بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

آموزش مجتمع سازی CLR در SQL Server 2012 قسمت اول

مدرس : مهندس افشین رفوآ

رمز فایل :tahlildadeh.com

خلاصه: این مقاله ویژگیهای جدید CLR integration درSQL Server 2005 ، را توصیف می کند و توضیح می دهد که چگونه یک database application developer وarchitect ها می توانند از آنها جهت نوشتن رویه هاprocedure()، functionها، وtrigger هایی که توسط کاربر تعریف شده اند، استفاده کنند. و همچنین typeها و aggregateهای جدیدی را تعریف می کنند.

ما، برنامه نویسی بر پایه CLR را با مدلهای برنامه نویسی از قبیل Transact-SQLو extended stored procedures، را که در SQL Server ساپورت می شوند، مقایسه ، و نقاط قوت و ضعف هر تکنیک را برجسته می کنیم، و لیستی از رهنمودهای سطح بالا در مورد چگونگی انتخابalternative های در دسترس برای برنامه نویسی را در اختیار می گذاریم. همچنین، گروهی از نمونه کدهایی را که خصوصیات CLR integration را نشان میدهند، نیز در اختیار می گذاریم.

مقدمه

Microsoft SQL Server 2005به طور چشمگیری مدل database programming را با میزبانی از Microsoft .NET Framework 2.0 Common Language Runtime (CLR)بهبود می بخشد. این موضوع، برنامه نویسان را قادر به نوشتن رویه ها (procedure)، functionها، وtrigger ها در هر زبانCLR ، مخصوصاً Microsoft Visual Basic .NET, Microsoft Visual C# و Microsoft Visual C++ می کند . همچنین به برنامه نویسان اجازه می دهد database را باtype ها وaggregateهای جدید، گسترش دهند.

این مقاله چگونگی استفاده بهینه از این تکنولوژی را از نگاه یک database application developer توضیح می دهد، و تکنیکهای CLR integration را با programming language support موجود در SQL Server ، Transact-SQL (T-SQL) و stored procedure های گسترش یافته، مقایسه می کند. این مقاله، مرجعی برای property ها نیست، اطلاعات مربوط به مرجع را می توان در SQL Server 2005 Books Online پیدا کرد. مثالهای این مقاله، CTP آوریل سال ۲۰۰۵ از SQL Server 2005 را در نظر می گیرد.

مخاطبین این مقاله شامل database application developer ها، architect ها و administrator ها می شود. این مقاله، شامل آشنایی موثر با .NET Framework-based programming و database programming می شود.

مرور CLR Integration

آنچه در ادامه می آید، مروریست مختصر بر عملکرد SQL Server که توسط CLR integration فعال شده، و اینکه Visual Studio 2005 ، چگونه این خصوصیات را ساپورت می کند.

نصب (deployment) دستی

ثبت و اجرای کدهای مدیریت شده در database شامل مراحل زیر می شود:

۱. برنامه نویس، یک برنامه مدیریت شده را به عنوان مجموعه ای از تعاریف class می نویسد. Routines های SQL Server ، stored procedure ها، function ها، به عنوان متدهای (static یا shared در Microsoft Visual Basic .NET) یک کلاس نوشته می شوند Type . و aggregate هایی که کاربر تعریف کرده به عنوان کلاسهای کلی نوشته می شوند. برنامه نویس کد را compile و یک assembly ایجاد می کند.
۲. Assembly به یک SQL Server database ، یعنی جاییکه در کاتالوگهای سیستم، با استفاده از عبارات DDL ، ذخیره می شود، آپلود می شود.
۳. سپس Object های Transact-SQL (T-SQL) از قبیل routine ها، type ها و aggregate ها ایجاد می شوند و به entry point هایی که قبلاً آپلود شده اند، محدود می شوند. این روند، با استفاده از عبارات CREATE PROCEDURE/FUNCTION/TRIGGER/TYPE/AGGREGATE به دست می آید .
٤. بعد از اینکه routine ها ایجاد می شوند، application ها می توانند آنها را مثل routine های T-SQL مورد استفاده قرار دهند. مثلاً، می توان CLR function ها را از query های T-SQL ، و CLR procedure ها را از یک client application یا از یک T-SQL batch فرا خواند، گویی procedure های T-SQL هستند.

ساختن، نصب(deployment) ، و debug کردن با استفاده از Visual Studio

Visual Studio 2005 از development ، deployment و debug کردن کدهای مدیریت شده در SQL Server 2005 ساپورت می کند. یک SQL Server Project جدید، code template هایی را که شروع نوشتن کدها برای routine ها، type ها و aggregate های CLR-based database را برای برنامه نویسان آسان می

کند، در اختیار می گذارد. همچنین این پروژه به برنامه نویسان اجازه می دهد مرجعهایی (reference)را به assemblyهای دیگری در database اضافه کنند.

زمانیکه یک SQL Server Project ساخته می شود، به یکassembly ، compileمی شود Deploy .کردن پروژه، assembly binaryرا در SQL Server database که مربوط به پروژه است، آپلود میکند. همچنین عملیاتdeployها، routine ، typeها وaggregate هایی را ایجاد می کند که در database assembly که برپایه attributeهای دلخواه (SqlProcedure, SqlFunction, SqlTrigger) موجود در کد است، تعریف می شوند. همچنینdeploy ، کدهای منبع و سمبولهای (debuggingفایلهای (pdb. را که مربوط به اسمبلی هستد، آپلود می کند.

از آنجاییکهdebugging ، بخش مهمی از تجربه یک برنامه نویس برای هر platformاست، SQL Server 2005 وVisual Studio 2005 ، database programmerهایی را با چنین قابلیتهایی در اختیار می گذارند. یک ویژگی کلیدی در debug کردن اشیاء درSQL Server 2005 ، سهولت نصب و استفاده است. اتصال به کامپیوترهایی که SQL Server را run می کنند، ممکن است مانند فرایندهایی که تحت یک سیستم عامل قدیمی اجرا می شوند، debugشوند. نوع اتصال به سروری کهclient ، دارد، عملکرد debugger را تحت تاثیر قرار نمی دهد. هم Tabular Data Stream (TDS) و همHTTP connection ها ممکن است debug شوند. به علاوه، debugکردن، در همه زبانها، کاملاً یکپارچه عمل می کند و به کاربر می دهد از T-SQL به متدهای CLRیا بالعکس، دست یابد.

CLRوAlternative هایش

در ارزیابی استفاده ازCLR integration ، یک برنامه نویس باید آنرا با optionهای دردسترس دیگر مقایسه کند. هدف ما در اینجا، فراهم کردن پایه ای برای این مقایسه است، طوری که بتوان آنرا مقابل تکنیکهای برنامه نویسی کنونی از قبیل Transact-SQL, extended stored procedures :وcode in the middle-tier ، قرارداد. در این قسمت، ما رویroutine هایی که کابر تعریف می کند، تمرکز خواهیم کرد.

CLRدر برابرTransact-SQL

Transact-SQL (T-SQL)یک زبان برنامه نویسی ذاتی (native) است که توسط SQL Serverساپورت می شود. مثل اکثر نسخه هایSQL ، شامل ویژگیهای تغییر داده ها (data manipulation features) و ویژگیهای تعریف داده ها (data definition features) می شود data manipulation features .را می توان به دو گروه کلی تقسیم بندی کرد:

یک) declarative query language تشکیل شده از عبارات (SELECT/INSERT/UPDATE/DELETEو یک procedural language (WHILE, assignment, triggers, cursorsو غیره). اگر بخواهیم کلی بگوییم، CLR supportدرSQL Server ، جایزگزینی را برای بخش روندی (T-SQL (procedural portion در اختیار می گذارد.

حتی بدون CLR support نیز، مهم است تشخیص دهیم که database application ها باید تا آنجا که ممکن است از declarative query language استفاده کنند. این قسمت از زبان، قادر به استفاده از قدرت query processor است که می تواند به نحو احسن، operation bulk ها را اجرا و بهینه کند . Database application ها باید فقط از procedural programming برای بیان منطق (logic) که در query language قابل بیان (express) نیست، استفاده کند.

تمامی اینها، با CLR support در SQL Server ، نیز صدق می کند: از CLR نباید برای نوشتن procedural code که با استفاده از ویژگیهای زبان T-SQL ، قابل نوشتن هستند، استفاده کرد. برنامه نویسان باید آگاه باشند که تعدادی enhancement های مهم T-SQL query language در SQL Server 2005 وجود دارند که قدرت T-SQL query language را افزایش می دهند، و باید مطمئن شوند که از همه مزایای آنها قبل از نوشتن procedural code بهره می برند، چه در CLR و چه در جای دیگر. بعضی از این ویژگیهای اضافه شده عبارتند از:

- قابلیت نوشتن Query های بازگشتی جهت traverse کردن سلسله مراتب (hierarchy) بازگشتی در یک جدول.
- Function های تحلیلی (analytical) جدید از قبیل RANK و ROW_NUMBER که اجازه درجه بندی ردیفها (RANKING ROW) را در لیست خروجی می دهد .
- اپراتورهای نسبی (relational) جدید از قبیل PIVOT, APPLY, INTERSECT, EXCEPT و UNPIVOT.

برنامه نویسان باید به CLR به عنوان جایگزینی کارآمد برای منطق (logic) که در query language قابل بیان نیست، نگاه کنند.

بیایید نگاهی به به برخی از سناریوهایی که در آنها برنامه نویسی بر پایه CLR میتواند قدرت T-SQL query language را تکمیل کند بیاندازیم. اغلب، ما نیاز به قراردادن procedural logic درون یک Query که میتوان function نامید، داریم. این، شامل موقعیتهای زیر می شود:

- اجرای محاسبات پیچیده (که باید با استفاده از procedural logic بیان شوند) روی value های ذخیره شده در جداول database. این کار مستلزم ارسال خروجی این محاسبات به client ، یا استفاده از آنها برای فیلتر کردن دسته ای از ردیفها (row) که به client ارسال می شوند، است؛ مانند مثال زیر:

```
SELECT (<column-name>,...)
FROM <table>
WHERE (<column-name>,...) = ...
```

- استفاده از procedural logic جهت ارزیابی خروجی که در قسمت FROM در یک عبارت SELECT یا DML، query می شوند (DML) . مخفف Data Manipulation Languaga ، به معنای زبان تغییر داده ها به دستورات INSERT، UPDATE، و DELETE است)

SQL Server 2000، (Function)تابع های T-SQL را معرفی کرد که این سناریو هارا فعال می کند. باSQL Server 2005، این تابع ها، با استفاده از زبانهایCLR ، راحتتر نوشته می شوند. زیرا برنامه نویسان می توانند ازlibrary های بسیار گسترده درNET Framework API. ، بهترین استفاده را ببرند. به علاوه، زبانهای برنامه نویسیCLR ، ساختارهای قوی data از قبیلarray) ها، list)ها، و غیره) رادر اختیار می گذارد که در T-SQL وجود ندارد، و می توانند به علت داشتن مدلهای متفاوت execution در CLR وT-SQL ، بهتر اجرا شوند.

بطور کلی، تابعها کاندیدهای خوبی برای نوشته شدن، با استفاده از CLR هستند؛ زیرا نیاز به دسترسی به databaseاز درون یک تابع به ندرت پیش می آید؛value های databaseمعمولاً تحت عنوان پارامتر ارسالی (argument)ارسال می شوند که درtask های محاسبه ای، بهتر از T-SQL هستند.

CLRjIData Access

حالا نگاهی به CLR به عنوان یک option برای نوشتنroutine هایی که data access اجرا میکند، می اندازیم، هم از منظر مدل برنامه نویسی (programming model) وهم ازمنظر اجرا.(performance)

درT-SQL ، عبارات query language از قبیلSELECT ، INSERT، UPDATEوDELETE ، فقط در procedural codeقرا می گیرند. ار سوی دیگر، کد مدیریت شده، از ADO.NET data access provider برای (SqlClient) SQL Server استفاده می کنند. در این روش، همه عبارات query language توسط stringهای دینامیک که به متدها و خصوصیات (propert) در ADO.NET API تحت عنوان پارامتر ارسالی (argument)ارسال می شوند.

به خاطر این تفاوت، data access codeنوشته شده که از CLR استفاده می کند، می تواند از T-SQL طولانی تر باشد. مهمتر اینکه، از آنجاییکه عبارات SQL درstring های دینامیک رمزگذاری (encode) می شوند، تا وقتی که اجرا شوند، compileو اعتباردهی (validate) نمی شوند ؛ که هم روی debug کردن کد و هم روی اجرایش (performance) تاثیر می گذارد. اما، مدل برنامه نویسی databaseبا ADO.NET بسیار شبیه مدلی است که در client یاmiddle-tier ها استفاده شده، که هم move کردن کد بینtier ها و هم استفاده از مهارتهای موجود را آسانتر می کند.

یه یاد داشته باشید که مدلهای برنامه نویسی بر پایه -T-SQL و هم بر پایهCLR ، از یک SQL query languageاستفاده می کنند؛ تنهاprocedural portion ها فرق دارند.

همانطور که قبلاً هم ذکر شد، کد مدیریت شده، با در نظر گرفتن اکثر محاسبات رویه ای(procedural) ، از مزیت "اجرا شدن حتمی" در مقایسه با T-SQL بهره می برد؛ اما برایdata-access ، T-SQLمعمولاً بهتر عمل می کند.

بنابراین، یک قانون رایج خوب این است که "کد حساس به محاسبه و منطق-(computation- and logic intensive)، انتخاب بهتری برای پیاده سازی در CLR نسبت به کد حساس به data-access است.

بیایید نگاهی به بعضی از الگوهای معمولی در data-access programming بیاندازیم، که توضیح می دهد T-SQLو برنامه نویسی مدیریت شده(managed programming) ، که از integrated CLR و ADO.NETاستفاده می کند، چگونه در این سناریوها اجرا می شوند.

ارسال خروجی بهClient

سناریوی اول ما، مستلزم ارسال دسته ای از ردیفها (row) به Client بدون consume کردن آنها در سرور است. مثلاً، هیچ ردیفی خارج ازroutine ، navigate نمی شود. باT-SQL ، وارد کردن یک عبارت SELECT در T-SQL procedure، به اندازه ارسال ردیفهای ایجاد شده توسط SELECT بهcleint ، موثر است. با کدهای مدیریت شده، شیSqlPipe ، برای ارسال خروجی به client استفاده می شود T-SQL .و ADO.NET این سناریو را تقریباً به یک صورت اجرا می کنند.

SubmitکردنعباراتSQL

SubmitکردنعباراتSQL از CLR مستلزم traverse کردن لایه های اضافی کد، به منظور switch بین کد مدیریت شده و کدSQL ، است. به همین دلیل، T-SQLدارای مزیتperformance ، هنگام صدور (issue) یک SQL query است. توجه داشته باشید که بعد از submit شدن این عبارت درquery processor ، فرقی در performance که بر پایه منبع عبارت است، وجود ندارد؛ (چه درT-SQL ، چه در کد مدیریت شده). اگر queryپیچیده باشد و ارزیابی اش (evaluate) طول بکشد، آنگاه تفاوتها در performance بین T-SQL و کد مدیریت شده، کم اهمیت خواهد بود. به طور خلاصه، queryهای ساده، overheadلایه های اضافی کد می تواند performance یک procedure مدیریت شده را تحت تاثیر قرار دهد.

احتمال داردstored procedure های معمولی و حساس بهdata-access ، درگیر submit کردن یک سری (sequence)از عبارات SQL شود. اگر عبارات SQL ساده باشند و زیاد طول نکشند تا اجرا شوند، آنگاه فراخواندن overhead از کد مدیریت شده می تواند بر زمان اجرا تسلط پیدا کند؛ چنینprocedure هایی نوشتن در T-SQL را بهتر اجرا می کنند.

Forward-Only, Read-Only Row Navigation)خواندن رکوردها، فقط رو به جلو و فقط خواندنی)

درT-SQL ، forward-only, read-only navigationبا استفاده از یک cursor پیاده می شود. در کدCLR ، این کار با یک SqlDataReader پیاده می شود. معمولاً، مقداری پردازش برای هر قطعه از DATA انجام می شود. اگر این موضوع را نادیده بگیریم، T-SQLیک مزیت دارد، زیرا Row Navigation که از CLR استفاده می کند، کمی آهسته تر از T-SQL است. اما، از آنجاییکه CLR در هر پردازشی که روی data انجام می شود، به طور برجسته ای اجرای (performance) بهتری نسبت به T-SQL دارد، CLR performance، هنگامی که مقدار پردازش افزایش می یابد، جای اجرای T-SQL را می گیرد.

به علاوه، شخص هنگام استفاده ازcursor های T-SQL ، باید از پتانسیل امکانات اضافی (latency) آگاه باشد. گرچه بعضی ازquery ها، الزاماً مقداری latency نشان می دهند، زیرا باید خروجی میانی (intermediate) را ظاهر کنند، cursorهای STATIC و KEYSET همیشه خروجی نهایی را در یک جدول موقت قبل از بدست آوردن هر نتیجه دیگری، ظاهر می کنند. یک cursor می تواند یا مستقیماً با STATIC یاKEYSET ، تعریف (declare)شود و یا غیر مستقیماً ، به علت برخی ویژگیهای query وdata ، به یک تبدیل شود CLR . SqlDataReaderهمیشه خروجی را هنگامی که در دسترس هستند و با پرهیز از این latency تولید می کند.

Row-Navigationبا updateها

اگر این مشکل مستلزم update کردن ردیفها (row) برپایه موقعیت کنونی cursor باشد، آنگاه هیچ مقایسه performanceمرتبطی وجود نخواهد داشت. این عملکرد در ADO.NET ساپورت نمی شود و باید از طریق cursorهای قابل update در T-SQL انجام شود. اما به خاطر داشته باشید که معمولاً استفاده از عبارات UPDATEبرای update کردن ردیفهای خالی بهتر است، وmodification هایی را که بر پایه cursor هستند، برای زمانیکه نمی توان با declarative SQL بیان کرد، ذخیره می کند.

خلاصه

در زیر، خلاصه ای از راهنماییهایی که دیده ایم می توانند در انتخاب CLR یا T-SQL استفاده شوند، آورده شده است:

- هر گاه ممکن است، از عباراتSELECT ، INSERT ، UPDATE، و DELETE استفاده کنید. پردازش رویه ای (procedural) و پردازش بر پایه row باید فقط هنگامی که logic با استفاده ازdeclarative languageقابل بیان نیست، استفاده شود.
- اگر رویه (procedure) ،فقط یک wrapper برای فرمانهای declarative T-SQL است، باید در T-SQLنوشته شود.
- اگر رویه(procedure) ، ابتداً مستلزم forward-only, read-only row navigation از طریق یک دسته از خروجی با همان پردازش هر row باشد، احتمالاً استفاده ازCLR ، بازده بیشتری دارد.
- اگر رویه(procedure) ، هم مستلزم data access و هم محاسبه باشد، تقسیم کردن کد رویه ای (procedural code)را در CLR portion که یک T-SQL procedure را جهت اجرای data access یا یک T-SQL procedure را که CLR را جهت اجرای محاسبه فرا می خواند، در نظر بگیرید. راه حل دیگر، استفاده از یک T-SQL batch واحد است که شامل یک سری query است که یک بار از managed codeجهت کاهش تعدادround trip های submit کردن عبارات T-SQL اجرا شده است.

قسمت های بعدی بیشتر به بحث درباره زمان و چگونگی استفاده صحیح از T-SQL و CLR هنگام کار با خروجی می پردازد.

CLRدر مقابل XP ها

در نسخه های قبلیSQL Server ، extended stored procedureهاXP) ها) تنها جایگزین T-SQL بودند که نوشتن کد سمت سرور (server-side code) به آن سبک بسیار سخت بودCLR integration . ، یک جایگزین قویتر برایXP ها در اختیار می گذارد. به علاوه، باCLR integration ، خیلی ازstored procedure ها بهتر بیان می شوند و به آنها، با استفاده ازquery language ، اجازه فراخوانده شدن و تغییر (manipulation) می دهد.

بعضی از مزیتهای استفاده ازCLR procedure ها نسبت بهXP ها عبارتند از:

- کنترل دانه ایadministrator :(Granular control) های SQL Server کنترل کمی روی کارهایی که XPها می تواند یا نمی تواند انجام دهند، دارند. با استفاده از مدلCode Access Security ، یک SQL Server administratorمی تواند یکی از سهpermission bucket های SAFE ، EXTERNAL_ACCESS، یاUNSAFE ، را جهت بکار گیری درجات مختلف کنترل عملیاتهایی که کد مدیریت شده اجازه دارد اجرا کند، تعیین کند.
- قابلیت اعتماد :(Reliability) کد مدیریت شده، مخصوصاً درpermission های SAFE و EXTERNAL_ACCESS، مدل برنامه نویسی قابل اعتماد تری را نسبت بهXP ها در اختیار می گذارد. کدهای مدیریت شدهِ قابل تایید، تضمین می کنند که دسترسی به اشیا از طریقinterface هایی که احتمال اینکه برنامه بهmemory buffer هایی که متعلق به SQL Server است، دسترسی پیدا کند یا آنها را مختل کند، کاهش می دهد.
- Data access:باXP ها، باید اتصالی مستقیم (loop-back connection) به database ، جهت دسترسی به SQL Server database محلی (local) ایجاد شود. به علاوه، این loop-back connectionباید مستقیماً به transaction context های جلسه اصلی (original session) محدود شود تا تضمین کند XP در transaction که در آن فراخوانده می شود، شرکت دارد. کد مدیریت شده CLRمی تواند با استفاده از یک مدل برنامه نویسی طبیعی و کارآمدارتر که از connection و transaction contextحاضر بیشترین بهره را ببرد، به داده های محلی (local data) دسترسی پیدا کند.
- Data Typeهای اضافیAPI :های مدیریت شده ازdata type های جدید (از قبیلXML ، varbinary(max)) و varchar(max)(n)، که در SQL Server 2005 معرفی شدند، ساپورت می کند؛ در حالیکهODS API ها جهت ساپورت از اینtype های جدید، گسترش نیافته اند.
- قابلیت صعود پذیریAPI :(Scalability) هایی که در معرض منابعی (resource) از قبیل حافظه، threadها، و همسان سازی (synchronization) هستند، در بالای SQL Server resource managerپیاده می شوند، و به SQL Server اجازه مدیریت این منابع برای کد CLR را می دهد. در مقابل، SQL Serverهیچ دید یا کنترلی روی منبع استفاده از یک XP ندارد. اگر یکXP ، زمان و حافظه زیادی از CPU را بگیرد، راهی برای شناسایی یا کنترل آن از درون SQL Server وجود ندارد. با کد CLR، SQL Serverمی تواند پی ببرد که یک thread تعیین شده، به مدت طولانی بازگشتی نداشته و task را مجبور به بازگشت می کند، طوریکه بتوان کار دیگری را برنامه ریزی کرد. در نتیجه، استفاده از کد مدیریت شده، قابلیت صعود (scalability) و قوت (robustness) بهتری را در اختیار می گذارد.

همانطور که در بالا اشاره شد، برای data access و ارسال خروجی به client CLR ، routineها بهتر ازXP ها عمل می کنند. برای کدی که مستلزم data access یا ارسال خروجی نیست، مقایسه اجرایXP ها (performance)و کد مدیریت شده، مانند مقایسه کد مدیریت شده با native code است. معمولاً، کد کنترل شده نمی تواند بر اجرای native code در این سناریوها فایق آید. علاوه بر این، یک cost اضافی نیز در زمان انتقال از کد مدیریت شده به کدnative ، هنگام run شدن درون SQL Server وجود دارد، زیرا SQL Server نیاز به انجام book-keeping روی تنظیمات مخصوص thread هنگام انتقال به native code و بالعکس دارد. در نتیجه، XPها می توانند اجرای (running) کد مدیریت شده را درون SQL Server برای مواردی که انتقالهای (transition)مکرر بین کد مدیریت شده و کد native وجود دارد، بهتر انجام دهد.

در اکثرprocedure ها، مزیتهای کد مدیریت شده می تواندprocedure های CLR را جایگزین جذابتری نسبت بهXP ها کند. در مواردی کهperformance ، ابتداً توسط پردازش حساس به محاسبه-computationally) (intensive processingو انتقالات مکرر بین کد مدیریت شده و کد native تعیین شده، مزایای CLR باید در مقابل مزایای اجرای خامXP ها (raw performance) سنجیده شود.

کد درMiddle Tier

گزینه دیگر برنامه نویسان، قرار دادنlogic شان بیرون از database است. این کار به آنها اجازه نوشتن کدشان را در زبانهای انتخابیشان می دهد. با در اختیار گذاشتن یک مدل برنامه نویسی غنی درdatabase ، CLR integrationبه برنامه نویسان امکان منتقل کردن چنینlogic را به database می دهد. البته، این بدین معنا نیست که همه یا اکثر کدها باید به database منتقل شوند.

انتقال logic به database tier می تواند مقدار داده ای را که روی network جاری است کاهش دهد، اما باری اضافی رویresource های باارزش سی پی یوی سرور می گذارد. این موضوع باید قبل از تصمیم گیری درباره قراردادن کد برای یکapplication ، با دقت مد نظر قرار گیرد. ملاحظات زیر می تواند database را تبدیل به code locationارجح کند:

·اعتبار دهی داده ها :(Data validation) نگه داشتن data validation logic درdatabase ، امکان بهتر خلاصه کردن این logic را با داده ها می دهد، و از تکثیر شدن validation logic درdata touch point ها، از قبیلback-end processing ، bulk uploadو update کردن داده ها از middle tier و غیره جلوگیری می کند.

·کاهش ترافیک شبکه :(Network traffic reduction) ممکن است قراردادن logic در database برای taskهای پردازش داده ها که مستلزم پردازش مقدار زیادی داده است، هنگام تولید درصد کمی از آن داده ها، مناسب باشد. نمونه های معمولی شاملapplication های تحلیل داده ها از قبیل demand forecasting)پیش بینی تقاضا) و) scheduling production زمانبندی تولید) بر اساسforecast demand ها و غیره باشد.

البته این ملاحظات بدون CLR integration هم مهم هستند؛ CLR integration فقط به تضمین اینکه انتخاب زبان برنامه نویسی در تعیین موقعیت مناسب کد دخالت نمی کند، کمک می کند.

مثال: زمانبندی تولید(Production Scheduling)

Production scheduling یک کار عادی در کارخانجات صنعتی است. در سطحی بالا، این کار مستلزم ارائه طرحی برای زمان تولید محصولات به منظور برآورده کردن تقاضا و در عین حال، به حداقل رساندن هزینه کل تولید و انبار کردن محصولات است. الگوریتمهای متعددی وجود دارند که پیش بینی تقاضا، هزینه های نگهداری انبار کالا، و هزینه های راه اندازی خط تولید را به عنوان ورودی، و استراتژی تولید را خروجی در نظر می گیرند.

با فرض اینکه پیش بینی تقاضاهای آینده در جدولی در SQL Server ذخیره می شوند، پیاده سازی چنین الگوریتمی دارای مشخصه های زیر است:

۱. مقدار زیادی داده (پیش بینی تقاضا) را ورودی در نظر می گیرد.
۲. نتیجه کمی را تولید می کند، مثلاً تعداد واحدهایی که باید در یک زمان یا زمانهای مشخص تولید شوند.
۳. مستلزم محاسبات زیاد جهت مشتق کردن خروجی از ورودی است.

پیاده سازی چنین الگوریتمی در middle tier اختیاری است، اما انتقال داده های تقاضا ازdatabase ، هزینه زیادی دربر دارد. انجام آن در T-SQL به عنوان یک stored procedure نیز شدنی است، اما عدم وجود data typeهای پیچیده پیاده سازی را مشکل می کند، و performance درT-SQL ، به علت کمیت و پیچیدگی محاسبات لازم، مسئله ساز خواهد بود. البته، مشخصه های اجرا (performance characteristics) بسته به مقدار واقعی داده ها و پیچیدگی الگوریتم، متنوع خواهد بود.

جهت تایید تطابق CLR integration با چنین سناریویی، ما یک الگوریتم برنامه ریزی تولید—الگوریتم Wagner-Whitin —را در نظر گرفتیم و آنرا با استفاده از هم CLR و هم T-SQL پیاده کنیم. همانطور که انتظار می رفت، CLR integrationپیاده سازی خیلی بهتری نسبت به T-SQL داشت. پیاده سازی در C# راحتتر بود، زیرا این الگوریتم ازarray های تک و چند بعدی که در T-SQL وجود ندارند استفاده می کند. روی هم رفته، نسخه CLR چندین orders of magnitude را بهتر از T-SQL پیاده می کند.

بیایید شکل database ساده زیر را که لیست محصولاتی را که تولید می شوند را نگه می دارد، در نظر بگیریم.

جدول t_products:

نام ستون	نوع	شرح
Pid	int	مشخصات اولیه یک محصول
pName	nvarchar(256)	نام محصول
inventoryCost	int	هزینه ذخیره این محصول در هر دوره
startupCost	int	هزینه راه اندازی خط تولید برای شروع تولید محصول

به علاوه، جدول زیر پیش بینی تقاضا برای هر محصول در هر روز را ذخیره می کند. ما فرض می کنیم که ستون pidیک کلید خارجی در جدول t_productsاست.

جدولt_salesForecast:

نام ستون	نوع	شرح
Pid	int	مشخصات محصول
demandDate	nvarchar(256)	روزی که تقاضای برای آن پیش بینی می شود
demandQty	int	پیش بینی تقاضا برای محصول معین شده

ما یک stored procedure ایجاد کردیم که علاوه بر داده های محصول، پارامترهایی را در نظر می گیرد که دامنه تاریخ هایی را که باید برنامه ریزی تولید ارائه کرد، بیان می کند.

اینstored procedure ، یک rowset را با schema در جدول زیر باز می گرداند.

نام ستون	نوع	شرح
product	nvarchar(256)	نام محصول
product	datetime	روز تولید
quantity	int	تعداد محصولاتی که قرار است تولید شوند

نسخه C# این کد، در زیر آورده شده تا نوع ستاریویی را که می تواند از CLR integration سود ببرد نشان دهد:

```
using System;
using System.Data;
using System.Collections;
using System.Data.SqlTypes;
using System.Data.SqlClient;
using Microsoft.SqlServer.Server;

public partial class ProductionScheduler
{
    const int MAXNAME = 256;                    // The maximum name size

    [Microsoft.SqlServer.Server.SqlProcedure]  // Flag as a SQL procedure
    public static void Schedule(SqlDateTime start, SqlDateTime end)
    {
        // Guarantee that we have a valid connection while we run
        using (SqlConnection conn =
            new SqlConnection("context connection=true"))
        {
            conn.Open();                        // open the connection
            SqlPipe pipe = SqlContext.Pipe;     // get the pipe

            // Find all the products in the database with any demand
```

```
            // whatsoever along with data about their production costs.
            // Make sure they are ordered.
            ArrayList items = new ArrayList();
            SqlCommand cmd = new SqlCommand(
                " SELECT DISTINCT tp.pid, pname, startupCost,"
                "     inventoryCost" +
                " FROM t_products tp" +
                " JOIN t_salesForecast ts" +
                " ON tp.pid = ts.pid" +
                " ORDER BY pid",
                conn);
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                items.Add(new Item(
                    reader.GetInt32(0), reader.GetSqlChars(1),
                    reader.GetInt32(2), reader.GetInt32(3)));
            }
            reader.Close();

            // Now get all the production schedule information, ordered
            // by PID and demand date
                "SELECT pid, demandDate, demandQty" +
                " FROM t_salesForecast" +
                "  WHERE demandDate >= @start" +
                "  AND demandDate <= @end" +
                " ORDER BY pid, demandDate",
                conn);
            cmd.Parameters.AddWithValue("@start", start);
            cmd.Parameters.AddWithValue("@end", end);
            reader = cmd.ExecuteReader();

            // Read each section of schedule information into the items.
            reader.Read();
            for (int i = 0; (i < items.Count) && (!reader.IsClosed); i++)
            {
                ((Item)(items[i])).readData(reader);
            }

            // ensure the reader is closed
            if (!reader.IsClosed) reader.Close();


            foreach (Item item in items)
            {
                // Compute the schedule and report it
                item.ComputeSchedule();
                item.OutputSchedule(pipe);
            }
        }
    }

    class Item
    {
        // Information about the product we are scheduling. These will
        // be pulled from the database.
```

```
private int pid;
private SqlChars name;
private int startCost;
private int holdCost;

// Store how many dates we have.
private int size = 0;

// The dates on which we have demand. These are guaranteed to
// be unique by the database, and we will load them in order.
private ArrayList dates = new ArrayList();
// Store what the demand was on each date.
private ArrayList quantities = new ArrayList();

// Our schedule, which we have not yet computed.
int[] schedule = null;

// Set up the metadata for the return
SqlMetaData[] metadata = new SqlMetaData[] {
    new SqlMetaData("product", SqlDbType.NVarChar, MAXNAME),
    new SqlMetaData("period", SqlDbType.DateTime),
    new SqlMetaData("quantity", SqlDbType.Int)
};

public Item(int pid, SqlChars name, int startCost, int holdCost)
{
    this.pid = pid;
    this.name = name;
    this.startCost = startCost;
    this.holdCost = holdCost;
}

/*
 * Read data from the stream until the PID does not match
 * ours anymore. We assume the reader is cued up to our
 * information and we leave it cued to the next item's
 * information UNLESS there is no more information, in which
 * case we close the reader to indicate as much.
 */
public void readData(SqlDataReader reader)
{
    size = 0;
    do
    {
        if (reader.GetInt32(0) == pid)
        {
            size++;
            dates.Add(reader.GetDateTime(1));
            quantities.Add(reader.GetInt32(2));
        }
        else
        {
            return;
        }
    }
    while (reader.Read());
```

```
        // reader ran out. close it.
        reader.Close();
    }

    /*
     * This method is called to compute the production schedule
     * for the item. It does no I/O, but puts in motion the
     * dynamic programming algorithm which produces the schedule.
     */
    public void ComputeSchedule()
    {
        int[] days = ComputeProductionDays();
        schedule = new int[size];
        for (int i = 0; i < size; i++)
        {
            schedule[days[i]] += (Int32)(quantities[i]);
        }
    }

    /*
     * Pipe the schedule to the user, computing it if need be.
     */
    public void OutputSchedule(SqlPipe pipe)
    {
        // Ensure that the schedule has been computed.
        if (schedule == null)
        {
            ComputeSchedule();
        }

        // Make a record in which to store the data.
        SqlDataRecord record = new SqlDataRecord(metadata);
        record.SetSqlChars(0, name);

        // Start up the output pipe.
        pipe.SendResultsStart(record);
        for (int i = 0; i < size; i++)
        {
            // Pipe each day of production. Omit zero production
            // days.
            if (schedule[i] != 0)
            {
                record.SetDateTime(1, (DateTime)(dates[i]));
                record.SetInt32(2, schedule[i]);
                pipe.SendResultsRow(record);
            }
        }
        pipe.SendResultsEnd();
    }

    /*
     * Compute the table and then walk it to find the best
     * days to produce the item.
     */
    private int[] ComputeProductionDays()
```

```
{
    // We fill this in. It says when each day's quota is
    // actually produced.
    int[] productionDays = new int[size];

    // First, compute the table.
    int[][] table = ComputeTable();

    // Then walk the table, creating a second table which encodes
    // the best production days.
    int[] optimal = new int[size + 1];
    int[] optimalLoc = new int[size];
    optimal[size] = 0;
    for (int i = size - 1; i >= 0; i--)
    {
        int min = table[i][i] + optimal[i + 1];
        int minloc = i;
        for (int j = i + 1; j < size; j++)
        {
            int temp = table[i][j] + optimal[j + 1];
            if (temp < min)
            {
                min = temp;
                minloc = j;
            }
        }
        optimal[i] = min;
        optimalLoc[i] = minloc;
    }

    // Finally, decode the optimal values into production days.
    int pday = 0;
    int until = optimalLoc[0] + 1;
    for (int i = 0; i < size; i++)
    {
        if (until == i)
        {
            pday = i;
            until = optimalLoc[i] + 1;
        }
        productionDays[i] = pday;
    }

    // We now have a list of days which we will produce the good.
    return productionDays;
}

/*
 * The main part of the dynamic programming solution. Each entry
 * table[i,j] stores the cost of producing enough of the good on
 * day i to meet needs through day j. This table is only half-
 * filled when complete.
 */
private int[][] ComputeTable()
{
    int[][] table = new int[size][];
```

```
            for (int i = 0; i < size; i++) table[i] = new int[size];
            for (int i = 0; i < size; i++)
            {

                // If we produce the good on the same day we ship it we
                // incur a startup cost.
                table[i][i] = startCost;

                // For other days, we have the cost for the previous
                // cell plus the cost of storing the good for this long.
                for (int j = i + 1; j < size; j++)
                {
                    table[i][j] = table[i][j - 1] +
                        (((int)quantities[j]) * holdCost *
                        diff((DateTime)(dates[i]), (DateTime)(dates[j])));
                }
            }
            return table;
        }

        /*
         * A utility to compute the difference between two days.
         */
        private int diff(DateTime start, DateTime end)
        {
            TimeSpan diff = end.Subtract(start);
            return diff.Days;
        }

    }
};
```