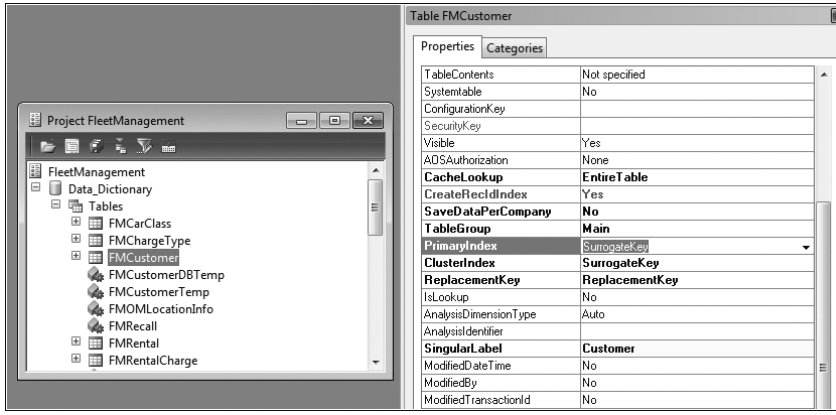


Рис. 17–3. Суррогатный ключ в таблице *FMCustomer*

Второе преимущество использования суррогатных ключей в том, что значение суррогатного ключа никогда не меняется, что устраняет необходимость изменять значения внешних ключей. К примеру, таблица *Currency* (рис. 17-4) использует в качестве первичного ключа индекс *CurencyCodeIdx*, который содержит поле *CurrencyCode*. Таблица проводок по ГК содержит два внешних ключа, которые ссылаются на таблицу *Currency* через поле *CurrencyCode*. Если когда-нибудь для записи в таблице *Currency* понадобилось бы изменить значение поля *CurrencyCode*, соответствующие записи в таблице проводок по ГК также необходимо было бы обновить. Но если бы в таблице *Currency* использовался суррогатный ключ, а проводки по ГК содержали суррогатный внешний ключ, то вы могли бы обновить значение поля *CurrencyCode* в таблице Currency, не затрагивая связи по ключам между записями в таблице *Currency* и в таблице проводок по ГК.

Третье преимущество использования суррогатных ключей в том, что это всегда ключ из одного поля. Некоторые возможности SQL Server, такие как полнотекстовый поиск, требуют использования ключа из одного поля, так что использование суррогатных ключей позволяет вам воспользоваться преимуществами таких возможностей.

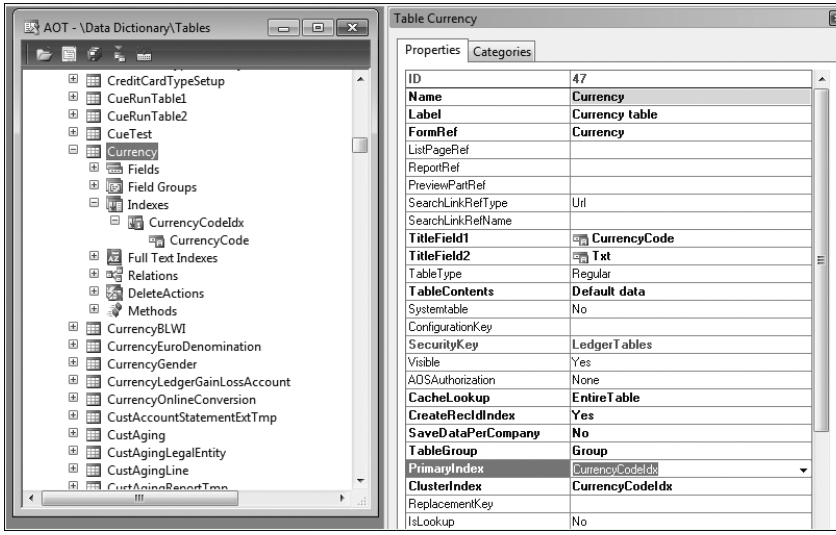


Рис. 17–4. Таблица *Currency* без суррогатного ключа

У суррогатных ключей есть и недостатки. Наиболее заметный заключается в том, что ключ неудобен для человеческого восприятия. Когда вы смотрите на внешние ключи в связанной таблице, нелегко определить, с какой записью они связаны. Чтобы отобразить осмысленную информацию, идентифицирующую внешний ключ, нужно выбрать из связанной записи какую-то удобную для восприятия человеком информацию и отобразить вместо него. Это требует объединения со связанной таблицей, а объединения создают дополнительную нагрузку с точки зрения производительности.

Альтернативные ключи

Для таблицы *потенциальным ключом* является минимальный набор полей, который уникально идентифицирует каждую запись в ней. *Альтернативный ключ* – это потенциальный ключ для таблицы, не являющийся первичным ключом. В Microsoft Dynamics AX2012 вы можете пометить, что тот или иной уникальный индекс является альтернативным ключом.

Поскольку Microsoft Dynamics AX уже использует концепцию уникальных индексов, вы можете поинтересоваться, какую пользу несет использование концепции альтернативного ключа. Разработчики создают

уникальные индексы по разным причинам. Обычно один уникальный индекс служит первичным ключом. Иногда создаются дополнительные уникальные индексы для повышения производительности, например, под определенный шаблон запросов. Когда вы смотрите на уникальные индексы на таблице, то не всегда очевидно, какой из них был создан для использования в качестве первичного ключа, а какие – по другим причинам. Например, если бы вам нужно было извлечь модель данных и представить ее бизнес-аналитику, вам не хотелось бы, чтобы он увидел ключи, созданные исключительно для оптимизации запросов. Для этого необходим способ отделения вашей семантической модели от физической модели данных. В этом вам поможет возможность указать, что определенные уникальные индексы являются альтернативными ключами.

На рис. 17-5 показан уникальный индекс, который был добавлен на таблицу *FMVehicleModel* демонстрационного приложения Fleet Management (управление парком автомобилей). Индекс не является первичным ключом для таблицы, поэтому его свойство *AlternateKey* установлено в *Yes*.

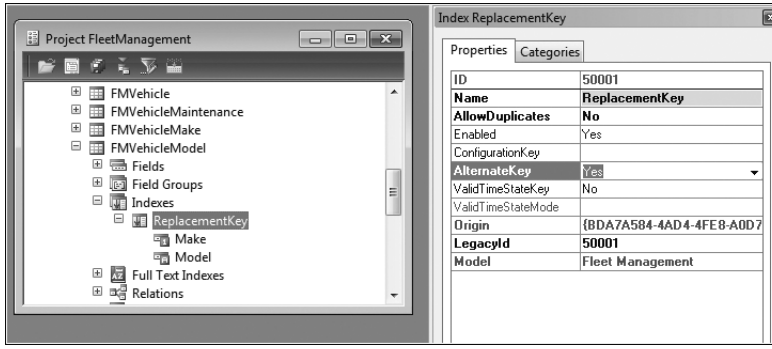


Рис. 17–5. Альтернативный ключ на таблице *FMVehicleModel*

Связи между таблицами

Связи между таблицами (в Microsoft Dynamics AX используется термин *relations* – отношения) являются ключевой информацией для модели данных и поведения во время выполнения. В Microsoft Dynamics AX они используются в следующих сценариях времени выполнения.

- Условия объединения в запросе или связи между источником данных формы и dynalink на другой форме.

- Действия удаления таблицы (delete actions)
- Условия для таблицы подстановок в случае привязанных к данным (через источник данных) или не привязанных к данным элементов управления на формах.

В Microsoft Dynamics AX 2012 для связей между таблицами сделано несколько изменений и улучшений.

Отношения расширенных типов данных и связи между таблицами

В Microsoft Dynamics AX связи между таблицами могут быть заданы явно на самих таблицах или же неявно – через настройки отношений, определенных на расширенных типах данных (EDT), которые непосредственно или косвенно используются табличными полями. Ядро Microsoft Dynamics AX просматривает свойства отношений для расширенных типов данных, а затем – для связей, определенных непосредственно на таблице. Порядок может быть обратным в зависимости от сценария.

Существует несколько проблем, связанных с определением отношений на расширенных типах данных и затем смешением их со связями, определенными на таблицах. Во-первых, отношения EDT задают отношения только для одного поля. Их нельзя использовать для настройки отношений между несколькими полями, что приводит к неполным отношениям, используемым в объединениях таблиц в запросе или в действиях удаления таблиц. Во-вторых, большинство свойств для отношения зависят от контекста, в котором это отношение используется. Этот контекст не может быть задан в отношениях на расширенных типах данных, потому что они хранятся в AOT централизованно. К примеру, название роли и связанной с ней другой роли, а также кардинальность связи могут отличаться для связей между различными таблицами. В-третьих, сложно понять, сколько на самом деле существует связей для таблицы, потому что те связи, которые определены явно на самой таблице, представляют лишь часть общей картины. Вам еще нужно просмотреть отношения, определенные на расширенных типах данных, непосредственно используемых полями таблицы, а также на тех расширенных типах, которые являются базовыми для расширенных типов полей.

Чтобы справиться с этими проблемами, в Microsoft Dynamics AX 2012 большинство отношений было перенесено с EDT на связанные таблицы. Чтобы начать процесс отказа от узла *Relations* в отдельных расширен-

ных типах данных, добавление новых отношений было запрещено. Если на EDT не определены отношения, то и узел *Relations* не отображается в Microsoft Dynamics AX 2012. В EDT появился новый узел *Table References* для случаев, когда элемент управления привязан напрямую к EDT, а не через табличное поле. Для уменьшения ручной работы по добавлению тех связей между таблицами, которые прежде заменялись отношениями EDT, система запрашивает у вас, нужно ли автоматически добавить связь между таблицами, когда вы указываете для поля таблицы EDT с корректной ссылкой на внешний ключ.

Поскольку система времени выполнения Microsoft Dynamics AX выполняет поиск отношений EDT и связей между таблицами в определенном порядке, то в случае обновления с предыдущих версий вам нужно убедить, что до и после обновления подхватывается одна и та же связь между таблицами. Это особенно важно, когда переносятся отношения EDT в нескольких связях между двумя таблицами. Среда времени выполнения Microsoft Dynamics AX обеспечивает обратную совместимость за счет того, что проверяет некоторые свойства, которые были установлены на полях таблиц и связях между таблицами. Эти свойства позволяют среде времени выполнения определить, была ли связь между таблицами создана из отношения EDT, которое существовало ранее. Обсуждаемые свойства описаны в табл. 17-1.

Табл. 17–1. Свойства полей таблицы

Расположение свойства	Название свойства	Значения	Описание
Связь между таблицами	EDTRelation	Yes/No	Указывает ядру, было ли свойство создано из-за миграции отношения EDT. Ядро использует эту информацию для сортировки и выбора связи для объединения таблиц в запросе, использования в таблицах подстановки и действиях удаления таблиц, когда не указана явная связь между таблицами
Поле таблицы	IgnoreEDTRelation	Yes/No	Указывает, нужно ли выполнять миграцию отношения EDT для поля таблицы в связь между таблицами. Среда времени выполнения Microsoft Dynamics AX не использует это свойство, однако его использует Инструмент переноса отношения EDT

Табл. 17–1. Свойства полей таблицы (*окончание*)

Расположение свойства	**Название свойства**	**Значения**	**Описание**
Ссылка в связи между таблицами	Source-EDT	Название EDT	Используется средой времени выполнения Microsoft Dynamics AX для определения случаев, в которых нужно использовать отношение EDT

Вам не нужно вручную выполнять миграцию отношений EDT в связи между таблицами и затем устанавливать значения свойств, описанных в табл. 17-1, – в этом процессе вам поможет Инструмент переноса отношения EDT. Он вызывается из меню Сервис > Обновить код > Инструмент переноса отношения EDT, как показано на рис. 17-6. Более подробную информацию об этом инструменте вы можете найти в разделе «EDT Relation Migration Tool» по адресу: *http://msdn.microsoft.com/en-us/library/gg989788.aspx*.

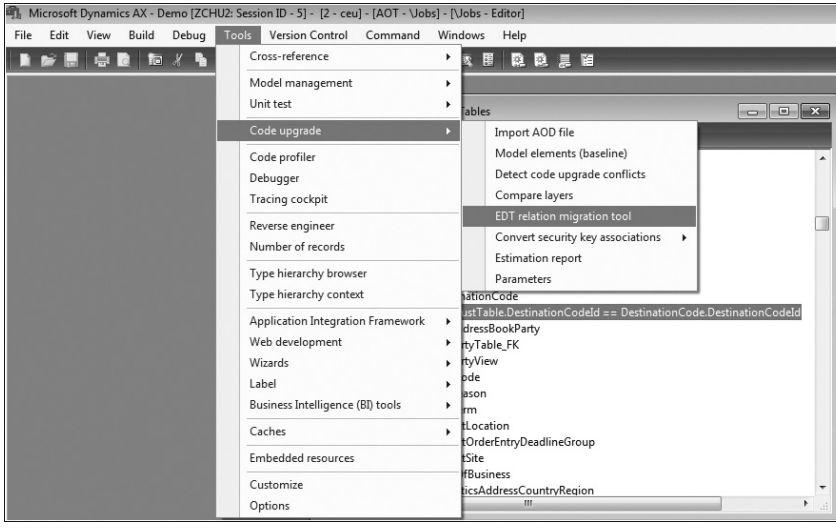


Рис. 17–6. Инструмент переноса отношения EDT

После миграции отношений EDT в связи между таблицами проверьте, что действия удаления таблиц, объединения таблиц в запросах и таблицы подстановки работают так же, как до миграции. Особое внимание удели-

те случаям, в которых миграция отношения EDT привела к созданию нескольких связей между двумя таблицами. Инструмент переноса отношения EDT перечисляет те объекты приложения, которые он изменяет.

Вот несколько примеров объектов приложения, которые могут быть изменены. На таблице *SalesTable* определены две связи между ней и таблицей *CustTable*, при этом одна из связей была перенесена из отношения EDT, поскольку на таблице ее прежде не существовало. Две упомянутые связи между таблицами основаны на полях *CustAccount* и *InvoiceAccount*. Для действий удаления таблицы до и после миграции должна подхватываться связь по полю *CustAccount*. Далее в запросе *SalesHeading* есть объединение таблиц *SalesTable* и *CustTable*, для которого на источнике данных для таблицы *SalesTable* свойство *Relations* установлено в *Yes*. Этот запрос до и после миграции должен подхватывать связь между таблицами по полю *CustAccount* вместо связи по полю *InvoiceAccount*.

Связи по внешнему ключу

Одной из целей Microsoft Dynamics AX 2012 было сделать модель данных более согласованной. Для этого там, где возможно, следует использовать суррогатные ключи. Также для этого всегда, когда возможно, следует использовать для связей по внешнему ключу только первичный ключ связанной таблицы. Последнее облегчается за счет появления в Microsoft Dynamics 2012 специального типа связи по внешнему ключу, который вы можете использовать при создании связей между таблицами, как показано на рис. 17-7. Связь между таблицами по внешнему ключу позволяет использовать только два типа ссылок на связанную таблицу. Первый – это ссылка на первичный ключ связанной таблицы, второй – ссылка на ее альтернативный ключ, состоящий из одного поля. Эта ссылка на альтернативный ключ, состоящий из одного поля, предоставляется для того, чтобы уменьшить число объединений по суррогатному внешнему ключу, которые обсуждались в разделе «Суррогатные ключи» ранее в этой главе.

Если вы используете понятный человеку альтернативный ключ, то можете отображать его на формах без необходимости объединения со связанной таблицей. Вы можете спросить, почему только альтернативные ключи из одного поля разрешено использовать для ссылок по внешнему ключу. Это необходимо, чтобы сбалансировать производительность в различных вариантах использования, когда ссылка по внешнему ключу в действительности нужна вам для объединения таблиц, а не для улучшения восприятия человеком значений внешнего ключа, отображаемого в поль-

зовательском интерфейсе. Объединения таблиц, использующие меньше данных из полей (тут важен как размер поля, так и число полей), работают быстрее. За счет того, что в данной возможности разрешено использовать альтернативные ключи только из одного поля, ограничено падение производительности в объединениях таблиц.

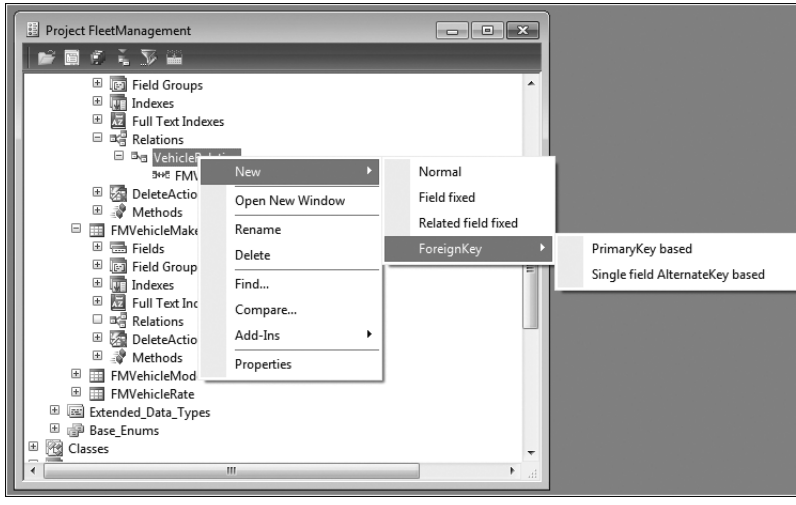


Рис. 17–7. Новая связь между таблицами в Microsoft Dynamics AX 2012 по внешнему ключу

Использование последовательного шаблона для создания связей между таблицами также может дать преимущества с точки зрения производительности. Например, если одна таблица ссылается на *CustTable* с помощью *keyA*, а другая – на *CustTable* с помощью *keyB*, то для соотнесения записей в этих двух таблицах нужно обе связать с таблицей *CustTable*. Однако если обе таблицы используют один и тот же ключ, то их записи можно соотнести напрямую, устраняя таким образом необходимость в дополнительных объединениях.

У связей по внешнему ключу есть определенные возможности, которых лишены другие типы связей. Например, вы можете использовать их в качестве условий объединения в запросах. За счет этого вам не нужно вручную указывать, по каким полям необходимо осуществлять объединение, что чревато возникновением ошибок. Кроме того, для внешних ключей можно генерировать методы навигации, как описано в следующем разделе.

Свойство *CreateNavigationPropertyMethods*

Когда вы разворачиваете узел *Relations* таблицы в AOT, то видите определенные там связи между этой и другими таблицами. Для связей по внешнему ключу доступно дополнительное свойство *CreateNavigationPropertyMethods*, имеющее специальное назначение. Установка этого свойства в *Yes,* как показано на рис. 17-8, приводит к тому, что на табличном буфере создаются специальные сгенерированные ядром методы. Вы можете использовать эти методы для установки, получения и перехода к связанному табличному буферу на основе указанной связи между таблицами. В примере далее в этом разделе показываются сигнатуры этих методов и способы их использования.

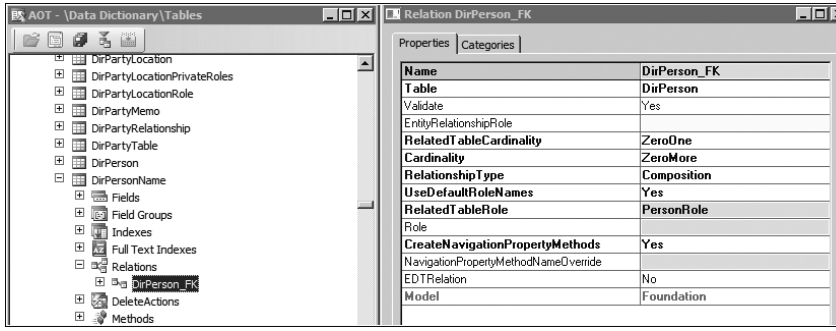


Рис. 17–8. Свойство *CreateNavigationPropertyMethods* на связи между таблицами по внешнему ключу

Метод свойств навигации *setter* связывает вместе два табличных буфера, относящихся друг к другу. Он часто используется вместе с классом *UnitOfWork* для создания записей в базе данных из этих табличных буферов. Это позволяет фактически создать в памяти граф объекта со связанными табличными буферами, который затем можно сохранить в базе данных, получив корректные ссылки между записями на основе установленных отношений. Более подробную информацию вы можете найти в разделе «Единица работы» далее в этой главе.

Метод свойств навигации *getter* возвращает связанный табличный буфер, если он был ранее установлен методом *setter*. В противном случае метод извлекает связанный табличный буфер из базы данных. Таким образом, можно фактически заменить шаблон использования метода *find*, который широко применяется на таблицах. В последнем случае возвра-

щенный табличный буфер не будет связан тем табличным буфером, на котором вызван метод. Это означает, что вызванный метод в следующий раз попытается снова получить запись из базы данных. Имейте в виду, что, когда метод свойств навигации *getter* отправляет базе данных запрос для получения связанной записи, он выбирает все поля этой записи. Это может повлиять на производительность, в частности, в тех случаях, когда прежде вы для оптимизации производительности выбирали лишь небольшое подмножество полей.

Следующий код использует метод *DirPartyTable_FK* для получения связанного табличного буфера *DirPartyTable* для клиента с кодом *1101* и выводит название клиента в Infolog.

```
static void NavigationPropertyMethod(Args _args)
{
    CustTable cust;

    select cust where cust.AccountNum == '1101';

    // Метод DirPartyTable_FK() получает связанную запись DirPartyTable
    // с использованием отношения DirPartyTable_FK, определенного на CustTable

    info(strFmt('Customer name for %1 is %2', cust.AccountNum, cust.DirPartyTable_FK().
Name));
}
```

На рис. 17-9 показано, что в этом примере выводится в Infolog.

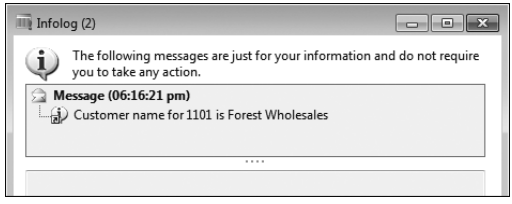


Рис. 17–9. Вывод из примера использования метода *DirPartyTable_FK*

Однако если до этого был использован метод навигации *setter*, чтобы установить связанную запись *DirPartyTable*, то всегда будет возвращаться эта запись и среда времени выполнения не будет отправлять запрос базе данных.

```
static void NavigationPropertyMethodSetter(Args _args)
{
```

```
    CustTable cust;
    DirPartyTable dp;

    select cust where cust.AccountNum == '1101';
    dp.Name = 'NotARealCustomer';

    // Устанавливаем связанную запись DirPartyTable

    cust.DirPartyTable_FK(dp);

    // Метод DirPartyTable_FK() возвращает запись DirPartyTable,
    // установленную выше, а не обращается к базе данных.

    info(strFmt('Customer name for %1 is %2', cust.AccountNum, cust.DirPartyTable_FK().
Name));
}
```

Вывод в Infolog для этого примера показан на рис. 17-10.

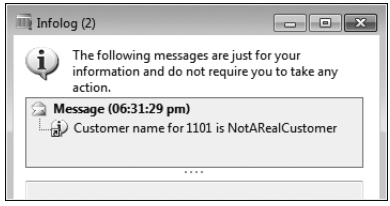


Рис. 17–10. Вывод для примера с использованием метода свойств навигации *setter*

У каждого метода навигации должно быть название, и, подобно любому другому табличному методу, его название не должно конфликтовать с названиями других методов. По умолчанию для формирования названия метода используется свойство *RelatedTableRole*. Если во время компиляции обнаружится конфликт с названием другого метода, то будет выведено сообщение об ошибке. В случае конфликта имен используйте свойство *NavigationPropertyMethodNameOverride* для указания того, какое название следует использовать.

Наследование таблиц

Наследование таблиц давно было частью расширенного моделирования связей сущностей (entity relationship, ER), но в предыдущих версиях Microsoft Dynamics AX для него не было встроенной поддержки, поэтому любое наследование или объектно-ориентированные характеристики разработчику приходилось реализовывать вручную. Microsoft Dynamics AX 2012

содержит встроенную полную поддержку наследования таблиц, включая моделирование, средства языка разработки, средства среды времени выполнения и пользовательский интерфейс.

Моделирование наследования таблиц

Для моделирования наследования таблиц в Microsoft Dynamics AX 2012 вы сначала должны создать корневую таблицу, а затем – производную от нее таблицу; эти задачи описаны ниже. В последующих разделах описывается, как работать с существующими таблицами, просматривать иерархии типов и задавать поведение таблиц.

Создание корневой таблицы

Сначала вы должны создать таблицу, которая является корнем табличной иерархии. Прежде чем создавать в таблице какие-либо поля, установите ее свойство *SupportInheritance* в *Yes*. Для корневой таблицы вы должны добавить поле типа *Int64*, называющееся *InstanceRelationType*, которое будет содержать информацию о действительном типе той или иной записи в таблице. Для этого поля свойство *ExtendedDataType* должно иметь значение *RelationType*, а свойство *Visible* установлено в *No*. После создания этого поля вы должны указать его название в значении свойства *InstanceRelationType* корневой таблицы. Далее можно продолжить обычное моделирование корневой таблицы, как для таблиц, не использующих наследование.

Создание производной таблицы

Создайте производную таблицу и установите ее свойство *SupportInheritance* в *Yes*. В свойстве *Extends* укажите ту таблицу, на которой должна быть основана создаваемая производная таблица. Данные свойства необходимо установить до того, как вы создадите какие-либо поля на таблице. Это позволит гарантировать уникальность названий и идентификаторов полей для всех таблиц в иерархии, что необходимо для корректной работы среды времени выполнения. Кроме того, это позволяет выбрать различные модели хранения, такие как хранение всех типов в одной таблице, не вызывая коллизии имен. Хранение обсуждается далее в этом разделе.

Работа с существующими таблицами

Если в таблице уже были поля, прежде чем вы добавили ее в иерархию, то может потребоваться обновить названия и идентификаторы полей как в метаданных, так и в коде. Если таблицы уже содержат данные, то эти данные необходимо обновить для работы с новой иерархией таблиц. Все

это – весьма нетривиальные задачи, поэтому создание новых иерархий наследования таблиц из уже существующих таблиц не поддерживается.

Просмотр иерархии типов

Для просмотра иерархии наследования таблиц вы можете использовать Браузер иерархии типа. Для этого щелкните правой кнопкой мыши по таблице в AOT и выберите пункт Надстройки > Браузер иерархии типа. На рис. 17-11 показана иерархия для таблицы *FMVehicle*.

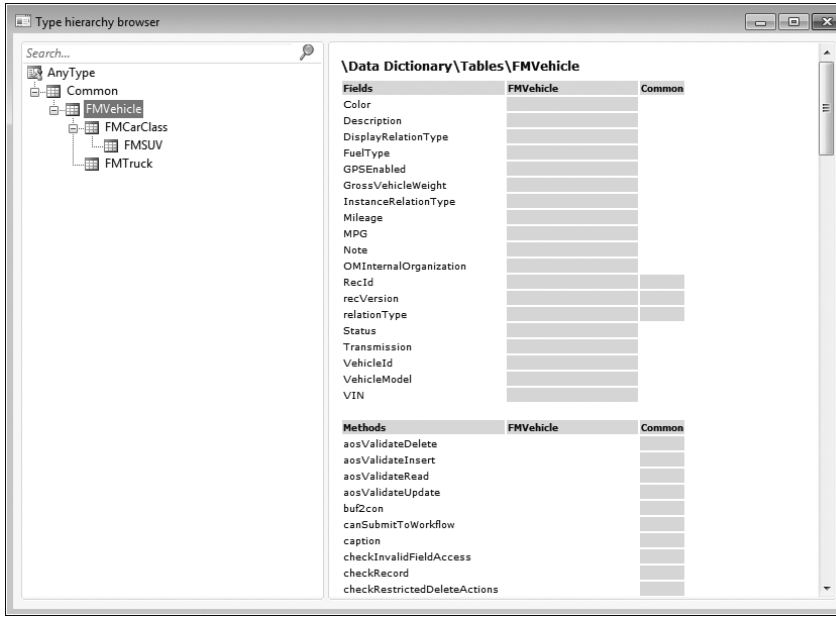


Рис. 17–11. Иерархия для таблицы *FMVehicle*

Задание поведения таблиц

Значения некоторых свойств являются общими для всех таблиц в иерархии наследования, за счет чего их поведение в рамках иерархии остается согласованным. Сюда входят настройка кэширования, контроль параллельного доступа (OCC) и настройка хранения данных в разрезе компаний. Используемые конфигурационные ключи должны быть согласованы с иерархией наследования таблиц. Иными словами, если конфигурационный ключ отключается для базовой таблицы, то конфигурационный ключ

для производной таблицы не должен включаться. Это условие проверяется во время компиляции таблицы, входящей в иерархию, и его нарушение приводит к ошибкам компиляции. Более подробную информацию о конфигурационных ключах вы можете найти в главе 11 .

Вы можете указать, является ли таблица в иерархии наследования определенной или абстрактной. По умолчанию таблицы являются определенными, что означает, что вы можете создать запись данного табличного типа. Указание того, что таблица является абстрактной, означает, что вы не можете создать запись данного табличного типа. Любая запись в абстрактной таблице должна быть типа одной из производных от нее таблиц (выше по иерархии), не помеченных как абстрактные. Эта концепция согласуется с концепцией абстрактного класса в объектно-ориентированных языках программирования.

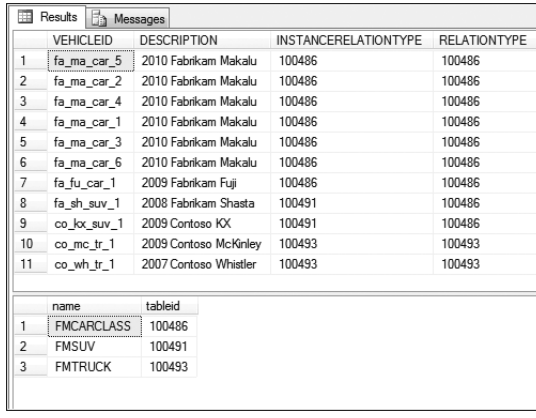
Модель наследования таблиц в Microsoft Dynamics AX 2012 является дискретной. Любая запись в иерархии таблиц может относиться лишь к одному конкретному типу (таблице, которая не помечена как абстрактная). Вы не можете изменять тип записи с одного конкретного типа на другой после того, как запись была создана.

Модель хранения для наследования таблиц

В некоторых реализациях технологий объектно-реляционного (object-relational, OR) отображения вы можете выбирать, как иерархия наследования таблиц будет отображена на хранилище данных. Обычно при этом предлагается выбор из одной таблицы на все моделируемые типы, одной таблицы на каждый конкретный тип или же одной таблицы на каждую иерархию. Microsoft Dynamics AX 2012 создает одну таблицу на каждый моделируемый тип (это относится к версии Microsoft Dynamics AX 2012 R1; в версии Microsoft Dynamics AX 2012 R2 создается одна таблица на каждую иерархию. – *Прим. перев.)*. Как и обычная таблица, таблица в иерархии наследования отображается на физическую таблицу в базе данных. Записи в иерархии наследования связываются по полю *RecId*. Данные для определенной записи, относящейся к экземпляру типа, хранятся в нескольких таблицах в иерархии, однако у них у всех одно и то же значение *RecId*.

Для каждой таблицы в иерархии наследования автоматически создается системное поле с названием *RELATIONTYPE*. Вы можете увидеть его в SQL Server, но оно не отображается в AOT. Это поле играет роль дискриминатора.

Данные для конкретного типа хранятся в нескольких таблицах, составляющих цепочку иерархии для данного типа. Для записи в одной таблице поле дискриминатора указывает следующую таблицу в цепочке. На рис. 17-12 показаны некоторые записи из таблицы *FMVehicle* и соответствующие идентификаторы производных таблиц. Значение поля *InstanceRelationType* для легковых автомобилей равно идентификатору таблицы *FMCarClass*, для внедорожников оно равно идентификатору таблицы *FMSUV*, а для грузовиков – идентификатору таблицы *FMTrucks*. Эти таблицы представляют конкретные типы каждой записи в таблице *FMVehicle*. Значение поля *RelationType* и для легковых автомобилей, и для внедорожников равно идентификатору таблицы *FMCarClass*, поскольку *FMCarClass* является непосредственно следующей производной таблицей для этих записей. Для грузовиков значение поля *RelationType* равно идентификатору таблицы *FMTruck*.



Results | Messages

	VEHICLEID	DESCRIPTION	INSTANCERELATIONTYPE	RELATIONTYPE
1	fa_ma_car_5	2010 Fabrikam Makalu	100486	100486
2	fa_ma_car_2	2010 Fabrikam Makalu	100486	100486
3	fa_ma_car_4	2010 Fabrikam Makalu	100486	100486
4	fa_ma_car_1	2010 Fabrikam Makalu	100486	100486
5	fa_ma_car_3	2010 Fabrikam Makalu	100486	100486
6	fa_ma_car_6	2010 Fabrikam Makalu	100486	100486
7	fa_fu_car_1	2009 Fabrikam Fuji	100486	100486
8	fa_sh_suv_1	2008 Fabrikam Shasta	100491	100486
9	co_kx_suv_1	2009 Contoso KX	100491	100486
10	co_mc_tr_1	2009 Contoso McKinley	100493	100493
11	co_wh_tr_1	2007 Contoso Whistler	100493	100493

	name	tableid
1	FMCARCLASS	100486
2	FMSUV	100491
3	FMTRUCK	100493

Рис. 17–12. Таблицы в иерархии *FMVehicle*

Полиморфное поведение

Когда вы выполняете запрос к таблице, которая является частью иерархии таблиц, среда времени выполнения Microsoft Dynamics AX по умолчанию обеспечивает точное воспроизведение типа. Это означает, что если для таблицы выполняется оператор *select* *, то возвращаются все записи для данного типа таблицы. К примеру, если вы выполняете запрос *select * from DirPartyTable*, то возвращаются все экземпляры *DirParty*, включая *DirPerson*, *OperatingUnit* и т.д. Более того, поскольку используется *select* *, запрос

возвращает данные полностью. Это означает, что таблица *DirPartyTable* должна быть объединена со всеми ее производными таблицами.

При выполнении *select ** для таблицы, являющейся частью иерархии таблиц, эта таблица объединяется по inner join со всеми таблицами ниже по иерархии (вплоть до корневой таблицы), а затем по outer join со всеми производными таблицами (включая производные таблицы всех уровней). Причина этого в том, что любая запись в этой таблице должна иметь соответствующую запись во всех родительских таблицах, но также она может иметь связанные записи, соответствующие любому производному конкретному типу. Таким образом, всегда обеспечивается выборка полного набора данных для записи, независимо от того, к какому конкретному типу она относится. Подобно полиморфному поведению в объектно-ориентированном программировании, этот механизм обеспечивает полиморфную выборку данных для таблицы, которая является частью иерархии наследования таблиц. Это поведение очень удобно в тех случаях, когда вам необходимы все данные для записи. Вы можете использовать функцию динамического связывания методов, поддерживаемую в наследовании таблиц, чтобы писать чистый и расширяемый код.

Например, в проекте Fleet Management на таблице *FMVehicle* есть метод *doAnnualMaintenance*, который просто выбрасывает исключение. Это происходит потому, что таблица *FMVehicle* является абстрактной и любая конкретная таблица, производная от нее, должна перекрыть этот метод. Этот метод перекрыт на таблицах *FMCarClass*, *FMTruck* и *FMSUV*, как показано на рис. 17-13. Обратите внимание, что каждый перекрытый метод обращается к полям, которые недоступны из базовой таблицы.

Следующий код выбирает данные из таблицы *FMVehicle* и вызывает метод *doAnnualMaintenance*.

```
static void PolyMorphicQuery(Args _args)
{
    FMVehicle vehicle;

    while select vehicle
    {
        vehicle.doAnnualMaintenance();
    }
}
```

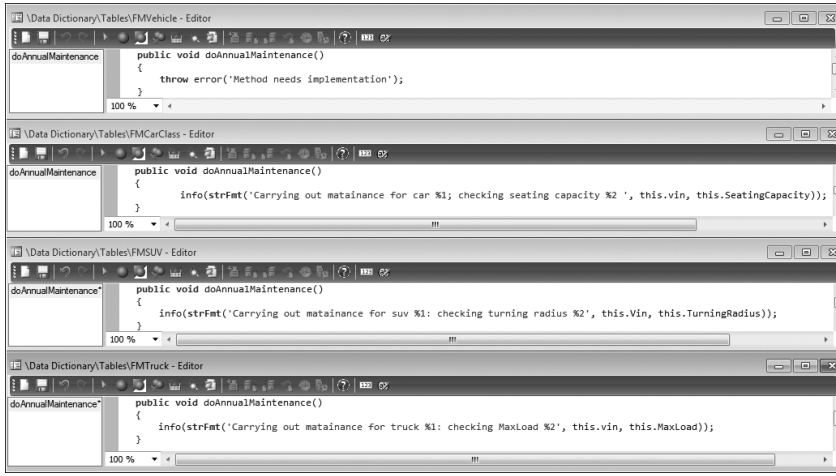


Рис. 17–13. Перекрытый метод *doAnnualMaintenance* на производных таблицах

Если вы выполните код в виде задания, то получите результаты, схожие с показанными на рис. 17-14.

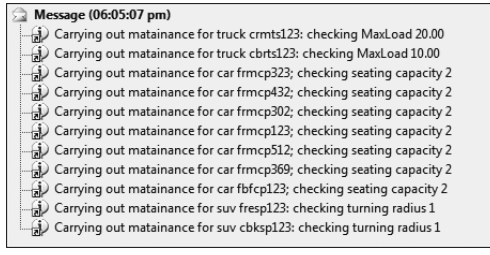


Рис. 17–14. Результаты вызова перекрытого метода *doAnnualMaintenance*

Как видите, несмотря на то, что оператор *select* выполнен для таблицы *FMVehicle*, возвращаются поля из производных таблиц. Настоящий оператор *select* для этого запроса выглядит примерно так:

```
SELECT <все поля из всех таблиц в иерархии > FROM FMVEHICLE T1 LEFT OUTER JOIN
FMTRUCK
T2 ON (T1.RECID=T2.RECID) LEFT OUTER JOIN FMCARCLASS T3 ON (T1.RECID=T3.RECID)
LEFT OUTER JOIN
FMSUV T4 ON (T3.RECID=T4.RECID)
```

Совет. Вы можете получить запрос Transact-SQL для оператора *select* напрямую из кода X++ без использования SQL Profiler. Вот пример:

```
static void PolyMorphicQuery(Args _args)
{
    FMVehicle vehicle;

    select generateonly vehicle;

    info(vehicle.getSQLStatement());
}
```

Соображения производительности

Когда иерархия наследования содержит множество таблиц на одном уровне наследования, на множестве уровней наследования или и то и другое, то полиморфный запрос может привести к многочисленным объединениям таблиц, что негативно сказывается на производительности. К примеру, запрос *select * from DirPartyTable* приводит к объединению восьми таблиц. С осторожностью используйте возможность наследования таблиц, определяйте, действительно ли вам нужны все данные из каждого производного типа. Если нет, то следует явно указывать список полей, которые вам в действительности нужны. (Обратите внимание, что вы также можете указывать поля из производных таблиц, когда создаете запрос в AOT или используете объект Query в коде X++. Но когда вы пишете *select* в коде X++, то можете указывать только поля из текущей таблицы или ее родительских таблиц.) Среда времени выполнения Microsoft Dynamics AX в этом случае добавит объединения только с теми таблицами, которые содержат поля из заданного списка. К примеру, если вы измените запрос к таблице *DirPartyTable* на *select name from DirpartyTable*, то в этот запрос будет включена только таблица *DirPartyTable* – никаких объединений с другими таблицами в иерархии создано не будет, поскольку не происходит доступа к данными из них. Аккуратное построение запроса может существенно улучшить производительность.

Перечисление лишь тех полей, которые нужны, дает преимущество не только в данном случае, но и в целом является хорошей практикой. Это может позволить SQL Server использовать покрывающий индекс при обработке запроса и снизить нагрузку на сеть. При этом обычно вызывают

беспокойство случаи, когда выбранный табличный буфер нужно передать в другую функцию или модуль, поскольку при этом нужно убедиться, что эта другая функция не использует поля, которые не были выбраны. При попытке чтения данных полей, которые отсутствовали в списке выборки, Microsoft Dynamics AX 2009 и более ранние версии не выбрасывали исключение. Вы получали то значение, которое оказывалось в табличном буфере, и в большинстве случаев это было пустое значение. В Microsoft Dynamics AX 2012 попытка доступа к полю, которого не было в списке выборки, приводит к выбрасыванию исключения, но только в тех случаях, когда поле относится к таблице, входящей в иерархию наследования таблиц. Есть настройка, управляющая тем, будет ли всегда выбрасываться исключение или только выводиться предупреждение. Для изменения этой настройки выполните следующие действия.

1. Откройте форму Администрирование системы > Настройка > Система > Конфигурация сервера.
2. На экспресс-вкладке Оптимизация производительности в группе Настройки производительности выберите значение из выпадающего списка рядом с Ошибка или недопустимое обращение к полю. Для сохранения обратной совместимости и снижения дополнительной нагрузки во время выполнения настройка по умолчанию выключена. Рекомендуется включать ее только для целей тестирования.

Единица работы

Для любой ERP-системы важно сохранение ссылочной целостности. Microsoft Dynamics AX 2012 позволяет вам моделировать связи между таблицами с использованием более обширных метаданных и выражать ссылочную целостность в вашей модели данных более точно. Однако за обеспечение целостности данных все еще отвечает приложение. Связи между таблицами в Microsoft Dynamics AX реализованы не на уровне ограничений базы данных по внешнему ключу. Такая реализация привела бы к дополнительной нагрузке на SQL Server, к тому же, по соображениям производительности и ряду других, код приложения может временно нарушать ссылочную целостность и исправлять эти нарушения позднее.

Поддержание ссылочной целостности требует выполнения операций с данными в правильной последовательности. Это особенно явственно проявляется при создании и удалении записей. Сначала должна быть создана

родительская запись, и только потом дочерняя запись может быть вставлена с корректным внешним ключом. Но при удалении сначала должна быть удалена дочерняя запись, а уже потом родительская. Обеспечение такой последовательности в коде вручную может быть чревато возникновением ошибок, особенно с учетом более нормализованной модели данных в Microsoft Dynamics AX 2012. Кроме того, операции с данными зачастую распределены между различными ветвлениями в коде. Все это приводит к увеличению числа блокировок и удлинению транзакций в базе данных.

Для помощи в решении этих проблем Microsoft Dynamics AX 2012 предоставляет программную концепцию, называющуюся «Единица работы» (Unit of Work). Единица работы – это, по сути, коллекция операций с данными, выполняющаяся над связанными данными. Код приложения устанавливает отношения между данными в памяти, изменяет данные, регистрирует запросы на выполнение операций в инфраструктуре Единицы работы, а затем выполняет запрос к этой инфраструктуре на выполнение всех зарегистрированных операций с данными в базе данных в виде одного блока. Основываясь на отношениях, установленных между сущностями в данных, находящихся в памяти, инфраструктура Единицы работы определяет корректную последовательность выполнения запрошенных операций и заполняет ссылки по внешним ключам, если это необходимо.

Ниже приводится пример использования Единицы работы.

```
public static void fmRentalAndRentalChargeInsert()
{
    FMTruck truck;
    FMRental rental;
    FMRentalCharge rentalCharge;
    FMCustomer customer;
    UnitofWork uoW;

    // Заполнение данных аренды и платы за аренду в единице работы.
    // Для одной записи аренды есть три типа записей платы за аренду.

    // Получение клиента и грузовика, который клиент арендует
    // На эти записи ссылается запись аренды

    select truck where truck.VehicleId == 'co_wh_tr_1';
    select customer where customer.DriverLicense == 'S468-3184-6541';

    uoW = new UnitofWork();
    rental.RentalId = 'Redmond_546284';
```

```
    // Это связывает запись аренды с записью грузовика. Во время ставки в записи
    // аренды будет содержаться корректная ссылка по внешнему ключу на запись грузовика.

    rental.fmVehicle(truck);

    // Это связывает запись аренды с записью клиента. Во время ставки в записи
    // аренды будет содержаться корректная ссылка по внешнему ключу на запись клиента.

    rental.fmCustomer(customer);
    rental.StartDate = DateTimeUtil::newDateTime(1\1\2008,0);
    rental.EndDate = DateTimeUtil::newDateTime(10\1\2008,0);
    rental.StartFuelLevel = 'Full';

    // Создание записи аренды с использованием для сохранения единицы работы.
    // Единица работы создает копию записи.

    uoW.insertonSaveChanges(rental);
    uoW.saveChanges();
}
```

Важно понимать, что Единица работы при выполнении методов регистрации копирует изменения в данных в ее собственные буферы. После этого исходный буфер отвязывается от Единицы работы. Любые изменения, сделанные в табличном буфере после этого, уже не будут учтены Единицей работы. Если вы хотите сохранить эти изменения с помощью Единицы работы, то должны снова вызвать соответствующий метод регистрации.

Когда вы регистрируете несколько изменений одной и той же записи в Единице работы, последние зарегистрированные изменения затирают все предыдущие. В приведенном выше примере код выполняется на сервере, потому что Единица работы привязана к серверу и ее экземпляр не может быть создан на клиенте. Движок форм Microsoft Dynamics AX в своей внутренней реализации использует инфраструктуру Единицы работы для выполнения операций с данными над источниками данных формы, где несколько источников данных напрямую объединены между собой. Такие сценарии не работали в предыдущих версиях Microsoft Dynamics AX. Когда движок форм использует инфраструктуру Единицы работ, она недоступна для кода X++.

Инфраструктура данных с датой вступления в силу

Во многих бизнес-сценариях требуется отслеживать, как данные изменялись с течением времени. Некоторые из требований включают возможность просмотра значения записи в том виде, в каком она была на определенную дату в прошлом, просмотр значения в настоящем времени или возможность ввода записи, которая вступит в силу в будущем. Типичный пример – это данные сотрудника в приложении, используемом службой персонала компании. Вот некоторые вопросы, на которые такое приложение могло бы помочь ответить.

- Какую должность данный сотрудник занимал на определенную дату?
- Какой в настоящее время оклад у сотрудника?
- Какова актуальная контактная информация сотрудника?

В дополнение к ответам на эти вопросы также может быть требование позволять пользователям вводить новые данные и изменть существующие. К примеру, пользователь мог бы изменить контактную информацию для сотрудника и сделать ее актуальной с 15 сентября 2014 года. Такие данные часто называют «данными с датой вступления в силу» (date-effective). Microsoft Dynamics AX 2012 поддерживает в базе данных создание и управление данными с датой вступления в силу. Инфраструктура данных с датой вступления в силу предоставляет целый набор возможностей, которые включают поддержку для моделирования сущностей, программный доступ для выборки и обновления данных с датой вступления в силу, поддержку среды времени выполнения для обеспечения согласованности данных и поддержку пользовательского интерфейса. Ключевая функциональность Microsoft Dynamics AX, такая как Глобальная адресная книга, и модули, такие как Управление персоналом, широко используют таблицы данных с датой вступления в силу в своих моделях данных.

Реляционное моделирование сущностей с датой вступления в силу

Microsoft Dynamics AX предоставляет поддержку моделирования сущностей с датой вступления в силу на уровне таблиц. Свойство *ValidTimeStateFieldType* таблицы указывает, содержит ли таблица данные с датой

вступления в силу. Эта информация хранится в метаданных таблицы и используется во время выполнения.

На рис. 17-15 показана таблица *DirPersonName*, которая используется для отслеживания истории имени респондентов. Таблица содержит данные с датой вступления в силу, потому что свойство *ValidTimeStateFieldType* установлено в *UtcDateTime*. Когда вы устанавливаете это свойство на таблице, инфраструктура данных с датой вступления в силу автоматически добавляет поля *ValidFrom* и *ValidTo*, которые необходимы для каждой таблицы, содержащей данные с датой вступления в силу. Тип данных этих полей зависит от значения, выбранного для свойства *ValidTimeStateFieldType*. Доступны два типа данных.

- ***Date.*** Отслеживание ведется на уровне дней. Записи вступают в силу, начиная с даты *ValidFrom* и заканчивая датой *ValidTo*.
- ***UtcDateTime.*** Отслеживание ведется на уровне секунд. В этом случае несколько записей могут вступать в силу в течение одного и того же дня.

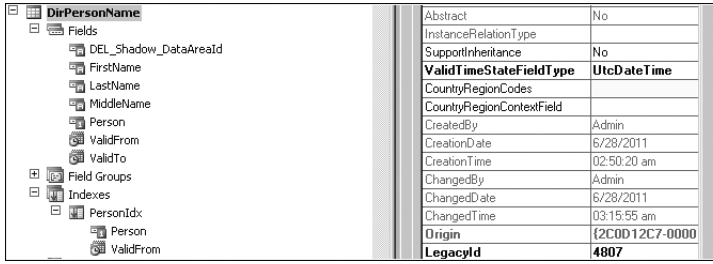


Рис. 17–15. Таблица *DirPersonName* со свойством *ValidTimeStateFieldType*, установленным в *UtcDateTime*

В дополнение к этим полям, которые должны быть на каждой таблице с данными, имеющими дату вступления в силу, на такой таблице еще должен быть хотя бы один альтернативный ключ, реализованный в виде уникального индекса. В инфраструктуре данных с датой вступления в силу такой альтернативный ключ называется *validtimestate* (ключ времени действия данных) – он используется для реализации семантики периодов времени, которые начинают работать для таблиц, содержащих данные с датой вступления в силу. Ключ *validtimestate* должен содержать поле *ValidFrom* и как минимум еще одно поле, отличное от *ValidTo*. На *validtimestate* есть дополнительное свойство, указывающее, допустимы ли в данных разры-

вы (отсутствующие записи для того или иного периода времени). На рис. 17-16 таблица *DirPersonName* используется для отслеживания изменений в поле *Person*. Ключ *validtimestate* содержит поля *Person* и *ValidFrom*. Когда свойство *ValidTimeStateKey* установлено для индекса в значение *Yes*, индекс также должен быть уникальным и являться альтернативным ключом.

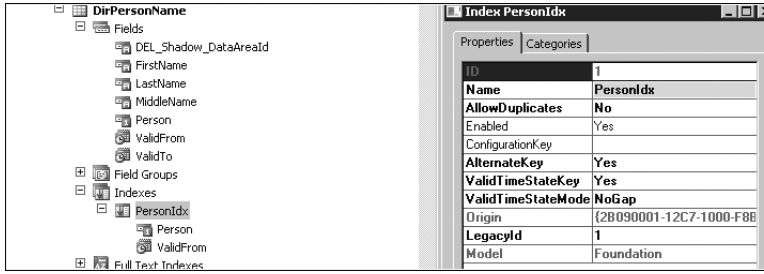


Рис. 17–16. Индекс *validtimestate* на таблице *DirPersonName*

Записи в табл. 17-2 отражают изменения имени двух людей на протяжении определенного промежутка времени. На таблице должен быть уникальный индекс со следующими полями: *Person* и *ValidFrom*. Поле *ValidTo* может быть частью индекса, но это необязательно. На этом индексе свойство *ValidTimeStateKey* установлено в *Yes*. Поскольку целью является отслеживание истории респондента, поле, представляющее респондента, также является частью ключа *validtimestate*. Этот ключ позволяет инфраструктуре данных с датой вступления в силу определить поле, для которого отслеживается история.

Табл. 17–2. Отслеживание изменений имени с течением времени в таблице с данными, имеющими дату вступления в силу

Person	FirstName	MiddleName	LastName	ValidFrom	ValidTo
1	Jim	M	Corbin	02/10/1983 00:00:00	04/16/1984 23:59:59
1	Jim	M	Daly	04/17/1984 00:00:00	12/31/2154 23:59:59
2	Anne		Wallace	04/14/2001 00:00:00	07/04/2005 23:59:59
2	Anne		Weiler	07/05/2005 00:00:00	12/31/2154 23:59:59

В этом сценарии есть требование не допускать разрывов в данных каждого респондента. Для реализации этого требования свойство *ValidTimeStateMode* установлено в *NoGap*. Для других сценариев разрывы в данных могут быть допустимы, что тоже реализуется с помощью свойства *ValidTimeStateMode*. Инфраструктура данных с датой вступления в силу также обеспечивает то, чтобы респондент не мог иметь два имени в один и тот же период времени. Это называется предотвращением наложений в данных. Если поле *ValidTo* записи содержит максимально допустимое значение (*12/31/2154 23:59:59*), то это означает, что срок действия данных записи не истечет.

Поддержка выборки данных

Для получения прикладной бизнес-логики, написанной на X++, и данных, хранящихся в таблице с датой вступления в силу, инфраструктура предлагает три режима.

- **Current (текущий).** Возвращает запись, которая является активной по умолчанию, когда для получения данных из таблицы вы используете оператор *select* или прикладной интерфейс (API).
- **AsOfDate (на дату).** Возвращает запись, которая была активной на указанную дату, или значение параметра *UtcDateTime*. Эта дата или дата/время могут быть в прошлом, настоящем или будущем. Значение поля *ValidFrom* возвращенной записи будет меньше или равно переданному значению, а значение поля *ValidTo* будет больше или равно ему.
- **Range (диапазон дат).** Возвращает запись, которая активная в указанном диапазоне.

Язык X++ поддерживает синтаксис, схожий с синтаксисом Transact-SQL, **используемым в запросах к реляционным базам данных. Инфра**структура данных с датой вступления в силу расширяет этот синтаксис за счет добавления ключевого слова *validtimestate*, которое указывает на тип запроса. Режимы, описанные ранее, транслируются в следующие запросы.

Примечание. Поля *ValidFrom* и *ValidTo* в этих примерах используют тип данных *Date*. Если бы они использовали тип данных *UtcDateTime*, то передаваемые даты должны были бы иметь формат in *UtcDateTime*.

Этот запрос получает для сотрудника *A* текущую информацию о контакте для экстренных случаев. Нет необходимости указывать какие-либо значения для полей *ValidFrom* и *ValidTo* в условии *where*, поскольку среда времени выполнения Microsoft Dynamics AX автоматически их добавит.

```
select * from EmployeeEmergencyContact where EmployeeEmergencyContact.Employee ==
'A';
```

Этот запрос получает запись, которая была актуальна 21 апреля 1986 года:

```
select validtimestate (21\04\1986) * from EmployeeEmergencyContact where
EmployeeEmergencyContact.Employee == 'A';
```

Этот запрос получает все записи для сотрудника *A*, действовавшие в период времени, который передан в запрос:

```
select validtimestate(01\01\1985, 31\12\2154) * from EmployeeEmergencyContact where
EmployeeEmergencyContact.Employee == 'A';
```

X++ также предоставляет Query API для получения данных из таблиц. Этот API для получения различных форм запросов был расширен с помощью следующих методов:

- *validTimeStateAsOfDate(date)*
- *validTimeStateAsOfDateTime(utcdatetime)*
- *validTimeStateDateRange(date)*
- *validTimeStateDateTimeRange(utcdatetime)*

Инфраструктура данных с датой вступления в силу использует эти методы и трансформирует их, добавляя предикаты по полям *ValidFrom* и *ValidTo*, для получения данных, отвечающих критериям запроса.

Поддержка времени выполнения для целостности данных

Данные, хранящиеся в таблицах с поддержкой даты вступления в силу, также должны соответствовать следующим требованиям целостности данных.

- Данные не должны иметь наложений.
- В зависимости от свойства *ValidTimeStateMode* ключа *validtimestate* разрывы либо разрешены, либо не разрешены.

Поскольку данные, хранящиеся в таблице, могут быть добавлены или изменены, инфраструктура данных с датой вступления в силу обеспечивает выполнение этих требований. Данная инфраструктура реализует эти требования за счет изменения других записей согласно следующим правилам.

- Если обновляется *ValidFrom*, получить предыдущую запись и затем обновить ее поле *ValidTo* значением *ValidFrom -1*, чтобы обеспечить отсутствие наложений или разрывов. Если значение *ValidFrom* редактируемой записи меньше, чем значение *ValidFrom* предыдущей записи, то отображается ошибка, поскольку требуется дальнейшее действие по удалению предыдущей записи, чтобы избежать наложения. Инфраструктура данных с датой вступления в силу в ходе изменении записей не удаляет записи автоматически.
- Если обновляется *ValidTo*, получить следующую запись и затем обновить ее поле *ValidFrom* значением *ValidTo+1*, чтобы обеспечить отсутствие наложений или разрывов. Если значение *ValidTo* редактируемой записи больше, чем *ValidTo* следующей записи, то отображается ошибка.
- Инфраструктура данных с датой вступления в силу не разрешает одновременно редактировать поля *ValidTo* и *ValidFrom*.
- Инфраструктура данных с датой вступления в силу не разрешает редактировать любые другие поля, которые являются частью ключа *validtimestate*.
- Когда вставляется новая запись, значение поля *ValidTo* существующей записи обновляется значением *ValidFrom -1* вставленной записи. Новые записи не могут быть вставлены для периодов в прошлом или в будущем, если для соответствующего периода времени уже существуют записи.
- Когда запись удаляется, *ValidTo* предыдущей записи обновляется значением *ValidFrom -1* следующей записи, но только если система требует, чтобы в данных не было разрывов. Если разрывы разрешены, такое обновление не производится.

Режимы обновления записей

Инфраструктура данных с датой вступления в силу позволяет выполнять обычное обновление данных в таблице, отслеживающей дату вступления

в силу данных. Она также предоставляет дополнительные режимы, характерные для тех типов изменений, которым подвергаются данные с датой вступления в силу. Вот три этих дополнительных режима.

- ***Correction (исправление).*** Этот режим аналогичен обычному обновлению данных в таблице. В случае изменения полей *ValidFrom* или *ValidTo* система обновляет дополнительные записи, если это необходимо, чтобы гарантировать отсутствие разрывов или наложения данных.
- ***CreateNewTimePeriod (создание нового периода).*** Инфраструктура данных с датой вступления в силу создает новую запись с измененными значениями и обновляет *ValidTo* значением *ValidFrom -1* новой созданной записи. По умолчанию поле *ValidFrom* новой созданной записи содержит текущую дату для поля типа *Date* и текущее время для поля типа *UtcDateTime*. Этот режим не позволяет редактировать записи в прошлом периоде. Режим скрывает от пользователя то, что данные имеют дату вступления в силу; пользователь редактирует запись как обычно, но внутренне создается новая запись, чтобы отслеживать историю сделанных изменений.
- ***EffectiveBased (на основе даты вступления в силу).*** Этот режим является комбинацией двух других. Записи, относящиеся к прошлому периоду, редактировать запрещено; текущие активные записи редактируются, как в режиме *CreateNewTimePeriod*; записи в будущем периоде обновляются, как в режиме *Correction*.

Режим обновления должен быть указать с помощью вызова метода *validTimeStateUpdateMode(ValidTimeStateUpdate_validTimeStateUpdateMode)* на табличном буфере, который обновляется. Этот метод принимает в качестве параметра значение перечисления *ValidTimeStateUpdate*, которое представляет список различных режимов обновления.

Важно. Если попытаться обновить запись в таблице данных с датой вступления в силу из кода X++, не указав режим обновления, то среда времени выполнения Microsoft Dynamics AX выдаст сообщение об ошибке.

Опыт пользовательского взаимодействия

Когда пользователь обновляет запись в таблице с датой вступления в силу данных, как следствие могут быть обновлены и другие записи. Инфра-

структура данных с датой вступления в силу сначала отображает диалоговое окно, где информирует пользователя о дополнительных изменениях и запрашивает подтверждение пользователя на выполнение действия. Исходные действия пользователя (до подтверждения) на самом деле не приводят к изменению данных. После подтверждения пользователем необходимости обновить данные происходит перечитывание всех обновленных записей, чтобы отобразить изменения в пользовательском интерфейсе.

Поддержка запросов полнотекстового поиска

В Microsoft Dynamics AX 2012 есть возможность выполнять запросы полнотекстового поиска в базе данных. Функциональность полнотекстового поиска предоставляется средствами SQL Server и позволяет выполнять лингвистический поиск в текстовых данных, хранящихся в базе данных. Более подробную информацию вы можете найти в разделе «Full-Text Search (SQL Server)» по адресу: *http://msdn.microsoft.com/en-us/library/ms142571.aspx*.

Microsoft Dynamics AX предоставляет возможность создать на таблице индекс полнотекстового поиска. С помощью новых методов класса *Query* вы можете писать запросы, использующие такой индекс. На рис. 17-17 показан индекс полнотекстового поиска, который создан в AOT на таблице *VendTable*.

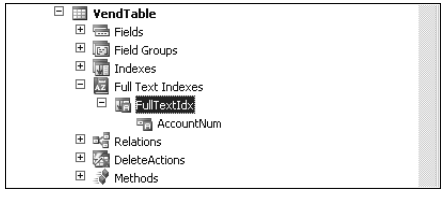


Рис. 17–17. Индекс полнотекстового поиска на таблице *VendTable*

На одной таблице может быть создан только один индекс полнотекстового поиска, в который могут быть включены только поля, имеющие тип данных *string*. При синхронизации таблицы соответствующий индекс полнотекстового поиска создается в базе данных. Microsoft Dynamics AX также требует, чтобы таблица была из группы Main или Group. Вы не можете создавать индекс полнотекстового поиска для временных таблиц. Следующий пример показывает, как можно выполнять запросы полнотекстового поиска с использованием Query API:

```
Query q;
QueryBuildDataSource qbds;
QueryBuildRange qbr;
QueryRun qr;
VendTable vendTable;

q = new Query();
qbds = q.addDataSource(tablenum(VendTable));
qbr = qbds.addRange(fieldnum(VendTable, AccountNum));
qbr.rangeType(QueryRangeType::FullText);
qbr.value(queryvalue('SQL'));
qr = new QueryRun(q);

while (qr.next())
{
    vendTable = qr.get(tablenum(VendTable));
    print vendTable.AccountNum;
}
```

Значение перечисления *QueryRangeType::FullText*, переданное методу *rangeType*, приводит к тому, что уровень доступа к данным генерирует запрос полнотекстового поиска в базе данных, при этом Microsoft Dynamics AX использует в запросах ключевое слово *FREETEXT*, предоставляемое SQL Server. Для предыдущего примера будет сгенерирован такой запрос Transact-SQL:

```
SELECT T1.ACCOUNTNUM,T1.INVOICEACCOUNT,...
FROM VENDTABLE T1 WHERE (((PARTITION=?) AND (DATAAREAID=?)) AND
(FREETEXT(ACCOUNTNUM,?))) ORDER
BY T1.ACCOUNTNUM
```

Более подробную информацию о ключевом слове *FREETEXT* вы можете найти в статье MSDN «Querying SQL Server Using Full-Text Search» по адресу: *http://msdn.microsoft.com/en-us/library/ms142559.aspx*.

Вы также можете использовать в полнотекстовых запросах расширенный синтаксис *range*, как показано в следующем примере. Ключевое слово *freetext* указывает, какой уровень доступа к данным должен сгенерировать запрос полнотекстового поиска.

```
qbrCustDesc = qsbdCustGrp.addRange(fieldnum(VendTable, AccountNum));
qbrCustDesc.value('((AccountNum freetext "bike") || (AccountNum freetext "run"))');
```

Прикладной программный интерфейс QueryFilter

Любимый вопрос по Transact-SQL на интервью – это попросить кандидата объяснить разницу между выражениями *on* и *where* в операторе *select*, который включает объединение таблиц. Вы можете найти длинную версию ответа в MSDN, где рассказывается о различных логических фазах вычисления запроса. Короткий ответ состоит в том, что для inner join никакой разницы между ними нет; для outer join записи, не удовлетворяющие условиям выражения *on*, включаются в результирующую выборку, а записи, не удовлетворяющие условиям выражения *where*, не включаются.

В связи с этим вы можете спросить, когда использовать *on*, а когда – *where*. Проиллюстрировать это поможет пример. Следующий полиморфный запрос находит все экземпляры *DirParty* с именем *John*. Тут используются два типа предикатов: первый сопоставляет отдельные записи с их соответствующим базовыми или производным типом:

```
<baseTable>.recID == <derivedTable>.recid.
```

Второй предикат сопоставляет имя:

```
DirPartyTable.name == 'John'
```

Возьмем первый предикат: когда вы выбираете определенную запись из корневой таблицы, для нее может иметься соответствующая запись в любой из производных таблиц. Поскольку цель состоит в выборке полного набора данных для всех типов, то вам бы не хотелось отбрасывать запись только потому, что для нее нет соответствия в одной из производных таблиц, следовательно, нужно использовать выражение *on*. В случае второго предиката вы хотите выбрать лишь те записи, которые ему удовлетворяют, следовательно, нужно использовать выражение *where*. Соответствующий Transact-SQL запрос выглядит примерно так:

```
SELECT * FROM DIRPARTYTABLE T1 LEFT OUTER JOIN DIRPERSON T2 ON (T1.RECID=T2.
RECID) LEFT OUTER
JOIN DIRORGANIZATIONBASE T3 ON (T1.RECID=T3.RECID) LEFT OUTER JOIN
DIRORGANIZATION T4 ON (T3.
RECID=T4.RECID) LEFT OUTER JOIN OMINTERNALORGANIZATION T5 ON (T3.RECID=T5.
RECID) LEFT OUTER JOIN
OMTEAM T6 ON (T5.RECID=T6.RECID) LEFT OUTER JOIN OMOPERATINGUNIT T7 ON (T5.
RECID=T7.RECID) LEFT
OUTER JOIN COMPANYINFO T8 ON (T5.RECID=T8.RECID) WHERE (T1.NAME='john')
```

Вы можете заметить, что выражение *on* указано сразу после объединения таблиц, а выражение *where* указано в конце запроса после всех объединений таблиц и выражений *on*. Это соответствует порядку, в котором вычисляется запрос. Предикаты выражения *where* вычисляются после того, как будут обработаны все объединения. Следующий X++-оператор *select* генерирует схожий запрос.

```
static void Job3(Args _args)
{
    SalesTable so;
    SalesLine sl;

    select so where so.CustAccount == '1101'
        outer join sl where sl.SalesId == so.SalesId;
}
```

Соответствующий Transact-SQL запрос выглядит примерно так:

```
SELECT * FROM SALESTABLE T1 LEFT OUTER JOIN SALESLINE T2 ON ((T2.
DATAAREAID=N'ceu') AND (T1.
SALESID=T2.SALESID)) WHERE ((T1.DATAAREAID=N'ceu') AND (T1.
CUSTACCOUNT=N'1101'))
```

Тут есть некоторое смешение: используется ключевое слово *where*, но оно идет после каждого объединения таблицы. Так где же должен располагаться предикат в Transact-SQL? Если вы воспользуетесь трассировкой операторов SQL, то увидите, что для outer join предикат появляется в выражении *on*, а для inner join – в выражении *where*. Чтобы понять это, запомните: *where* в X++ на самом деле соответствует выражению *on* в Transact-SQL. Поскольку для inner join нет разницы между *on* и *where*, среда времени выполнения Microsoft Dynamics AX просто перемещает выражение *where*. Модель программирования Query в X++ ведет себя также: условия в Query, которые вы указываете с помощью выражения *QueryBuildRange*, попадают в выражение *on*.

Так как же вам указать предикаты в выражении *where*? Для запроса *select* в X++ вы могли бы присоединить эти предикаты *where* к одной из таблиц, объединенных с помощью inner-join. Другой способ заключается в добавлении этих предикатов к выражению *where* для первой таблицы, если все другие таблицы объединяются по outer-join. В случае с моделью программирования Query решение сложнее, потому что вы не можете указать *QueryBuildRange* на другом источнике данных без использования синтаксиса расширенного query range. Чтобы решить эту проблему, в

Microsoft Dynamics AX 2012 был добавлен программный интерфейс *QueryFilter*.

Поскольку выражение *where* вычисляется на уровне запроса после того, как были вычислены все объединения, программный интерфейс *QueryFilter* также доступен на уровне запроса. Вы можете ссылаться на любой источник данных запроса, не являющийся частью подзапроса exists/notexists. Ниже приводится пример использования *QueryFilter*.

```
public void setFilterControls()
{
    Query query = fmvehicle_qr.query(); // Используем запрос QueryRun, чтобы фильтр
можно было сбросить
    QueryFilter filter;
    QueryBuildRange range;
    boolean useQueryFilter = true; // Измените на false, чтобы увидеть QueryRange в outer
join

    if (useQueryFilter)
    {
        filter = this.findOrCreateQueryFilter(
            query,
            query.dataSourceTable(tableNum(FMVehicleMake)),
            fieldStr(FMVehicleMake, Make));
        makeFilter.text(filter.value());
    }
    else
    {
        // Необязательный код, демонстрирующий разницу в поведении
        // QueryFilter и QueryRange
        range = SysQuery::findOrCreateRange(
            query.dataSourceTable(tableNum(FMVehicleMake)),
            fieldNum(FMVehicleMake, Make));
        makeFilter.text(range.value());
    }
}

public QueryFilter findOrCreateQueryFilter(
    Query _query,
    QueryBuildDataSource _queryDataSource,
    str _fieldName)
{
    QueryFilter filter;
    int i;
```

```
    for(i = 1; i <= _query.queryFilterCount(); i++)
    {
    filter = _query.queryFilter(i);
    if (filter.dataSource().name() == _queryDataSource.name() &&
       filter.field() == _fieldName)
        {
            return filter;
        }
    }

    return _query.addQueryFilter(_queryDataSource, _fieldName);
}
```

QueryFilter использует схожие правила группировки, касающиеся того, как отдельные предикаты объединяются с помощью операторов *AND* или *OR*. Сначала объекты *QueryFilter* группируются по источнику данных запроса, и результаты объединяются с использованием оператора *AND*. Затем, в рамках группы для того или иного источника данных запроса, те же правила применяются к *QueryRange*: сначала предикаты для одного и того же поля используются оператор *OR*, а затем они объединяются с использованием оператора *AND*.

Если вы выполните следующий код X++ и посмотрите на трассировку Transact-SQL, то увидите CROSS JOIN в операторе Transact-SQL. Вы можете подумать, что в нем не хватает условий объединения и что выполняется картезианское соединение двух таблиц, но на самом деле условие объединения находится в выражении *where*. Cross join, подобный тому, который вы увидели, эквивалентен inner join с условиями объединения. Необходимость в cross join вызвана тем, что две таблицы должны быть указаны перед таблицами, объединяемыми с помощью outer join, поскольку они появляются в условиях объединения по outer join, а Transact-SQL не позволяет вам ссылаться на таблицы до того, как они появятся в тексте запроса.

```
static void CrossJoin(Args _args)
{
    CustTable cust;
    SalesTable so;
    SalesLine sl;

    select generateonly cust where cust.AccountNum == '*10*'
        join so where cust.AccountNum == so.CustAccount
            outer join sl where so.SalesId == sl.SalesId;
```

```
    info(cust.getSQLStatement());
}
```

Соответствующий Transact-SQL запрос выглядит примерно так:

```
SELECT * FROM CUSTTABLE T1 CROSS JOIN SALESTABLE T2 LEFT OUTER JOIN SALESLINE
T3 ON (((T3.
PARTITION=?) AND (T3.DATAAREAID=?)) AND (T2.SALESID=T3.SALESID)) WHERE (((T1.
PARTITION=?) AND
(T1.DATAAREAID=?)) AND (T1.ACCOUNTNUM=?)) AND (((T2.PARTITION=?) AND (T2.
DATAAREAID=?)) AND (T1.
ACCOUNTNUM=T2.CUSTACCOUNT))
```

Разделы данных

В предыдущих версиях Microsoft Dynamics AX для реализации границы безопасности, разделяющей данные, использовалось табличное поле *DataAreaId*. Оно также использовалось для определения юридических лиц через концепцию компании. Как часть изменений организационной модели и нормализации данных в Microsoft Dynamics AX 2012 большое число сущностей, таких как Продукты и Субъекты, прежде хранившихся в каждой компании, теперь стали общими (за счет использования глобальных компаний tables). Это было сделано главным образом для обеспечения возможности разделения данных между юридическими лицами, чтобы избежать несоответствия данных в разных компаниях.

Однако в некоторых внедрениях Microsoft Dynamics AX данные не предполагается делать общими для всех юридических лиц. В таких внедрениях поле *DataAreaId* используется в основном как граница безопасности, разделяющая данные по разным компаниям. Такие компании-клиенты хотят разделить между собой стоимость внедрения, развертывания и поддержки Microsoft Dynamics AX, но у них при этом нет общих данных или бизнес-процессов. Также существуют холдинговые компании, которые растут за счет покупки независимых компаний (становящихся дочерними), однако данные и бизнес-процессы не разделяются между этими дочерними компаниями.

Microsoft Dynamics AX 2012 R2 предоставляет решение, удовлетворяющее таким требованиям. В этой версии Microsoft Dynamics AX используется концепция разделения данных за счет добавления в таблицы базы данных поля *Partition*. Это позволяет по-настоящему поделить данные между отдельными разделами. Когда пользователь входит в Microsoft Dynamics AX, он всегда работает в контексте определенного раздела. Эта си-

стема позволяет обеспечить то, чтобы были видны данные только из определенного раздела, и чтобы все бизнес-процессы выполнялись в контексте этого раздела.

Управление разделами

Таблица Partitions содержит список всех разделов, которые определены в системе. Во время установки Microsoft Dynamics AX создает раздел по умолчанию с названием *initial*. Системный администратор может создать дополнительные разделы с помощью формы Разделы из модуля Администрирование системы.

Разработка с учетом разделов

Во все таблицы в AOT было добавлено свойство *SaveDataPerPartition*. По умолчанию оно установлено в значение *Yes*, и изменить это значение нельзя. Его изменение доступно только в случае, если свойство *SaveDataPerCompany* установлено в *No* и таблица помечена как системная, либо если таблица относится к группе Framework. Эти проверки были добавлены, чтобы все таблицы с данными приложения хранились по разделам. Только определенные таблицы, используемые ядром, могут содержать общие данные для всех разделов.

Во всех таблицах, у которых свойство *SaveDataPerPartition* установлено в *Yes*, в метаданных есть системное поле *Partition*. В базе данных – это поле *PARTITION* с типом *int64*. Это суррогатный внешний ключ, связанный с полем *RECID* таблицы *PARTITION* и всегда содержащий значение, соответствующее одной из записей таблицы *PARTITION*, по умолчанию – *RECID* раздела *initial*. Ядро добавляет поле *PARTITION* во все индексы на таблицах, которые хранят данные в разрезе разделов, за исключением индекса по *RECID*.

Работа с учетом разделов

Ядро Microsoft Dynamics AX заполняет поле *Partition* и фильтрует выборки данных по нему, используя информацию о разделе, указанную для текущей сессии. Различные операции с базой данных, предоставляемые Microsoft Dynamics AX, имеют следующую функциональность.

- **Операторы *select*.** Все операторы *select* автоматически фильтруются с использованием текущего раздела сессии. Генерируемый оператор Transact-SQL всегда содержит поле *Partition* в выражении *WHERE*.

- **Операторы *insert*.** Вставляемая запись в поле *Partition* всегда содержит значение раздела из текущей сессии. Microsoft Dynamics AX выводит сообщение об ошибке, если код приложения пытается установить значение этого поля, отличающееся от раздела в текущей сессии.
- **Операторы *update*.** Все обновления выполняются в текущем разделе. Обновлять поле *Partition* не разрешается.

С учетом этой функциональности вам в общем случае не придется писать код для работы с полем *Partition*; исключением тут является любой код, выполняющий прямые запросы Transact-SQL. В этом случае обработка поля *Partition* автоматически не осуществляется, и код, выполняющий прямые SQL-запрос, должен сам позаботиться о том, чтобы выражение *WHERE* содержало предикат с номером раздела из текущей сессии.

Для обеспечения строгой изоляции данных инфраструктура не предоставляет возможности программно изменять текущий раздел во время выполнения. Для изменения раздела необходимо создать новую сессию, у которой установлен другой код раздела. Некоторые компоненты инфраструктуры, такие как Установка и Пакетные задания, используют метод *runAs*, создающий новую сессию для выполнения кода в другом разделе. Это не является общеупотребимым подходом и не должно использоваться в программной логике, не относящейся к инфраструктуре. Аналогично, инфраструктура не позволяет выполнять операции базы данных, затрагивающие несколько разделов. Это поведение отличается от функциональности cross-company, позволяющей выполнять операции с базой данных, затрагивающие несколько юридических лиц.