رایان

شماره دوم

دی ماه ۱۳۹۵

مجله الکترونیکی کامپیوتر و فناوری اطلاعات گروه آموزشی کامپیوتر فنی و حرفه ای و کاردانش استان اردبیل

Microsoft®

SQL Server®

رایان

مجله الکترونیکی کامپیوتر و فناوری اطلاعات گروه آموزشی کامپیوتر

فنی و حرفه ای و کاردانش استان اردبیل

هنرآموزان استان می توانید مطالب و مقالات خود را برای ما ارسال کنید تا پس از بررسی در نشریه با نام خودتان چاپ شود.

وبلاگ گروه آموزشی کامپیوتر فنی و حرفه ای استان اردبیل:

www.ardcomputer.blog.ir

کانال تلگرامی گروه آموزشی کامپیوتر فنی و حرفه ای استان اردبیل:

@ardcomputer

آنچه در این شماره خواهید خواند:

بهینه سازی پرس و جوها در SQL Server

یک برنامه ۱۲ مرحله‌ای

تنظیم پرس و جو، یک ابزار قدرتمند برای مدیران پایگاه اطلاعاتی و توسعه دهندگان در جهت توسعه عملکرد SQL Server است. برخلاف مقیاس‌های مربوط به عملکرد سرور در سطح- سیستم (حافظه، پردازنده‌ها، و غیره)، تنظیم پرس و جو بیشتر بر کاهش مقدار I/O منطقی در یک پرس و جو تاکید دارد، چرا که هر چه مقدار ورودی‌ها و خروجی‌ها کمتر باشد، پرس و جو سریعتر خواهد بود. در حقیقت، برخی مشکلات مربوط به عملکرد، می‌توانند از طریق تنظیم پرس و جو حل شوند. تمرکز روی منابع سیستم می‌تواند منجر به سرمایه‌گذاری‌های پرهزینه و غیر ضروری سخت افزاری گردد که در نهایت هم پرس و جو را سریعتر نمی‌کند.

هنوز هم اکثر مدیران پایگاه اطلاعاتی با تنظیم پرس و جو مشکل دارند. چگونه به پرس و جو دسترسی دارید؟ چگونه می‌توانید نواقص موجود در نحوه نوشتن یک پرس وجو را کشف کنید؟ چگونه می‌توانید فرصت‌های پنهان برای ترقی و پیشرفت را آشکار نمایید؟ چگونه مطمئن می‌شوید که با انجام اصلاحات و تغییرات خاص می‌توانید در واقع سرعت پرس و جو را بهبود ببخشید؟ آنچه موجب می شود که تنظیم پرس و جو بیشتر شبیه یک هنر باشد تا علم، این است که پاسخ درست یا غلطی وجود ندارد، غیر از این که چه چیزی بیشترین تناسب را با موقعیت فعلی دارد.

این مقاله با اراده یک پروسه ۱۲ مرحله ای پرده از راز و رمز تنظیم پرس و جو برمی‌دارد و متخصصین پایگاه اطلاعاتی در هر سطحی می‌توانند به طور سیستماتیک به آن دسترسی داشته باشند و عملکرد پرس و جو را تنظیم کنند، از تکنیک های ابتدایی گرفته تا تنظیم پیشرفته پرس و جو همچون شاخص‌گذاری. وقتی این پروسه را از ابتدا تا انتها به کار بگیرید، می‌توانید عملکرد پرس و جو را به روشی که قابل اندازه‌گیری باشد، بهبود ببخشید و متوجه خواهید شد که پرس و جو را تا حد امکان بهینه سازی کرده‌اید. همیشه از اصول اولیه آنالیز پرس و جو آغاز کنید وقتی در مورد اصلاح یک پرس و جوی کُند سوال می‌شود، مدیران با تجربه پایگاه اطلاعاتی مستقیماً به سراغ آزمایش طرح‌های اجرایی می‌روند و سپس با اجرای پرس و جو شگفت زده می شوند که برگردان داده‌ها چقدر طول می‌کشد! در آن لحظه، گاهی اوقات این فهم وجود دارد که جدول واقعاً بزرگ است. به همین دلیل است که همیشه توصیه می‌شود با اصول اولیه آغاز کنید، یعنی از ابتدا بفهمید که با چه چیزی سر و کار دارید.

۱-جدول‌ها و rowcount های خود را بشناسید ابتدا، مطمئن شوید که در واقع روی یک جدول کار می‌کنید، نه یک ویو یا تابع table-valued. اگر ویو باشد، به تعریف ویو نیاز دارید. توابع table-valued مفاهیم اجرایی خودشان را دارند.

نکته: برای آزمایش این جزئیات می‌توانید از SSMS استفاده کنید.

با پرس و جو کردن DMV ها (مثال زیر)، recount را بررسی کنید. اگر، برای مثال، پرس و جو در محیط تو سعه ساخته و آزمایش شـده اسـت، اما برای اولین بار در محیط تولید اجرا می‌شود، rowcount واقعی احتمالاً بالاتر است.

۲- فیلترهـای پرس و جو را آزمـایش کنیـد. عبـارت‌هـای WHERE و JOIN را آزمایش کنید و rowcount فیلتر شـده را یادداشت نمایید.

نکته: اگر فیلتری وجود ندارد، و بیشتر جدول بازگردانده شده، ببینید آیا تمام آن داده‌ها مورد نیاز هسـتند. اگر به طور کل هیچ فیلتری وجود ندارد، این می‌تواند یک زنگ خطر باشـد و دلیل قانع کننده‌ای برای بررسی بیشتر. در واقع همین مسئله می‌تواند موجب کند شدن پرس و جو گردد.

۳- گـزینـش پـذیـری جـدول هـای خـود را بـدانـیـد بر اسـاس جدول ها و فیلترهای موجود در دو مرحله قبلی، بدانید که با چه تعداد سطر کار خواهید کرد و اندازه مجموعه واقعی، منطقی چقدر اسـت. در مورد بحث گزینشـی‌پذیری، SQL Tuning دن تو (Dan Tow) را پیشـنهاد می‌کنیم و استفاده از رسم نمودار به عنوان یک ابزار قدرتمند در ارزیابی پـرس و جـوهـا و گـزیـنـش پـرس و جـو. این موضـوع بخصـوص برای joinهای Left، Right و Outer مهم اسـت. باید خوب بفهمید که چه زمانی از گزاره اسـتفاده کنید به این ترتیب می‌توانید مطمئن شـوید که با کوچکترین مجموعه ممکن آغاز می‌کنید و فیلترها به موقع به کار گرفته می‌شوند.

۴- سـتون‌های پرس و جوی اضـافی را آنالیز کنید- چیزهای اضـافـی خـارج از فـیـلـتـرهـا و Joinهـا توابع اسـکالر یا * SELECT را به دقت بررسـی کنید تا مشـخص شـود که آیا سـتون‌های اضـافی درگیر هسـتند. آیا CASE، CAST و CONVERT در عبـارت WHERE رخ می‌دهند؟ آیا SARGable (شـاخص قابل جسـتجو) اسـت؟ هرچه سـتون های بیشـتری را برگردانید، دیگر برای طرح اجرایی مطلوب نیست که از عملیات شاخص خاص استفاده کند، و این می‌تواند باعث تنزل عملکرد شود.

ادامه دهید تا به آنالیز پیشـرفته‌تر پرس و جو برسـید ۵- کلید های موجود، قید ها (Constrain)، شـاخص ها را بازبینی کنید تا مطمئن شوید که تلاش مضاعف نمی‌کنید یا همپوشـانی شـاخص‌هایی که از قبل وجود داشـتند اتفاق نمی‌افتد.

این قیدها را بشناسید و استفاده کنید چراکه می‌توانند برای شروع تنظیم مفید باشند.

تعریف کلید اصلی (Primary Key) چیست؟ آیا آن کلید نیز Clustered است؟ اگر یک کلید Clustered وسیع دارید، باید از این کلید در شـاخص های non-Clustered کپی بگیرید، یعنی برای حل مشگل پرس و جو باید صفحات بیشتری را در مخزن بافر بخوانید.

اگر از قید های کلید خارجی (Foreign Key) استفاده می‌کنید، بررسـی کنید که آیا آن‌ها در مدل داده‌ای شـما کار می‌کنند. بهینه ساز می‌تواند از قیدهای کلید خارجی استفاده کند تا طرح‌های اجرایی را بهتر نماید، این به اجرای سـریعتر پرس و جو کمک می‌کند. برای بدست آوردن اطلاعات مربوط به شاخص‌ها، رویه ذخیره شده sp_helpindex را اجرا نمایید:

توجه داشته باشید که ستون‌های گنجانده شده به حساب آورده نشـوند. اگر به این اطلاعات نیاز دارید؛ باید از یک پرس و جـوی مـتـفـاوت اسـتـفـاده نـمـایـیـد.
۶- طرح اجرایی واقعی (نه طرح تخمین زده شده) را آزمایش کنید. طرح های تخیمنی از آمار های تخمینی برای تعیین سـطرهای تخمینی اسـتفاده می‌کنند؛ طرح های واقعی از آمارهای واقعی در زمان‌اجرا اسـتفاده می‌کنند. اگر طرح‌های واقعی و تخمینی متفاوت هستند، ممکن است به جستجوی بیشتری نیاز داشته باشید.

توجه کنید که در این مرحله، ممکن است بخواهید آمارها را در (SET STATISTIC IO On و SET STATISTICSTIME On) تنظیم کنید.

۷- نتایج خود را ثبت کنید، روی تعداد I/O های منطقی تمرکز کنید. این مرحله بسیار مهم است و ممکن است از سوی مدیران پایگاه اطلاعاتی نادیده گرفته شود، اگر نتایج را ثبت نکنید، قادر نخواهید بود اثر واقعی تغییراتی که اعمال می‌کنید را مشخص نمایید.

۸- پرس و جو را براساس آنچه یافته‌اید، تنظیم کنید؛ تغییرات کوچک و فردی را در لحظه اعمال نمایید اگر تغییرات زیادی را در لحظه اعمال کنید، ممکن است ببینید که تغییرات همدیگر را لغو می‌کنند. کار را با جستجوی پرهزینه‌ترین عمل‌ها آغاز کنید. هیچ پاسخ درست و غلطی وجود ندارد، فقط مناسب موقعیت فعلی است (توجه کنید آمارهای منقضی شده می‌تواند همگی آنها را تحت تاثیر قرار دهد).

برخی از عملیات پرهزینه‌ای که ممکن است با آن‌ها مواجه شوید:

– انتقال داده ها از یک عمل به عمل دیگر. آیا تعداد واقعی سطرها بسیار بیشتر از تعداد تخمین زده شده است؟ اگر تخمین‌ها از واقعیت‌ها متفاوتند، ممکن است نیاز به بررسی بیشتر باشد.

– آیا جستجوها یا اسکن‌ها در این سناریوی خاص بسیار گرانتر هستند؟ برخلاف باور عموم، در برخی نمونه‌ها اسکن یک جدول ممکن است کم هزینه‌تر از یک جستجو باشد. برای مثال، اگر جدول بسیار کوچک است، SQL Server کل جدول را در حافظه می‌خواند و بنابراین جستجو نیاز نیست.

– آیا اسنیفینگ پارامتر یک مسئله است (اسنیفینگ پارامتر نتیجه استفاده مجدد از طرح از پیش کش‌شده است که برای مقادیر پارامتر، از اجرای اصلی بهینه سازی می‌شود، و آن پارامترها ممکن است بسیار متفاوت باشند؟) آیا از مقایر محلی استفاده می‌کند؟

– آیا spool operations وجود دارند (مجموعه نتایج ذخیره شده در tempdb برای استفاده بعد) و اگر وجود دارد، آیا ضروری هستند؟

– در این موقعیت چه چیزی بهتر است:join های LOOP ، MERGE یا Hash ؟ بستگی به شرایط خاص و آماری که بهینه ساز ا ستفاده می‌کند، دارد. آیا ج ستجوها وجود دارند، اگر اینطور است آیا ضروری هستند؟

۹- اجرای مجدد پرس و جو و ضبط نتایج حاصل از تغییری که انجام داده‌اید. اگر شاهد بهبودی در I/O های منطقی هستید، ولی این بهبودی به قدر کافی نیست، به مرحله ۸ برگردید تا فاکتورهای دیگری که ممکن است به تنظیم نیاز داشته باشند را بررسی نمایید. در هر لحظه یک تغییر را انجام دهید، پرس و جو را مجدد اجرا نمایید و نتایج را مقایسه کنید تا زمانی که این رضایتمندی در شما ایجاد شود که تمام عملیات پرهزینه را مخاطب قرار داده‌اید.

۱۰- در این لحظه اگر باور دارید که پرس و جو به خوبی آنچه که می‌توانست باشد، نوشته شده و شما هنوز به بهبود بیشتری نیاز دارید، تنظیم شاخص را برای کاهش I/O منطقی در نظر بگیرید. اضافه کردن و تنظیم کردن شاخص‌ها همیشه بهترین کار نیست، اما اگر نمی‌توانید کد را تغییر دهید، این تنها کاریست که می‌توانید انجام دهید.
– شاخص‌های موجود را در نظر بگیرید. آیا به شکل کارآمدی مورد استفاده قرار می‌گیرند؟ با حداقل گزینش‌پذیری روی آن جدول‌ها تمرکز کنید.

– یک Covering index را در نظر بگیرید- شاخصی که شامل هر ستون متناسب با پرس و جو است. مطمئن شوید ابتدا دستورهای Delete/Update/Insert را آزمایش می‌کنید: حجم آن تغییرات چقدر است؟
– یک شاخص فیلترشده را در نظر بگیرید (SQL Server

2008 به بعد) – یک شاخص non-clustered که دارای گزاره یا عبارت WHERE ا ست. اما توجه دا شته با شید که اگر یک دستور پارامتربندی شده یا متغییرهای محلی دارید، بهینه ساز نمی‌تواند از شاخص فیلتر شده استفاده کند.

۱۱- اگر تنظیمات را در مرحله ۱۰ انجام دادید، پرس و جو را دوباره اجرا کنید و نتایج را ثبت کنید

۱۲- در نهایت، هر وقت که نیاز شد، این بازدارنده‌های اجرایی را که غالباً با آن ها مواجه می‌شوید را حذف نمایید

– توجه داشته باشید که code-first generator (برای مثال، nHibernate، LNQ،EMF) می‌توانند باعث پر شدن Plan Cache شود.

نکته: اگر از code-first generatorها استفاده می‌کنید، در نظر داشته باشید که OPTIMIZE FOR AD HOC WORKLOADS را فعال کنید.

– به دنبال سوءاستفاده از wildcardهای (*) باشید که می‌تواند ستون‌های بیشتری را بازگرداند

– توابع اسکالر و توابع چند دستوری برای هر سطری که بازگردانده می شود، فراخوانده می شوند و می‌توانند مورد سوء استفاده قرار بگیرند

– ویوهای تو در تو که در تمام سرورهای لینک شده مورد استفاده قرار می‌گیرند، می‌توانند زمان پردازش را اضافه کنند

– مکان‌نماها و پردازش سطر به سطر می‌توانند پردازش را کند کنند.

– hint های Join/query/index/table می‌توانند به شکل چشمگیری نحوه کار کردن یک پرس و جو را تغییر دهند. از آنها تنها زمانی استفاده کنید که از تمام امکانات دیگر خسته شده‌اید.

تفاوت‌های STORED PROCEDURE و FUNCTION ها در SQL SERVER

۱. Functionها حتما باید مقدار بازگشتی داشته باشند ولی مقدار بازگشتی در *SPها به صورت اختیاری است. یک SP می‌تواند از صفر تا n خروجی داشته باشد.
۲. Functionها تنها می‌توانند پارامترهای ورودی داشته باشند ولی SPها علاوه بر پارامترهای ورودی، می‌توانند پارامتر خروجی نیز داشته باشند.
۳. Functionها می‌توانند از داخل SPها صدا زده شوند ولی SPها قابل فراخوانی از داخل Functionها نیستند.
۴. Functionها تنها می‌توانند دستورات Select را اجرا کنند ولی SPها می‌توانند تمامی دستورات را از قبیل Select, Delete, Insert و... اجرا نمایند.
۵. Functionها قابل فراخوانی از طریق دستورات Select هستند ولی SPها این قابلیت را ندارند.
۶. Functionها قابلیت استفاده به صورت مستقیم را در دستوراتی مانند Having، دارند ولی SPها این گونه نیستند.
۷. Functionها در هربار استفاده کامپایل می‌شوند لیکن SPها می‌توانند یکبار کامپایل شده، execution plan آنها برای استفاده آتی نگهداری شوند. این مهمترین تفاوت این دو می‌باشد.
۸. استثناء و خطاها در SPها قابل دریافت توسط Try...Catch می‌باشند ولی در Functionها این قابلیت وجود ندارد.

مَستِر کردن فیلتر های CSS

اســـتفاده از فیلتر های CSS برای افزودن به اطلاعات و داده ها، واکنش پذیری سایت شما را کاهش می دهد. آن چیزی که کمتر شناخته شده است، این است که شما می توانید از ابتدا با ترکیب قسمت های مختلف فیلتر SVG در یک فیلتر CSS، در زمان بالا رفتن ترافیک مشـاهده یا شـرایط پیچیده ی دیگر، عملکرد قابل قبولی از سایت خود را مشاهده کنید. در این مقاله، روش ساختن یک فیلتر چند مرحله ای (با ا ستفاده از نور، صدا و تیرگی) برای ایجاد جلوه های ویژه ی سه بعدی جذاب بر روی متن ها، توضیح داده شـــده اســـت. فیلتر های CSS، ابزار بی نظیری برای تغییر شـرایط نمایش متن های قرار داده شـده بر روی سـایت شـما ه ستند. هر کدام از این بلوک های ساخته شده، مانند یک آجر بوده و هنگامی که در کنار یکدیگر قرار می گیرند، یک ساختمان زیبا را می سازند. بهتر از همه این ا ست که شما می توانید این آجر ها را به سلیقه ی خودتان در کنار یکدیگر قرار داده تا فیلتر CSS مورد نظرتان را بسـازید. اما چگونه؟ در این مقاله، سـاختن چنـیـن فـیـلـتـر هـایـی را تـوضـــیـح خـواهـیـم داد. در ابتدا نگاهی به SVG خواهیم داشـــت. در حقیقت SVG یک زبان گرافیکی برداری اســـت که در تمام مرورگر های جدید HTML5 قرار داده شده و یکی از ویژگی های بی نظیر آن، فیلتر ها می باشـــند. در حقیقت SVG شـــامل یک <filter> و تعداد زیادی فیلتر ابتدایی (مقدماتی) اســـت. می توان گفت فیلتر های ابتدایی SVG همان آجر هایی هستند که ساختمان مورد نظر ما را می سـازند. هدف ما این اسـت که تعدادی از آن ها را در کنار هم قرار داده و از آن ها در وب ســـایت خود اســـتفاده کنیم. هنگامی که آجر های خود را در فیلتر مورد نظرمان قرار بدهیم، می توانیم با اســـتفاده از انتخاب کننده های CSS، به آن ها مراجعه کرده تا در وب سایت ما به کار گرفته شوند. در ابتدا یک نمونه ی بسیار ساده را بیان کرده و در ادامه به نمونه ی پیچیده تری اشاره خواهیم داشت.

ســـاخـــتـــن یـــک فـیـلـتـر تـابـش (Glow Filter) در نخستین نمونه، ساختن فیلتری را شرح می دهیم که با استفاده از آن می توان یک تابش یا برافروختگی ایجاد کرد. برای ســـاختن این فیلتر، از چهار فیلتر مقدماتی feGaussianBlur (ایجاد کننده ی اثر تیرگی)، feFlood (پُر کننده ی یک محدوده با یک رنگ م شخص)، feComposite (ترکیب کننده ی پیکسل ها) و feMerge (ترکیب کننده ی محتویات مورد نظر با فیلتر به منظور به دســـت آوردن نتیجه ی مورد نظر)، اســـتفاده خواهیم کرد.

نخســـتین چیزی که به آن نیاز داریم این اســـت که یک تصـویر (snapshot) از متن مورد نظر تهیه کرده و آن را به رنگ ســـفید تبدیل کنیم (زیرا در ادامه از متن سفید شده برای ایجاد تابش یا برافروختگی اســـتفاده خواهیم کرد). از کد فیلتر SVG زیر برای ایجاد متن سفید رنگ استفاده می کنیم:

```
filter id=”white”> <feFlood flood-color=”white” /> >
<feComposite in2=”SourceAlpha” operator=”in”
<result=”white”/></filter
```

این فیلتر شامل دو بلوک ا ست. نخ ستین بلوک، فیلتر feFlood است که محدوده ی مورد نظر را با رنگ سفید پُر می کند (فیلتر ها از یک ت صویر خارج از صفحه ی م ستطیلی برای ایجاد هر اثر در زنجیره ی اثر های مورد نظر ا ستفاده می کنند). دومین فیلتر مورد اســـتفاده، اثر فیلتر feComposite اســـت که برای ترکیب پیکســـل های متن اصـــلی و پیکســـل های خروجی از یک فیلتر دیگر (با ا ستفاده از برخی قوانین ترکیب) به کار گرفته می شود. در این جا، می بایســـت بیان کرد که فیلتر های SVG به یک ترتیب مشخص شده کار کرده و به همین دلیل مانند حلقه های زنجیر، به یکدیگر متصـــل می شـــوند. وقتی ما از اثر این فیلتر بر روی متن خود اســـتفاده می کنیم، ابتدا اثر فیلتر feFlood اجرا شـــده و ســـپس اثر feComposite، خروجی به دســـت آمده از

feFlood را به عنوان یک ورودی، مورد اســـتفاده قرار می دهد. فیلتر FeComposite از عملگر in اســـتفاده می کند که همان قانون ترکیب «ایجاد تصـــویر از ورودی مورد نظر با اســـتفاده از پیکسل های داخلی متن اصلی» می باشد. در حقیقت باید توجه داشت که این قوانین، از اصول ریاضی پیروی می کنند. نتیجه ی استفاده از این فیلتر این است که تصویر مستطیل شکل سفید رنگ، اثر همان چیزی که مورد نظر ما است را به خود می گیرد. در چنین شرایطی، متن سفید رنگ ما، زیر پوشش اثر مورد نظر قرار خواهد گرفت. روش انتخاب متن، اســـتفاده از یک انتخاب کننده ی CSS برای انتخاب متن (محتویات) مورد نظر اســـت. برای آشـــنایی بیشـــتر می توانیم به نمونه ی زیر اشـــاره داشـــته باشیم:

{;(white { -webkit-filter: url(#white); filter: url(#white.

در چنین حالتی، اثر فیلتر مورد نظر بر روی تمامی محتویاتی که با یک CSS ســـفید رنگ («white») مشـــخص شـــده اند، قابل مشاهده خواهد بود. البته استفاده از چنین فیلتری بر روی صفحه ای که رنگ آن سفید است، چندان منا سب نبوده و بهتر است یـــک زمـــینـــه ی رنـــگـــی را بـــه آن اضـــافـــه کـــنـــیـــم:

bluebg { background-color: lightblue; } <div . class="bluebg"> <h1 class="white">WHITE GLOW <<FILTER</h1> </div

اکنون که مرحله های ابتدایی و اســـاســـی ایجاد یک فیلتر دو مرحله ای را مشاهده کرده ایم، زمان آن رسیده است که چگونگی استفاده از فیلتر های بیشتر را بررسی کنیم.

ســـاخـــتـــن یـــک فـــیـــلـــتـــر تـــیـــرگـــی (Blurred Filter) برای ایجاد تابش یا برافروختگی بر روی متن مورد نظرمان، می بایست متن سفید رنگ به دست آمده از زنجیره ی فیلتر بالا را گرفته و فیلتر تیرگی را بر روی آن قرار دهیم. در حقیقت این فیلتر، اساس ایجاد برافروختگی مورد نظرمان را تشکیل می دهد. بـــرای بـــه کـــار بـــردن فـــیـــلـــتـــر تـــیـــرگـــی از

filter id="whiteblur"> <feFlood flood-color="white" > /> <feComposite in2="SourceAlpha" operator="in" /> <filter/> </ "۴"=<feGaussianBlur stdDeviation

اســـتفاده می کنیم. همان طور که مشـــاهده می کنیم، از یک اثر ا ضافی feGaussianblur ا ستفاده کرده ایم. این اثر، متن سفید رنگ ایجاد شده از مراحل قبل را گرفته و آن را تیره می کند:

whiteblur { -webkit-filter: url(#whiteblur); filter: . url(#whiteblur); } <h1 class="whiteblur">WHITE GLOW <FILTER</h1></div

برای برجســـته تر شـــدن تیرگی، لازم اســـت اثر تیرگی را بیشـــتر کنیم. برای انجام این کار از feComponentTransfer اســـتفاده می کنیم. در حقیقت این همان فیلتری اســـت که امکان بازی کردن با رنگ ها و کانال های آلفا را در تصـــویر خارج از صـــفحه (offscreen)، به تمامی روش های ممکن، فراهم می کند.

اکنون می خواهیم یک تصـــویر تیره شـــده را با اســـتفاده از کانال آلفا، درخشان تر کنیم. در چنین شرایطی، زنجیره ی فیلتر ما به این شکل خواهد بود:

filter id="whitetransfer"> <feFlood flood-> color="white" /> <feComposite in2="SourceAlpha" </ "۴"=operator="in" /> <feGaussianBlur stdDeviation feComponentTransfer><feFuncA type="linear" > feComponentTransfer> /> "۰"=intercept "۳"=slope <</filter

توجه می کنید که از اثر feFuncA ا ستفاده کرده ایم. با ا ستفاده از این اثر، کانال آلفا به وســـیله ی افزایش ســـه برابری مقادیر، بهبود داده می شود. چگونگی استفاده از این فیلتر، به این ترتیب است:

whitetransfer { -webkit-filter: url(#whitetransfer); filter: url(#whitetransfer); } <h1 class="whitetransfer">WHITE GLOW FILTER</ h1> </div<<

تنها می بایست با ا ستفاده از اثر فیلتر feMerge، نتایج به د ست آمده از فیلتر ها را با متن اصـــلی مورد نظرمان ترکیب کنیم. نسخه ی نهایی زنجیره ی فیلتر ما، به این شکل تمام می شود:

filter id=”whiteglow”> <feFlood flood-> color=”white” /> <feComposite in2=”SourceAlpha” </ ”۴”=operator=”in” /> <feGaussianBlur stdDeviation feComponentTransfer> <feFuncA type=”linear” > feComponentTransfer> /> </ ”۰”=intercept ”۳”=slope <feMerge> <feMergeNode /> <feMergeNode <in=”SourceGraphic” /> </feMerge> </filter

در انتها می توانیم برای مشخص کردن اثر مورد نظر، از کُد زیر استفاده کنیم:

whiteglow { -webkit-filter: url(#whiteglow); filter: . url(#whiteglow); } <h1 class=”whiteglow”>WHITE GLOW FILTER</h1> <</div

ساختن یک فیلتر برجستگی (Chiselled Filter)

در این قسمت از مقاله، چگونگی استفاده از اثر فیلتر ها برای ایجاد یک نمایش سه بعدی برجسته را شرح خواهیم داد. آن دسته از فیلتر های SVG که از آن ها استفاده می کنیم، شامل نور پردازی ها، شکل ها و صدا های مختلف می باشند و اگر چه ممکن است پیچیده به نظر برسند اما استفاده از آن ها بسیار آسان است. برای استفاده از این فیلتر باید سه مرحله ی مختلف را با یکدیگر ترکیب کرد که در هر کدام از آن ها از چند فیلتر مختلف استفاده شده است. برای استفاده از این فیلتر، مراحل زیر را انجام خواهیم داد:

۱, یک تصویر را به bump map (نقشه ی برجسته) تبدیل می کنیم (برای توضیحات بیشتر به قسمت «نگاه دقیق تر» همین مقاله مراجعه کنید).

۲, یک ظاهر برجسته از متن مورد نظرمان ایجاد می کنیم.

۳, با ایجاد یک سایه در پشت متن مورد نظر، یک ظاهر سه بعدی از آن ایجاد می کنیم.

فیلتر برجستگی: نقشه های برجسته و روشن شدن، برای ایجاد یک ظاهر سه بعدی با یکدیگر ترکیب می شوند نکته ی شگفت انگیز در مورد اثر فیلتر ها این است که هیچ محدودیتی برای استفاده از آن ها وجود ندارد!

ساختن یک نقشه ی برجسته (Bump Map)

نقشه های برجسته (Bump Map ها) که در بازی های سه بعدی مورد استفاده قرار می گیرند را می توانید مانند سطح نا هموار پارچه یا تخته سنگ تصور کنید که هر پیکسل آن یک ارتفاع مشخص را نشان می دهد.

برای ایجاد bump map، چند فیلتر مختلف را در کنار هم قرار می دهیم. برای این که با کُد مورد استفاده برای ایجاد bump map ها آشنا شوید، به قسمت «نگاه دقیق تر» مراجعه کنید. بعد از ایجاد bump map، از این فیلتر استفاده می کنیم:

bumps { -webkit-filter: url(#bumps); filter: . url(#bumps); } <<h1>CHISELLED EFFECT</h1> </div

ساختن یک نمایش برجسته ی طلایی (Gold Embossed)

برای اینکه متن مورد نظر ما یک ظاهر سه بعدی طلایی برجسته ی خوش رنگ داشته باشد، باید از فیلتر feComponentTransfer در کنار چند فیلتر دیگر مانند feComposite و feBlend استفاده کنیم.

در فیلتر feBlend، حالت های ترکیب رنگی که در بیشتر برنامه های شناخته شده ی دسکتاپ مانند Photoshop و Inkscape (inkscape.org)) به کار رفته است مورد استفاده قرار می گیرد. این ویژگی، امکان استفاده از ترکیبات رنگی زیبا و گوناگون را برای کاربران فراهم کرده است. برای ایجاد رنگ طلایی، ابتدا باید رنگ متن مورد نظرمان را با استفاده از ویژگی های CSS، به رنگ طلایی تبدیل کنیم. سپس به وسیله ی screen و multiply از فیلتر feBlend استفاده می کنیم. در چنین شرایطی امکان ترکیب رنگ های لایه های مختلف و پیکسل های تصویر های ورودی گوناگون نیز وجود خواهد داشت. می توانید کُد مربوط به ایجاد نمایش برجسته ی طلایی را از netm.ag/gold-250 دانلود کنید.

ساختن یک سایه ی برجسته (Drop Shadow) در انتها، چگونگی ایجاد یک سایه ی برجسته را با استفاده از اثر فیلتر شرح می دهیم. برای ایجاد سایه از feMorphology (که کم رنگ کننده ی تصویر است)، feGaussianBlur (که برای ایجاد فیلتر تابشی یا برافروختگی مورد استفاده قرار می گیرد)، feOffset (که جای تصویر را تغییر می دهد) و فیلتر پُر کاربرد feComposite در کنار یکدیگر استفاده می کنیم. می توانید کُد اثر فیلتر برای ایجاد سایه ی برجسته را از netm.ag/drop-250 دانلود کرده و نتیجه را به این شکل به کار ببرید:

```
shadowbumps { -webkit-filter: url(#shadowbumps);
filter: url(#shadowbumps); } <div>
class=”shadowbumps”> <h1>CHISELLED
<EFFECT</h1> </div
```

نکته ی شگفت انگیز در مورد اثر فیلتر ها این است که هیچ محدودیتی برای استفاده از آن ها وجود ندارد. شما می توانید تمام بلوک های اثر فیلتر های SVG را به هر شکلی که می خواهید با یکدیگر ترکیب کرده تا انواع مختلف زنجیره های فیلتر را ایجاد کنید.

بازگردانی پایگاه داده بدون فایل لاگ

در بعضی مواقع ممکن است که در حین کار و با تراکنش‌های باز، دیتابیس SQL Server دچار م شکل شود و از د سترس خارج شود و این فرض را هم در نظر بگیرید که بک آپ دیتابیس مربوط به زمانی بیش از حد انتظار است. آیا ممکن است که دیتابیس را با وجود از دست دادن فایل لاگ آن بازگردانی کرد؟ جواب: بله ولی بدون عواقب نیست .

بطور معمول در زمانی که تراکنش‌های باز بر روی سرور دیتابیس وجود دارد و بانک کرش میکند، کرش ریکاوری، تراکنش‌های باز را رول بک میکند. این امر مانع از اثرات پراکنده از تراکنش‌های فعلی در پایگاه داده می‌باشد .

اگر لاگ در د سترس نبا شد هنگامی که کار سرور SQL شروع می شود، پایگاه داده در حالت SUSPECT قرار داده می شود. در این مواقع تنها راه آن لاین نمودن بانک (البته منظور آماده بکار نمودن بانک نیست) استفاده از حالت قابلیت تعمیر اورژانسی است که از نسخه ۲۰۰۵ افزده شده است که با ساخت یک فایل لاگ جدید و سپس اجرا نمودن DBCC CHECKDB با استفاده از REPAIR_ALLOW_DATA_LOSS انجام می‌شود .

مشکل اینجاست که اگر شما به همین طریق بخواهید ادامه بدهید و از این قابلیت اورژانسی استفاده کنید، مسلما امکان وجود دسته‌ای از تغییرات در بانک که ممکن است در میانه اعمال بروز رسانی چند رکورد فروش در یک جدول نیمی از تراکنش های آن ها روی بانک اعمال شده و بعد از راه اندازی دوباره با لاگ جدید، امکان برگرداندن آن‌ها وجود داشته باشد، مواجه شوید. به این معنی که در بهترین حالت، پایگاه داده بدست آمده با برنامه هماهنگ نیست و یا اینکه به بک آپی که وجود دارد رضایت بدهید .

حالت تعمیر اورژانسی، زمانیکه همه راه‌حل‌های بازگردانی بانک مغلوب شوند آخرین روش مانده ا ست. این حالت، حالتی بین دو شر، ریکاور کردن بانک به حالتی نا هماهنگ با نرم افزار و یا برگرداندن به زمانی خیلی عقب‌تر است که در نهایت اقدام به انجام هر دو و هماهنگ‌تر کردن بانک برای کار با برنامه میشوید که بسیار وقت گیر و مشکل ساز است .

ولی باز ممکن ا ست این و ضعیت پیش بیاد چون شما از اتفاقی که در زمان کرش در بانک افتاده اطلاعی ندارید. برای جلوگیری از این موارد در آینده سازوکار بک آپ گیری از بانک را باید تغییر بدهید و دفعات بک آپ گیری را افزایش بدهید و نیز از روش‌های جدید که قابلیت دسترسی بالا دارند استفاده کنید همانند mirroring و SQL Server 2012 Availability Groups است.

پشتیبانی از JSON در SQL SERVER 2016

مقدمه

زمانی که XML در عمل ا ستانداردی برای تبادلات داده‌ها شده بود، SQL Server 2005 پشتیبانی محلی از ذخیره سازی، مدیر یت و پردازش داده ها را معرفی کرد. این ویژگی هنوز در SQL Server وجود دارد چراکه XML امتیازات به ارث برده‌ای دارد و همچنان به صورت گ سترده مورد ا ستفاده قرار می‌گیرد، اما از آنجاییکه اکثر اپلیکیشن‌های جدید برای مکانیزم تبادل داده‌های سبک، استفاده از JSON را آغاز کرده‌اند؛ SQL Server 2016 پ شتیبانی توکار را برای ذخیره سازی، مدیریت و پردازش داده‌های JSON معرفی می کند. در این مقاله این ویژگی جدید را شرح می‌دهیم و ن شان می‌دهیم که چگونه می‌توانید از آن در اپلیکیشن‌های خود در سناریوهای مختلف استفاده نمایید.

پردازش داده های JSON در SQL Server 2016 SQL server پشتیبانی توکار برای ذخیره سازی، مدیریت و تجز یه داده های JSON را معرفی کرد. گر چه امکان ذخیره ، مدیریت و تجزیه داده‌های JSON حتی در ورژن‌های قبلی SQL Server وجود دارد اما اجرای آن مستلزم تلاش بیشتری در بخش تو سعه ا ست. با وجود پشتیبانی توکار می‌توانید این کار را خیلی سریع تر انجام دهید ضمن این که به جای نوشتن توابع برای تجز یه داده های JSON می‌توانید روی تجز یه داده های خود یا منطق اپلیکیشن خود تمرکز کنید. JSON مخفف JavaScript Object Notation می‌باشد و یک فرمت سبک برای تبادل داده‌های متنی خواندنی برای انسان می‌باشد که مبتنی بر زبان برنامه نویسی JavaScript است (گرچه این زبان برنامه نوی سی م ستقل از زبان می‌با شد و امروزه در اکثر زبان‌های برنامه نویسی مورد استفاده قرار می‌گیرد). با این ویژگی جدید می‌توانید از طریق داده‌های جدولی موجود در جدول‌های رابطه‌ای، فرمت خود را به JSON تبدیل نمایید و نیز داده‌های JSON را تجزیه کنید و آن‌ها را برای گزارش گیری، پیوند با جدول‌های دیگر یا انتقال به اپلیکیشن‌های دیگر که منتظر داده های جدولی هستند، به قالب فرمت جدولی در بیاورید.

برخلاف پشتیبانی بومی XML، جایی که نوع داده‌های XML را برای ذخیره کردن داده ها یا اسناد XML دارید، برای ذخیره کردن داده ها یا اسناد JSON در SQL Server 2016 ، از داده های NVARCHAR استفاده می‌کنید. این یعنی شما محدود نیستید و هرکجا که NVARCHAR پشتیبانی می شود، می‌توانید داده های JSON را تقریباً در هرجایی ذخیره و تجزیه نمایید. همچنین، نیاز به شاخص خاصی در داده های JSON ندارید و همچنان می‌توانید از شاخص‌های متداول که با آن‌ها آشنایی دارید، استفاده نمایید.

صادر کردن داده های جدولی به صورت داده های JSON SQL Server 2016 عبارت FOR JSON [AUTO | PATH] را معرفی می‌کند تا با پرس و جوی شما استفاده شود و آن را قبل از بازگرداند به سرویس گیرنده در قالب فرمت JSON فرمت نماید.اگر از قبل تجربه کار کردن با عبارت FOR XML را دارید، احتمالا خواهید فهمید که این عبارت جدید شبیه همین عبارت FOR XML است.

بگذار ید این را با مثال نشان دهیم. فرض کنید که پایگاه اطلاعاتی AdventureWorks2014 را دارید. می‌توانید این پرس و جو را اجرا کنید تا مح صولاتی را که در این دو سفارش خاص سفارش داده شده اند را اجرا نمایید. همانطور که در ت صویر زیر می‌بینید برای OrderID 43663 تنها یک محصول وجود دارد در حالیکه برای OrderID 43687 دو محصول.

```
SELECT H.SalesOrderID, H.Status,
H.PurchaseOrderNumber, H.ShipDate, P.ProductID

FROM [Sales].[SalesOrderHeader] H
INNER JOIN [Sales].SalesOrderDetail D ON
H.SalesOrderID = D.SalesOrderID
INNER JOIN [Production].[Product] P ON
D.ProductID = P.ProductID
WHERE H.SalesOrderID IN (43663, 43687)
GO
```

پرس و جوی سفارش محصولات

حال عبارت FOR JSON AUTO را در انتهای پرس و جوی بالا اضافه می‌کنیم و آن را اجرا می‌نماییم. همانطور که در تصویر زیر می‌بینید، این بار SQL Server نتایج این پرس و جو را می‌گیرد، آن را به صورت سند JSON فرمت می‌کند و در نهایت به سرویس گیرنده باز می‌گرداند.

```
SELECT H.SalesOrderID, H.Status,
H.PurchaseOrderNumber, H.ShipDate, P.ProductID
FROM [Sales].[SalesOrderHeader] H
INNER JOIN [Sales].SalesOrderDetail D ON
H.SalesOrderID = D.SalesOrderID
INNER JOIN [Production].[Product] P ON
D.ProductID = P.ProductID
WHERE H.SalesOrderID IN (43663, 43687)
FOR JSON AUTO
GO
```

پرس و جوی فرمت شده به صورت سند JSON

SQL Server بنا به دلایل مشخصی، خروجی JSON را با اضافه کردن فضاهای خالی مثل تب‌ها یا فاصله‌ها فرمت نمی‌کند، گرچه می‌توانید برای بهتر شدن قابلیت خواندن، این کار را به صورت دستی انجام دهید یا از یک ابزار آنلاین یا محلی برای فرمت کردن آن استفاده نمایید. برای یافتن یکی از این ابزارهای آنلاین جهت فرمت کردن خروجی JSON از این آدرس استفاده کنید:

https://jsonformatter.curiousconcept.com/

وقتی داده های JSON را از پرس وجوی بالا فرمت می‌کنید، متوجه خواهید شد که SQL Server با عبارت FOR JSON AUTO ، به طور خودکار خروجی JSON را بر اساس ساختار پرس وجو فرمت کرده است (ترتیب ستون ها در لیست SELECT و جدول منبع آن‌ها).

در این مورد، از آنجاییکه SalesOrderHeader اولین جدول است و SalesOrderDetail دومین جدول است، ستون‌ها از SalesOrderHeader به صورت خصوصیات آبجکت والد ایجاد می‌شوند در حالیکه ستون‌ها از SalesOrderDetail به عنوان خصوصیات آبجکت تو در تو:

SQL Server به طور خود کار خروجی JSON را فرمت کرده است. عبارت FOR JSON AUTO برای اکثر سناریوها مناسب است، اما این احتمال وجود دارد که در برخی سناریوهای خاص بخواهید روی نحوه ساخته شدن یا تودرتو شدن داده‌های JSON کنترل داشته باشید. عبارتFOR JSON PATH به شما کنترل تام می‌دهد تا فرمت خروجی داده‌های JSON را مشخص نمایید، به شما امکان می‌دهد آبجکت‌های wrapper بسازید و خصوصیات پیچیده را تو در تو نمایید. اگر پرس وجوی شما شامل دو یا چند جدول است، به طور پیش فرض یک نتیجه یکنواخت را باز می‌گرداند بطوریکه هر ستون در نهایت خصوصیتی از آبجکت JSON می‌شود. شما می‌توانید برای نتایج تودرتو از نام‌های ستونی جداشده با نقطه استفاده کنید؛ در مورد زیر ما از Order به عنوان نام آبجکت استفاده کردیم و نتیجه این شد:

```
SELECT H.SalesOrderID AS ‘Order.OrderID’,
H.Status AS ‘Order.Status’,
H.PurchaseOrderNumber AS ‘Order.PONumber’,
H.ShipDate AS ‘Order.ShipDate’,
P.ProductID AS ‘Order.ProductDI’
FROM [Sales].[SalesOrderHeader] H
INNER JOIN [Sales].SalesOrderDetail D ON
H.SalesOrderID = D.SalesOrderID
INNER JOIN [Production].[Product] P ON
D.ProductID = P.ProductID
WHERE H.SalesOrderID IN (43663, 43687)
FOR JSON PATH
```

شما می‌توانید با استفاده از نام‌های ستون جداشده با نقطه به همراه پسوند متفاوت یا نام آبجکت (با مشخص کردن نام‌های مستعار ستون که ساختار داده‌های JSON شما را تعریف می‌کنند) نتیجه را تودرتو کنید. برای مثال، در این مورد شما یک آبجکت تراز بالا دارید که دارای دو آبجکت Order و Product می‌باشد که در داخل تودرتو شده‌اند. با هر کدام از آبجکت ها ما برخی خصوصیات خاص را داریم که از داخل کپسول سازی شده‌اند.

```
SELECT H.SalesOrderID AS ‘Order.OrderID’,
H.Status AS ‘Order.Status’,
H.PurchaseOrderNumber AS ‘Order.PONumber’,
H.ShipDate AS ‘Product.ShipDate’,
P.ProductID AS ‘Product.ProductID’
FROM [Sales].[SalesOrderHeader] H
INNER JOIN [Sales].SalesOrderDetail D ON
H.SalesOrderID = D.SalesOrderID
INNER JOIN [Production].[Product] P ON
D.ProductID = P.ProductID
WHERE H.SalesOrderID IN (43663, 43687)
FOR JSON PATH
GO
```

نتیجه ا ستفاده از نام‌های ستون جدا شده با نقطه به طور پیش فرض، خروجی JSON شامل یک عنصر ریشه نیست و بنابراین می توانید از کلمه کلیدی ROOT با عبارت FOR JSON AUTO یا FOR JSON PATH استفاده کنید تا یک عنصر تراز بالا و منحصر را به خروجی JSON بیافزایید. مثلاً با پرس وجوی زیر یک عنصر تراز بالای Orders را اضافه می‌کنیم و نتیجه به صورت یک آرایه از داخل تودرتو خواهد شد، همانطور که در تصویر زیر می‌بینید.

```
SELECT H.SalesOrderID AS ‘Order.OrderID’,
H.Status AS ‘Order.Status’,
H.PurchaseOrderNumber AS ‘Order.PONumber’,
H.ShipDate AS ‘Product.ShipDate’,
P.ProductID AS ‘Product.ProductID’
FROM [Sales].[SalesOrderHeader] H
INNER JOIN [Sales].SalesOrderDetail D ON
H.SalesOrderID = D.SalesOrderID
INNER JOIN [Production].[Product] P ON
D.ProductID = P.ProductID
WHERE H.SalesOrderID IN (43663, 43687)
FOR JSON PATH, ROOT (‘Orders’)
GO
```

نتایج تودرتو شده داخل یک آرایه

همانند ROOT ، شما می توانید از کلمه کلیدی INCLUDE_NULL_VALUES با عبارت FOR JSON AUTO یا FOR JSON PATH استفاده کنید تا مقادیر null (هماند خصوصیات JSON) را در خروجی JSON بگنجانید، چراکه آن‌ها به طور پیش فرض در خروجی گنجانده نشده‌اند.

توجه: ویژگی ذکر شده و به نمایش درآمده در این مقاله بر مبنای SQL Server 2016 CTP 3.2 است و ممکن است در RTM موجود یا در نسخه‌های بعدی تغییر کند. نتیجهSQL Server 2016پشتیبانی توکار برای ذخیره سازی، مدیریت و تجزیه داده‌های JSON را معرفی می‌کند. در این مقاله در مورد پشتیبانی از JSON در SQL Server 2016 صحبت کردیم و نحوه فرمت کردن یا تبدیل داده‌های جدولی به فرمت JSON با استفاده ازعبارت FOR JSON را عنوان نمودیم. همچنین نگاهی داشتیم به انواع مختلف FOR JSON ،ا ستفاده از AUTO برای دریافت خودکار ساختار داده‌های JSON بر اساس ترتیب جدول‌های منبع و ستون‌ها در پرس و جو یا با استفاه از PATH برای داشتن کنترل کامل در تعیین ساختار خروجی داده‌های JSON.

هشت اشتباه متداول در

SQL SERVER

نصب و راه اندازی SQL Server به واسطه وجود Wizard Installation توسط مایکروسافت در این سالها بسیار آسان شده است. به صورت پایه ای چند عملیات باید قبل و بعد از نصب SQL Server بر روی سیستم سخت افزاری و تنظیمات SQL Server توسط مدیران پایگاه داده انجام شود تا از نبود اشکالات پایه ای اطمینان حاصل نمایند. بر اساس تجربه کاری بنده برخی از مشکلات اساسی SQL Server از انجام ندادن این عملیات قبل و بعد از نصب و راه اندازی است.

اشتباه ۱: نصب و راه اندازی SQL Server بدون انجام تست های سخت افزاری و نرم افزاری بر روی دیسک سخت.

اصولا مدیران پایگاه داده باید سیستم سخت افزاری اعم از دیسک سخت و قدرت پردازنده را برای استفاده در محیط SQL Server ارزیابی کنند و این ارزیابی باید دقیقاً به صورت باشد که SQL Server از این دو منابع استفاده می کند.

اشتباه ۲: استفاده از تنظیمات پیشفرض در SQL Server.

مدیران پایگاه داده در همه حال از تنظیمات پیشفرض در SQL Server استفاده میکنند. همیشه به یاد داشته باشید که تنظیمات در هر محیط سخت افزاری و طرز استفاده از SQL Server متفاوت است.

اشتباه ۳: استفاده از Primary Filegroup در پایگاه های داده

مدیران پایگاه داده اصولا باید تمامی داده های کاربران را از داده های سیستمی جدا کنند. این عمل چند فواید به همراه دارد.

اشتباه ۴: قرار دادن فایل داده و تراکنش در یک درایو.

قرار دادن این دو فایل بر کاهش سرعت تراکنش و استفاده از منابع سیستم تأثیر بسیاری دارد.

اشتباه ۵: کم حجم کردن فایل تراکنش.

این بدترین عملی است که یک مدیر پایگاه داده می تواند انجام دهد به این صورت که اول Recovery Model را به Simple تغییر داده و بعد فایل تراکنش را کاهش داده و بعد Recovery Model را به Full تغییر داده و در آخر بدون گرفتن Backup پایگاه داده را به امید خدا رها کند. این عمل زنجیره تراکنش را در پایگاه داده از بین میبرد و باعث می شود که در هنگام اختلال و خرابی دیگر نتوانیم داده ها را تا زمان قبل از خرابی بازیابی کنیم.

اشتباه ۶: فعال سازی Auto_Close در پایگاه داده

این تنظیمات استفاده از دیسک سخت را افزایش داده و سرعت کلی سیستم را پایین می آورد.

اشتباه ۷: استفاده و فعال سازی Auto_Growth در پایگاه داده.

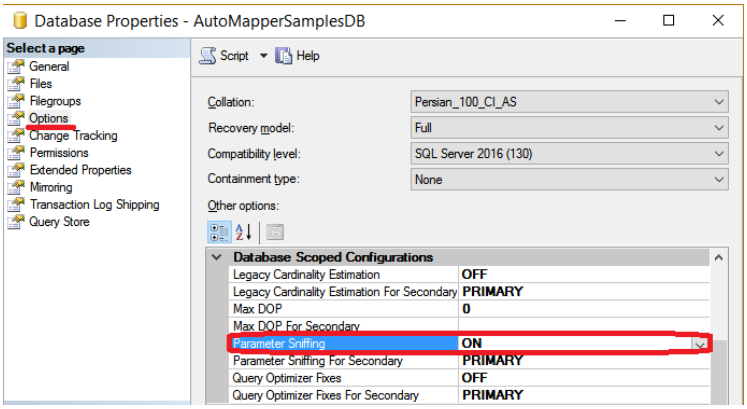
این تنظیمات باعث میشود که فایل تراکنش به صورت اصولی و مرتب ساخته نشود.

اشتباه ۸: گرفتن فایل پشتیبان پایگاه داده بدون تست کردن.

گرفتن فایل پشتیبان بر اساس RTO و RPO به تسویب رسیده از طرف مدیریت سازمان بسیار کار پسندیده و عالی است اما در خیلی از موارد مدیران پایگاه داده فایل پشتیبان را تست نکرده و در هنگام بازیابی بعد از اختلال یا خرابی به خطا هایی همچون «فایل پشتیبان خراب است» برخورد می کنند که دیگر برای دانستن این موضوع خیلی دیر است.

قابلیت PARAMETER SNIFFING در SQL 2016

در اس کیو ال سرور ۲۰۱۶، قابلیت غیر فعال نمودن parameter sniffing در سطح بانک اطلاعاتی مهیا شده است. اما چرا؟



قبل از پاسخگویی به سؤال بالا، به یک سری مقدمات نیاز است:

وقتی یک کوئری به اس کیو ال ارسال می‌شود، چه اتفاقی رخ می‌دهد؟

وقتی یک کوئری ار سال می شود، تعدادی از پرو سس‌ها بر روی کوئری شروع به فعالیت هایی مانند مهیا نمودن داده های بازگشتی، یا ذخیره سازی و ... می‌کنند.
پروسس‌ها به دو دسته زیر تقسیم می‌شوند:

۱. پروسس‌هایی که در relational engine رخ می‌دهند
۲. پروسس‌هایی که در storage engine رخ می‌دهند

در relational engine، هر کوئری pars شده و سپس بوسیله query optimizer پردازش و پلن اجرایی (execution plan) آن که بفرمت باینری است، ایجاد می‌شود و به storage engine ارسال می‌گردد. در storage engine پروسس‌هایی مانند قفل گذاری، نگهداری ایندکس‌ها و تراکنش‌ها رخ می‌دهد. هنگامیکه اس کیو ال سرور کوئری را دریافت می‌نمایند، آن را بلافا صله به relational engine ارسال می‌کند. سپس نحو (syntax) آن بررسی می‌شود؛ این عمل query parsing نامیده می‌شود. خروجی عملیات پارسر، یک ساختار درختی (query tree) است.

این ساختار درختی مشخص کننده مراحل لازم جهت اجرای کوئری ارائه شده می‌باشد.

اگر یک کوئری شامل DML نباشد (مانند ساخت جدول)، علمیات بهبود برروی آن صورت نخواهد گرفت. ولی در صورتیکه کوئری ارسالی، DML باشد، درخت اشاره شده در بالا به algebrizer فرستاده می‌شود که وظیفه آن تفسیر و بررسی کلیه نام ا شیاء، جداول و ستون‌های ا شاره شده در متن کوئری است. فرآیند algebrizer بسیار مهم و حیاتی است؛ بدلیل اینکه در کوئری ممکن است اشاره کننده‌هایی به اشیایی باشند که در بانک اطلاعاتی موجود نیست. خروجی algebrizer یک query processor tree باینری ا ست که به بهبود دهنده کوئری ار سال می‌گردد.

معرفی بهبود دهنده پرس و جو (Query Optimizer)

بهبود دهنده، بهترین مسیر اجرای کوئری را مشخص می‌کند. این بهبود دهنده ا ست که م شخص می‌کند که اطلاعات بو سیله ایندکس دریافت شوند، یا اینکه از چه اتصالی استفاده شود و الی آخر. این تصمیمات براساس محاسبات هزینه‌های (میزان پردازش لازم cpu و I/O) پلن اجرایی صورت خواهد پذیرفت. بهمین دلیل به پلن cost-based نیز شناخته می‌شود.

هنگامیکه کوئری ساده‌ای مانند دریافت اطلاعات از یک جدول، که بر روی آن ایندکس گذاری انجام نشده‌است، ارسال شود، بهبود دهنده بجای مشخص نمودن یک پلن مناسب بهینه، از یک پلن ساده (trivial) استفاده می‌کند. ولی برعکس در صورتیکه کوئری trivial نباشد (یعنی مثلا کوئری به گونه‌ای با شد که از ایندکس‌ها به شکل صحیحی ا ستفاده شده با شند)، بهبود دهنده یک پلن مناسب را براساس اطلاعات آماری مهیا شده در اس کیو ال سرور، تولید و انتخاب می‌نماید.

اطلاعات آماری از ستون‌ها و ایندکس‌ها جمع آوری می شود. این اطلاعات شامل نحوه توزیع داده، یکتایی و انتخاب شوندگی است. این اطلاعات تو سط یک histogram ارائه می شود. اگر اطلاعات آماری برای یک ستون و یا ایندکس وجود داشته باشد، بهبود

دهنده از آن‌ها برای محاسبات خود استفاده خواهد کرد. اطلاعات آماری بصورت خودکار برای تمام ایندکس‌ها و یا هر ستونی که بشود بر روی آن‌ها where یا join نوشت، فراهم خواهد شد.

بهبود دهنده با مقایسه پلن‌ها براساس بررسی تفاوت‌های انواع joinها، چیدمان مجدد ترتیب join و بررسی ایندکس های مختلف و سایر فعالیت‌های دیگر، پلن منا سب را انتخاب و از آن استفاده می‌کند. در طی هر کدام از فعالیت‌های اشاره شده، زمان اجرای آن‌ها نیز تخمین زده (estimated cost) خواهد شد و در پایان، زمان کل تخمینی بدست خواهد آمد و بهبود دهنده از این زمان برای انتخاب پلن مناسب بهره خواهد برد. باید توجه داشت که این زمان تقریبی است. زمانیکه بهبود دهنده پلن اجرایی انتخاب می‌کند، یک actual plan را ایجاد و در حافظه ذخیره می شود؛ بنام plan cache. البته درصورتیکه پلن مشابه و بهینه‌تری وجود نداشته باشد.

استفاده مجدد از پلن ها

تولید پلن هزینه بر است. به‌همین دلیل اس کیوال سرور اقدام به ذخیره سازی و نگهداری آن‌ها می‌کند تا بتواند از آن ها مجددا استفاده نماید؛ البته تا جایی که مقدور باشد. هنگامیکه آن‌ها تولید می‌شوند، در قسمتی از حافظه بنام plan cache ذخیره می‌شوند. به این عمل procedure cache نیز گفته می‌شود.

هنگامیکه کوئری به سرور ارسال می شود، بوسیله بهبود دهنده، یک estimated plan ایجاد خواهد شد و قبل از اینکه به storage engine ارسال شود، بهبود دهنده estimated plan را با actual execution planهای موجود در plan cache مقایسه می‌کند. در صورتیکه یک actual plan را مطابق با estimated plan پیدا نماید، از آن مجدد استفاده خواهد کرد. این استفاده مجدد به عدم تحمیل سربار اضافه‌ای به سرور جهت کوئری‌های بزرگ و پیچیده که در زمان واحد، هزاران بار اجرا خواهند شد، منجر می‌شود.

هر پلن فقط یکبار در حافظه ذخیره خواهد شد. ولی در مواقعی با تشخیص بهبود دهنده و هزینه پلن، یک کوئری می‌تواند پلن دیگری نیز داشته باشد. بنابراین پلن دوم نیز با مجموعه عملیاتی متفاوت، جهت اجرای موازی (parallel execution) برای یک کوئری ایجاد و در حافظه ذخیره می شود. پلن های اجرایی برای همیشه در حافظه باقی نخواهند ماند. پلن‌های اجرایی دارای طول عمری طبق فرمول حاصل ضرب هزینه، در تعداد دفعات می‌باشند. مثلاً پلنی با هزینه ۱۰ و تعداد دفعات اجرای ۵، طول عمر ۵۰ را خواهد داشت. پروسس lazywriterکه یک پروسس داخلی است وظیفه آزاد سازی تمام انواع کش ها، از جمله پلن کش را دارد. این پروسس در بازه‌های مشخص، تمام اشیاء درون حافظه را بررسی کرده و یک واحد از طول عمر آن‌ها می‌کاهد.

در موارد زیر، یک پلن از حافظه پاک خواهد شد:

۱. ۱ به حافظه بیشتری نیاز باشد
۲. طول عمر پلن صفر شده باشد

حال فرض کنید شما یک پروسیجر یا یک کوئری پارامتری دارید (پارامتر ورودی: شناسه سفارش یا نال) که کلیه محصولات

سفارش داده شده یا محصولات یک سفارش خاص را نمایش می‌دهد. هنگامی که SQL Server optimizer پلن این کوئری را ایجاد می‌کند و یا آن را کامپایل می‌کند، به پارامترهای ورودی این پروسیجر گوش می‌دهد (نال یا یک شناسه سفارش). optimizer بوسیله column statistics از تعداد رکوردهایی که بازگشت داده می‌شود، برآوردی می‌کند (مثلا ۴۰ رکورد). سپس یک پلن مناسب را انتخاب می‌کند و آن را برای اجرا ارسال می‌کند و پلن را ذخیره می‌نماید.جمله آخر، معمولا باعث ایجاد مشکل می‌شوند.

اگر optimizer تکست کوئری مشابهی را مشاهده نماید، ولی با پارامترهای متفاوت، به کش پلن مراجعه کرده و اگر در آن جا قرار داشت، از آن مجددا استفاده می‌نماید. این استفاده مجدد خوب است؛ اما در صورتیکه پارامتر ارسالی نال باشد چه اتفاقی رخ می‌دهد؟ جدول سفارشات محصول بسیار حجیم است و متاسفانه از پلنی که برای بازگشت ۴۰ رکورد قبلا ایجاد شده، برای بازگشت این حجم بالای از رکوردها استفاده می‌شود که این کشنده است.

هیچ تضمینی وجود ندارد که از وقوع این اتفاق جلوگیری نمایید؛ اما می‌توانید در هنگام توسعه، پروسیجر را شناسایی و نسبت به رفع آنها اقدام نمایید. ابتدا کش پلن را خالی نمایید و سپس پروسیجر را با مقادیر متفاوت، اجرا نمایید. در صورتیکه پلن‌های متفاوتی مشاهده نمودید، این یک علامت هشدار است و می‌بایست نسبت به رفع آن‌ها اقدام فوری نمایید.

اس کیو ال سرور بوسیله ایجاد پلن‌های اجرایی کامپایل شده، سعی در بهینه سازی پروسیجرها دارد. هنگامیکه اس کیو ال سرور یک پروسیجر را کامپایل می‌نماید، به پارامترهای ارسال شده توجه دارد و یک پلن اجرایی را بر اساس پارامترهای ارسالی ایجاد می‌کند. به فرآیند تما شا یا توجه به پارامترهای ار سالی به پروسیجر، شنود پارامترها گفته می‌شود. شنود پارامترها می‌تواند بعضی از اوقات به کاهش کارآیی پلن اجرایی منجر شود؛ مخصوصا زمانیکه پارامترهایی با کاردینالیتی متفاوت، فراخوانی شوند.

شنود پارامتر چیست؟

شنود پارامتر، قابلیتی است که اس کیو ال سرور توسط آن یک پلن مناسب و بهینه اجرایی را برای پروسیجری با پارامترهای ارسالی، در اولین مرتبه اجرا، تولید خواهد کرد. در ادامه هر فراخوانی پروسیجر با پارامترهای مشابه، منجر به استفاده از پلن اجرایی ذخیره شده خواهد شد. شاید در اولین نگاه این استفاده مجدد، مناسب به نظر برسد. ولی در صورتیکه پروسیجر با پارامترهای متفاوتی فراخوانی شود، ممکن است پلن اجرایی تولید شده بر اساس آن پارامترهای اولیه، برای پارامترهای جدید بهینه نباشد.

پلن‌های اجرایی بر اساس چیزهایی که از SQL Server خواسته می‌شوند، بهینه سازی می‌شوند. اس کیو ال سرور کوئری را بررسی کرده و یک استراتژی بهینه را برای اجرای آن مشخص می‌کند. به کارهایی که کوئری قرار است انجام دهد نگاه می‌کند؛ از مقادیر پارامترها برای استفاده از اطلاعات آماری استفاده کرده و محاسباتی را انجام خواهد داد.

مثالی از مصرف شدید I/O بدلیل شنود پارامتر

در ادامه، برای درک بهتر شنود پارامتر، با مثالی خواهید دید که پروسیجر ذیل، باعث مصرف بالای منابع، بر اساس پارامترهای ارسالی خواهد شد. در این مثال دو دسته متفاوت پارامتر برای اجرای پروسیجر ارسال خواهند شد و خواهید دید که فراخوانی دوم، منابع I/O بیشتری را نسبت به فراخوانی اول، مصرف خواهد کرد. در ادامه کدهای جدولی را که پروسیجر قرار است بر روی آن فراخوانی اطلاعات را انجام دهد، می‌بینید.

```
SET NOCOUNT ON;
DROP TABLE BillingInfo;
CREATE TABLE BillingInfo)
ID INT IDENTITY,
BillingDate DATETIME,
BillingAmt MONEY,
BillingDesc varchar(500;((
```

```
DECLARE @I INT;
DECLARE @BD INT;
SET @I = 0;
WHILE @I < 1000000
BEGIN
  SET @I = @I + 1;
  SET @BD=CAST(RAND()*10000 AS INT)%3650;
  INSERT BillingInfo (BillingDate, BillingAmt(
  VALUES (DATEADD(DD,@BD,
    CAST('1999/01/01' AS DATETIME,((
    RAND()*5000;(
END

ALTER TABLE BillingInfo
  ADD  CONSTRAINT [PK_BillingInfo_ID[
  PRIMARY KEY CLUSTERED (ID;(

CREATE NONCLUSTERED INDEX IX_BillingDate
  ON dbo.BillingInfo(BillingDate;(
```

در جدول BilingInfo بالا، یک میلیون رکورد با مقادیر BilingDate و BilingAmt به صورت تصادفی ایجاد شده است. بر روی ستون ID، ایندکس خوشه‌ای و ستون ایندکس غیر خوشه‌ای بر روی ستون BilingDate ایجاد شده‌است.

بو سیله پرو سیجر زیر هم قرار ا ست اطلاعات درخوا ستی فراهم شود:

```
CREATE PROC [dbo].[DisplayBillingInfo[
@  BeginDate DATETIME,
@  EndDate DATETIME
AS
SELECT BillingDate, BillingAmt
  FROM BillingInfo
  WHERE BillingDate between @BeginDate AND
@EndDate;
```

سپس پروسیجر را ۲ بار، با مقادیر پارامترهای متفاوتی اجرا می‌کنیم:

```
SET STATISTICS IO ON;
DBCC FREEPROCCACHE;
EXEC dbo.DisplayBillingInfo
@  BeginDate = '1999-01-01,'
@  EndDate  = '1999-12-31;'
```

```
EXEC dbo.DisplayBillingInfo
@  BeginDate = '2005-01-01,'
@  EndDate  = '2005-01-03;'
```

اطلاعات آماری I/O رو شن ا ست و اطلاعات I/O در هر بار اجرا، نما یش داده می‌شود. در خط دوم توسط DBCC FREEPROCCACHE، پلن کش خالی خوا هد شد؛ جهت اطمینان از عدم وجود پلن اجرایی مشابهی.

در فراخوانی اول، اطلاعات در بازه یک سال و در فراخوانی دوم، در بازه چند روز، درخوا ست شده‌اند. همانطور که گفته شد، پلن اجرایی بر ا ساس فراخوانی اول ایجاد خواهد شد و فراخوانی دوم نیز براساس همین پلن اجرایی ایجاد شده، اجرا می‌شود.

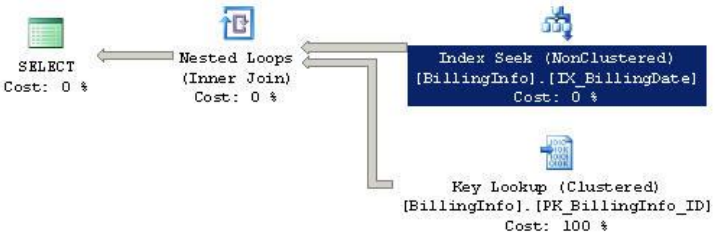
همانطور که مشاهده می‌کنید عملیات Clustered Index Scan، اجرا شده و اطلاعات I/O نیز بشرح زیر است (خط اول فراخوانی اول، خط دوم فراخوانی دوم):

```
Table 'BillingInfo'. Scan count 1, logical reads 3593,
physical reads 0, read-ahead reads 0, lob logical reads
.0, lob physical reads 0, lob read-ahead reads 0

Table 'BillingInfo'. Scan count 1, logical reads 3593,
physical reads 0, read-ahead reads 0, lob logical reads
.0, lob physical reads 0, lob read-ahead reads 0
```

حال ترتیب فراخوانی‌ها را به‌شرح زیر جابجا می‌کنیم:

```
SET STATISTICS IO ON;
DBCC FREEPROCCACHE;
EXEC dbo.DisplayBillingInfo
@  BeginDate = '2005-01-01,'
@  EndDate  = '2005-01-03;'

EXEC dbo.DisplayBillingInfo
@  BeginDate = '1999-01-01,'
@  EndDate  = '1999-12-31;'
```

در کد بالا ابتدا فراخوانی که بازه کوچکتری دارد اجرا و پلن اجرایی بر اساس آن ایجاد خواهد شد و فراخوانی دوم از پلن کش شده استفاده می‌کند.



اکنون عملیات Index Seek را بجای Index Scan مشاهده می‌کنید. اطلاعات I/O هم بشرح زیر است:

Table 'BillingInfo'. Scan count 1, logical reads 2965, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'BillingInfo'. Scan count 1, logical reads 337040, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

اکنون فراخوانی اول بهینه‌تر اجرا شده و بجای Index Scan از Index Seek استفاده کرده‌است و logical reads آن کاهش یافته‌است. در حالیکه فراخوانی دوم که در بازه یکسال اجرا شده‌است، با عملیات هزینه بر I/O بیشتری نسبت به آزمایش (حدود ۱۰۰ برابر) انجام شده‌است. انجام این ۲ آزمایش، پلن اجرایی متفاوتی را بر اساس پارامترهای ورودی ایجاد کرد و هزینه‌های I/O آن را مشاهده کردید. اکنون درک خوبی را نسبت به این قابلیت و اثرات آن خواهید داشت و در ادامه به راه حل‌هایی جهت کاهش اثرات منفی و مقابله با آن‌ها اشاره خواهد شد.

نکته: راه‌های اشاره شده برای مقابله با شنود پارامترها برای تمام شرایط قابل استفاده نیستند.

راه حل اول: استفاده از دستور With Recompile

مشکل شنود پارامتر این است که در اولین اجرای پروسیجر، پلن اجرایی را بر اساس پارامترهای ارسالی اولیه ایجاد می‌کند. راه حل غلبه بر این مشکل، کامپایل مجدد پروسیجر، بعد از هر اجرای آن است. بهمین جهت از دستور WITH RECOMPILE هنگامیکه قصد ایجاد پروسیجر را دارید استفاده نمایید. مانند کد زیر:

```
CREATE PROC [dbo].[DisplayBillingInfo]
@ BeginDate DATETIME,
@ EndDate DATETIME
WITH RECOMPILE
AS
SELECT BillingDate, BillingAmt
  FROM BillingInfo
  WHERE BillingDate between @BeginDate AND
@EndDate;
```

مجددا ۲ آزمایش اشاره شده در مطلب قبلی (اشاره شده در زیر) را تکرار می‌کنیم.

```
DBCC FREEPROCCACHE;
EXEC dbo.DisplayBillingInfo
@ BeginDate = '2005-01-01,'
@ EndDate  = '2005-01-03;'

EXEC dbo.DisplayBillingInfo
@ BeginDate = '1999-01-01,'
@ EndDate  = '1999-12-31;'
```

هنگامیکه کد بالا را اجرا نمایید، فراخوانی اول، عملیات Index Seek و فراخوانی دوم، Index Scan را موجب خواهد شد. نقص این روش، کامپایل مجدد با هر بار اجرای پروسیجر است که باعث تحمیل سربار اضافه‌ای می‌شود.

راه حل دوم: غیر فعال نمودن شنود پارامتر

روش دیگر برطرف کردن مشکلات مرتبط با شنود پارامتر، غیر فعال کردن آن است. البته منظور از غیر فعال کردن، غیر فعال نمودن گزینه‌ای در بانک اطلاعاتی نیست؛ بلکه با تغییر متن و نحوه‌ی اجرا، می‌توان شنود را غیر فعال نمود. در کد زیر با تغییر نحوه اجرای پروسیجر، قابلیت شنود پارامتر غیر فعال شده است:

```
CREATE PROC [dbo].[DisplayBillingInfo]
```

@ BeginDate DATETIME,
@ EndDate DATETIME
WITH RECOMPILE
AS
DECLARE @StartDate DATETIME;
DECLARE @StopDate DATETIME;
SET @StartDate = @BeginDate;
SET @StopDate = @EndDate;
SELECT BillingDate, BillingAmt
FROM BillingInfo
WHERE BillingDate between @StartDate AND @StopDate;

برای غیرفعال نمودن، تمام کاری که انجام شده، نحوه استفاده از پارامترهای ارسالی تغییر داده شده است. در کد بالا دو متغیر محلی با نامهای @StartDate و @EndDate ایجاد شدها ست. پارامترهای ارسالی درون متغیرهای محلی ذخیره می‌شوند و سپس از متغیرهای محلی در شرط between استفاده خواهد شد. بدین صورت شنود غیر فعال می‌شود. دلیل غیر فعال شدن شنود این است که بهبود دهنده (optimizer) قادر به شناسایی مقادیر پارامترهای ورودی در بدنه دستور Select نمی‌باشد. بدلیل عدم رهگیری محل مصرف مقادیر پارامترهای ارسالی توسط اس کیو ال سرور، بهبود دهنده یک پلن جنریک را براساس اطلاعات آماری ایجاد خواهد کرد.

راه حل سوم: ایجاد چند نوع پروسیجر

راه دیگر، ایجاد پروسیجر های متفاوت برای پارامتر هایی با کاردینالیتی متفاوت است. به‌عبارت دیگر، دسته بندی پارامترهای ارسالی و ایجاد پروسیجرهایی خاص همان دسته. در مثال‌های این سری از مطالب، دو دسته پارامتر ۱) بازه زمانی کوتاه، مثلا چند روز و ۲) بازه زمانی بلند، مثلا ماهیانه وجود داشت که می‌توانید ۲ دسته پروسیجر را یکی برای بازه‌های روزانه و دیگری برای بازه‌های زمانی ماهیانه ایجاد نمایید.

```
CREATE PROC [dbo].[DisplayBillingInfoNarrow]
@ BeginDate DATETIME,
@ EndDate DATETIME
AS
SELECT BillingDate, BillingAmt
  FROM BillingInfo
  WHERE BillingDate between @BeginDate AND
@EndDate;
GO
CREATE PROC [dbo].[DisplayBillingInfoWide]
@ BeginDate DATETIME,
@ EndDate DATETIME
AS
SELECT BillingDate, BillingAmt
  FROM BillingInfo
  WHERE BillingDate between @BeginDate AND
@EndDate;
GO
DROP PROCEDURE [dbo].[DisplayBillingInfo];
GO
CREATE PROC [dbo].[DisplayBillingInfo]
@ BeginDate DATETIME,
@ EndDate DATETIME
AS
IF DATEDIFF(DD,@BeginDate, @EndDate) < 4
  EXECUTE DisplayBillingInfoNarrow @BeginDate,
@EndDate
ELSE
  EXECUTE DisplayBillingInfoWide @BeginDate,
@EndDate
GO
```

در کد بالا، دو گروه پروسیجر (برای بازه زمانی کوتاه و بلند) به‌همراه یک پروسیجر تصمیم گیرنده جهت تشخیص استفاده از پروسیجر مناسب، بر اساس پارامترهای ورودی ایجاد شده است. یکی از مزایای این روش استفاده پروسیجر از پلن اجرایی مناسب، فارغ از پارامترهای ارسالی خواهد بود. البته نگهداری کد در این روش به مرور زمان، کمی دشوار و سخت خواهد شد.

اتصال برنامه های Android به SQL Server

تسـهیل اسـتفاده از اطلاعات بانک های اطلاعاتی در پلتفرم های مختلف می تواند یکپارچگی مناسـبی را برای برنامه های تولیدی ما به وجود آورد. تصور کنید شما یک سرویس شبکه اجتماعی دارید که نیاز است از طریق بسترهای مختلف کاربران بتوانند از آن بهره ببرند. با توجه با اینکه بسـترهای موجود از لحاظ پیاده سازی با یکدیگر متفاوتند چه راهکاری می تواند این ارتباط را تسهیل کند؟

تمرکز ما برروی اندروید به عنوان پرطرفدارترین پلتفرم موبایل است که نه تنها می تواند محبوبیت سرویس شما را صدچندان کند بلکه این قابلیت را دارد تا بعنوان پلتفرم اصـلی محسـوب شود.

اندروید و بانک اطلاعاتی

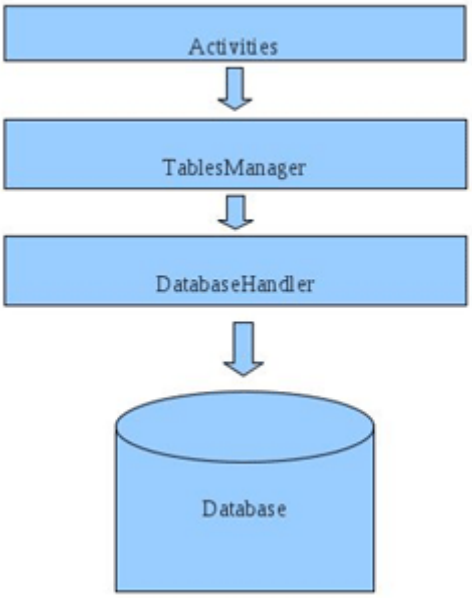
بصورت پیشفرض در اندروید برای ذخیره سازی داده از سه روش ذیل استفاده می شود:

۱- Shared Preferences

۲- File System(internal and external storage)

۳- SQL Lite

هریک از این روش ها در در جای خود می تواند بسیار کاربردی و مفیدتر از دیگر روش ها با شد. اما ذخیره سازی سازمان یافته در اندروید از طریق SQLite که یک نسخه بانک اطلاعاتی ساده اما بسیار قدرتمند است (در scale خود) ، صورت می پذیرد.



موضوع بحث ما اما روش های مرسوم ذخیره سازی اطلاعات در اندروید نیسـت، بلکه اتصـال به بانک اطلاعاتی قدرتمند SQL Server می باشد.

یک کار نو

به دو روش می تواند در اندروید از اطلاعات SQL Server بهره مند شد:

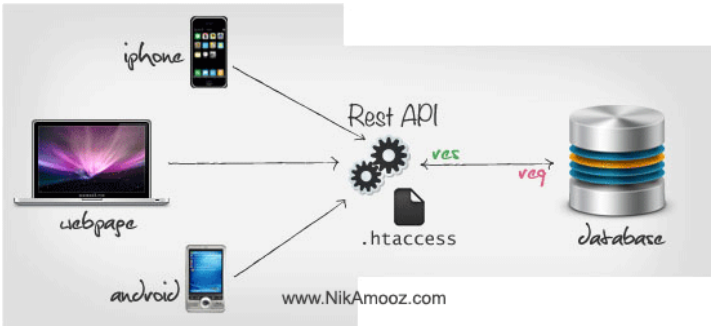
۱- پیاده سـازی Web Service مبتنی بر تکنولوژی هایی که بصورت بومی به SQL Server دسترسی دارند مثل Web Api در دات نت.

۲- اتصال مستقیم با استفاده از Connection String.

پیاده سازی وب سرویس ها جهت بهره برداری از اطلاعات بانک اطلاعاتی SQL Server

وب سرویس ها معمولا به سه سبک مختلف پیاده سازی می شوند : RPC، Service-oriented و REST

در این بین REST یا همان Representational state transfer ساده ترین رابط و بیشترین طرفدار را دارد. در این سبک با ایجاد افعال شناخته شده ای مثل DELETE، PUT، POST و GET بر روی Http ، کار با داده های ارائه شده توسط سرویس بسیار راحت و سریع خواهد بود.



ا ستفاده از وب سرویس ها در اندروید هم با ا ستفاده از متدهای بومی قابل پیاده سازی و هم بواسطه استفاده از کتابخانه های ارائه شد (بیشترین کتابخانه های ارائه شده برای اندروید بصورت Open Source هستند و قابلیت بسط دارند).

برای مثال یک نمونه بهره برداری از وب سرویس REST را با استفاده از کتابخانه Android Asynchronous Http Client بررسی می کنیم.

فرض کنید وب سرویس داریم که آدرس آن http://192.168.2.2:9999/useraccount/login/dologin ا ست و از بانک SQL Server ا ستفاده می کند و سرویس ارائه شده توسط آن نام کاربری (ایمیل) و کلمه عبور را دریافت کرده و سپس عملیات لاگین را انجام می دهد. ابتدا می بایست با استفاده از یک شی RequestParams پارامترهای خود را درج کنیم:

```
RequestParams params = new RequestParams;()

params.put("username", email;(

params.put("password", password;(
```

سپس تابع مربوط به ارسال درخواست را فراخوانی کنیم.

```
invokeWS(params(
```

این تابع به شرح ذیل می باشد:

```
public void invokeWS(RequestParams params){
AsyncHttpClient client = new AsyncHttpClient();
client.get("http://192.168.2.2:9999/useraccount/login/dologin",params ,new AsyncHttpResponseHandler() {
@Override
public void onSuccess(String response) {
JSONObject obj = new JSONObject(response);
if(obj.getBoolean("status")){
Toast.makeText(getApplicationContext(), "You are successfully logged in!", Toast.LENGTH_LONG).show();
}
else{
errorMsg.setText(obj.getString("error_msg"));
Toast.makeText(getApplicationContext(), obj.getString("error_msg"), Toast.LENGTH_LONG).show();
}
}
@Override
public void onFailure(int statusCode, Throwable error,
String content) {
if(statusCode == 404){
Toast.makeText(getApplicationContext(), "Requested resource not found", Toast.LENGTH_LONG).show();
}
else if(statusCode == 500){
Toast.makeText(getApplicationContext(), "Something went wrong at server end", Toast.LENGTH_LONG).show();
}
// When Http response code other than 404, 500
else{
Toast.makeText(getApplicationContext(),
"Unexpected Error occcured! [Most common Error: Device might not be connected to Internet]",
Toast.LENGTH_LONG).show();
}
}
});
}
```

در این تابع می بینیم که یک وهله از AsyncHttpClient ایجاد شده سپس با فراخوانی متد get این شی و ارائه پارامترهای لازم پاسخ از سرور دریافت می شود. که در صورت انجام عملیات بصورت موفقیت آمیز انجام شود در صورت صحت اطلاعات ارسالی پیام You are successfully logged in و در غیر اینصورت پیام خطا با نام error_msg نمایش داده می شود. اما اگر به هر صورت اگر فرآیند این درخواست به درستی صورت نگیرد، در حالی که کد پاسخ از سرور ۴۰۴ باشد پیام Requested resource not found و اگر کد پاسخ ۵۰۰ باشد Something went wrong at server end و در غیر اینصورت

Unexpected Error occcured! [Most common Error: Device might not be connected to Internet or remote server is not up and running

نمایش داده می شود.

اتصال مستقیم با استفاده از Connection String

می دانیم که برای برنامه نویسی بومی اندروید از زبان برنامه نویسی جاوا استفاده می شود. در جاوا برای اتصال به بانک های اطلاعاتی از JDBC بهره گرفته می شود. ابتدا می بایست یک کتابخانه JDBC برای SQL Serverداشته باشیم. اطلاع دارید که اندروید بصورت بومی فقط از SQLite پشتیبانی می کند.

پس ابتدا کتابخانه کد باز http://jtds.sourceforge.net را دریافت می کنیم.

حال که این کتابخانه را وارد برنامه خود کردید کافیست از کد زیر جهت اتصال به SQL Server کمک بگیرید:

در کد بالا که بسیار سلیس است، بعد از برقراری اتصال ،کوئری select * from TableName اجرا می شود و سپس ستون چهارم هر رکورد در Logcat نمایش داده می شود. شاید مواردی زیادی نبا شد که در آن نیاز به اتصال مستقیم از اندروید به SQL Server داشته باشیم. اما دانستن امکان این اتصال شاید قسمتی از طراحی چارت سیستمی مارا دچار دگرگونی کند.

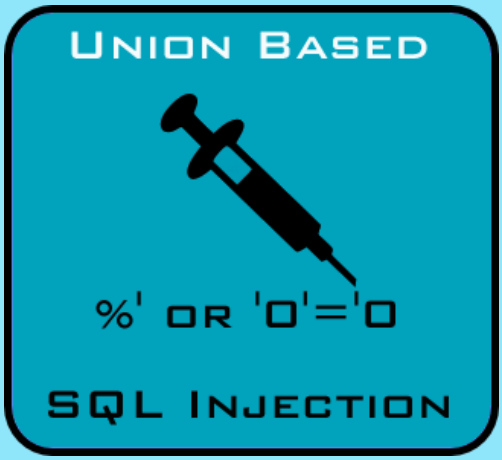
مزیت‌های استفاده از رویه‌های ذخیره شده؛ واقعیت یا توهم

۱. رویه‌های ذخیره شده در مقابل SQL Injection مقاوم هستند. کوئری‌های Ad hoc همیشه این آسیب پذیری را به همراه دارند. نادرست است! رویه‌های ذخیره شده‌ای که رشته‌ها را به صورت پارامتر دریافت کرده و آن‌ها را به صورت یک عبارت sql اجرا می‌کنند، آسیب پذیر هستند. اگر هنگام استفاده از کوئری‌های Ad hoc از پارامترها استفاده شود، در برابر حملات SQL Injection مصون خواهید بود.

۲. Executon planرویه‌های ذخیره شده کش می‌شوند اما این Plan برای کوئری‌های Ad hoc هر بار محاسبه و تولید می‌گردد. نادرست است! اس کیوال سرور تا این اندازه بی هوش نیست! اگر execution plan ایی موجود باشد حتما استفاده خواهد شد و برای موتور اس کیوال سرور اصلا اهمیتی ندارد که کوئری در حال اجرا از یک رویه ذخیره شده صادر شده است یا از یک کوئری Ad hoc . رویه‌های ذخیره شده پیش کامپایل شده نیستند و مانند تمامی کوئری‌های دیگر در زمان اجرا کامپایل می‌شوند.

۳. زمانیکه از رویه ذخیره شده استفاده می‌کنید همه چیز را در یک مکان به صورت متمرکز و مجتمع خواهید داشت (مدیریت بهتر) نادرست است! در یک مکان متمرکز در اختیار شما نیستند. برنامه جای خود را دارد و رویه‌های ذخیره شده در دیتابیس در جای دیگری قرار دارند و برای مثال اگر قرار باشد یک پارامتر را به رویه ذخیره شده خود اضافه کنید، کدهای شما نیز باید تغییر کنند.

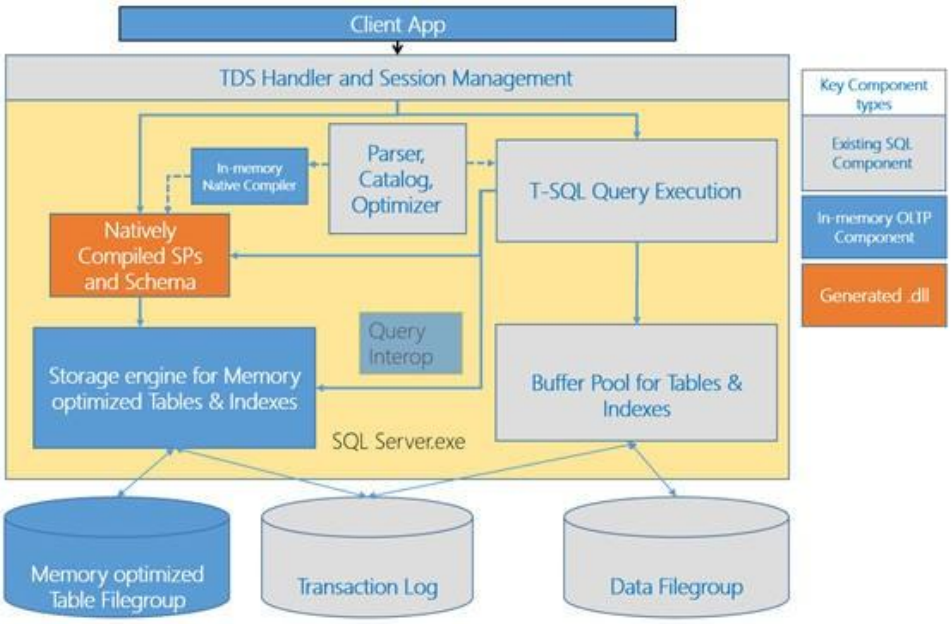
۴. می‌توان از یک رویه ذخیره شده استفاده مجدد کرد (در نقاط مختلف یک کد) و اعمال تغییرات تنها در یک مکان (دیتابیس) باید صورت گیرد.هر چند این مورد درست است، اما باید دقت داشت که اگر چندین برنامه از این رویه ذخیره شده استفاده می‌کنند نباید تغییرات شما باعث از کار افتادن سایر برنامه‌ها شوند.



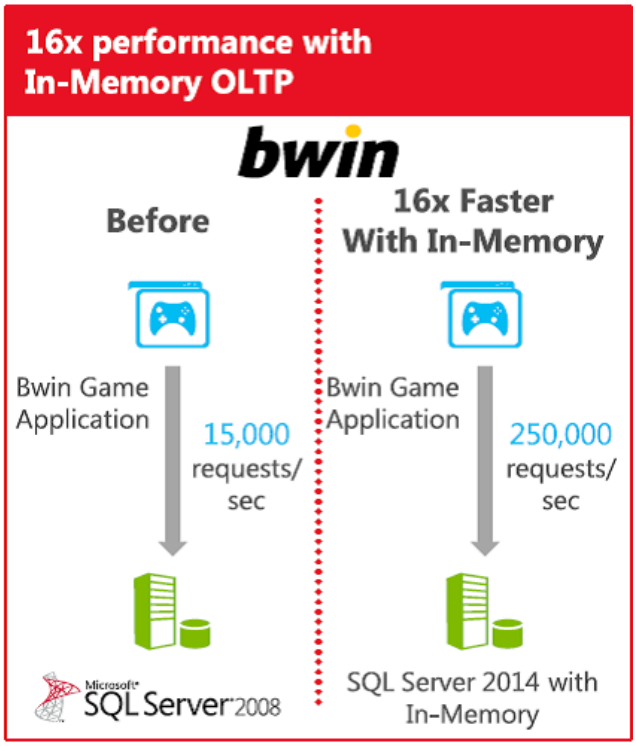
۵. می توان رویه ذخیره شده را بدون نیاز به توزیع مجدد برنامه تغییر داد. این مورد تا حدودی صحیح است. اگر تنها بحث بهینه سازی و امثال آن مطرح باشد صحیح است اما اگر واقعا نیاز به تغییر یک کوئری در رویه ذخیره شده وجود داشته باشد به احتمال زیاد برنامه نیز باید دستخوش تغییراتی گردد تا این دو با هم هماهنگ شوند.

آیا دوران پادشاهی اوراکل در حوزه‌ی مدیریت پایگاه‌های داده عملیاتی به پایان رسیده است؟

از سال ۱۹۷۰ تا به حال سیستم‌های مدیریت پایگاه داده عملیاتی – ODBMS - مختلفی ایجاد شده‌اند. بعضی از آنها به مرور زمان از بین رفته‌اند و برخی قدرتمندتر شده‌اند. در دهه‌های اخیر بین سیستم‌های مدیریت پایگاه داده عملیاتی، محصولات شرکت‌های اوراکل، مایکرو سافت، IBM و SAP از بقیه موفق تر بوده‌اند. اما مسلما در این بین بهترین سیستم مدیریت پایگاه داده، محصول شرکت اوراکل بوده است و سخن گزافی نیست که بگوییم محصول شرکت اوراکل در دهه های اخیر در بین محصولات دیگر شرکت ها پادشاهی می‌کرده است . تا حدود ۴ سال پیش بین کیفیت oracle db و sql server اختلاف فاحشی وجود داشت. چه از نظر سرعت و چه از نظر دیگر امکانات، اوراکل کاملا برتر از رقیب خود بود. در نسخه‌ی sql server 2012، امکانات قابل توجهی به محصول شرکت مایکروسافت افزوده شد. از مهمترین این امکانات می‌توان به ویژگی AlwaysOn و ColumnStore Index ها اشاره کرد. امکانات این نسخه باعث شد که اختلاف بین oracle db و sql server تا حدی کاهش یابد. مایکرو سافت سرانجام در نسخه‌ی sql server 2014 خود تغییرات اساسی بوجود آورد. مهمترین این تغییرات ایجاد موتور درونی In-Memory OLTP می‌باشد که برای تراکنش‌های درون حافظه بهینه شده است. با استفاده از امکانات این نسخه می‌توان بدون نیاز به دوباره نویسی محصولات، سرعت اجرای کوئری‌های آنها را به طور متوسط ده برابر کرد. در شکل ذیل ساختار جدید sql server مشاهده می‌شود.

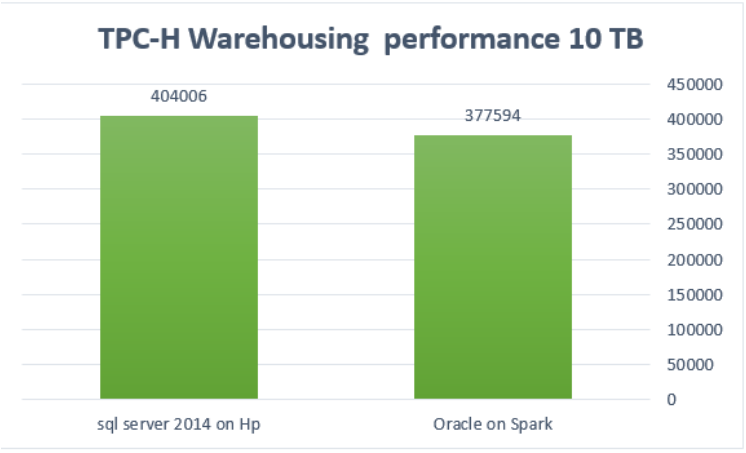


شرکت بوین که یک شرکت مشهور ارائه خدمات آنلاین و پیش بینی بازی‌های ورزشی است و در هر لحظه، کاربران آنلاین بسیاری در وب سایت شرکت، کوئری اجرا می‌کنند، از قابلیت‌های جدید اس کیو ال سرور ۲۰۱۴ استفاده کرده است و با استفاده از این قابلیت‌ها توانسته سرعت اجرای پرس و جوهای مشتریانش را از ۱۵ هزار پرس و جو در ثانیه به ۲۵۰ هزار پرس و جو در ثانیه برساند. در نتیجه کارایی سرور این شرکت ۱۶ برابر شده است.



در تحقیقی دیگر، یک محقق، با استفاده از قابلیت‌های جدید اس کیو ال سرور ۲۰۱۴ توانسته است دو رکورد جدید را از اجرای

کوئری‌های انبار داده ای برای حجم‌های ۳ ترابایت و ۱۰ ترابایت و نوع پارتیشن بندی نشده به ثبت بر ساند و رکوردهای قبلی را که متعلق به اوراکل بوده، بشکنند. این محقق تواند سته ۴۰۴۰۰۵ کوئری نسبتا سنگین انبار داده‌ای را در پایگاه داده‌ای با ۱۰ ترابایت اطلاعات، در یک ساعت اجرا کند و رکورد قبلی را که متعلق به اوراکل و برابر ۳۷۷۵۹۴ کوئری با همین شرایط بوده، بشکند. همچنین هزینه‌ی اجرای کوئری‌های سرور اس کیو ال مذکور برابر ۲,۰۴ دلار در هر ساعت اجرای کوئری بوده است. به این معنی است که کمتر از نصف هزینه‌ی مشابه در رکورد ثبت شده‌ی اوراکل که برابر ۴,۶۵ دلار در ساعت اجرای کوئری بوده است، هزینه داشته است.



3,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1	hp invent	DL580 G8	461,837	2.04 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14
2	ORACLE	SPARC T5-4 Server	409,721	3.94 USD	NR	09/24/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	06/07/13
3	cisco	Cisco UCS C420 M3 Server	230,119	1.29 USD	NR	12/30/13	Sybase IQ 16.0 SP02	Red Hat Enterprise Linux 6.4	10/31/13

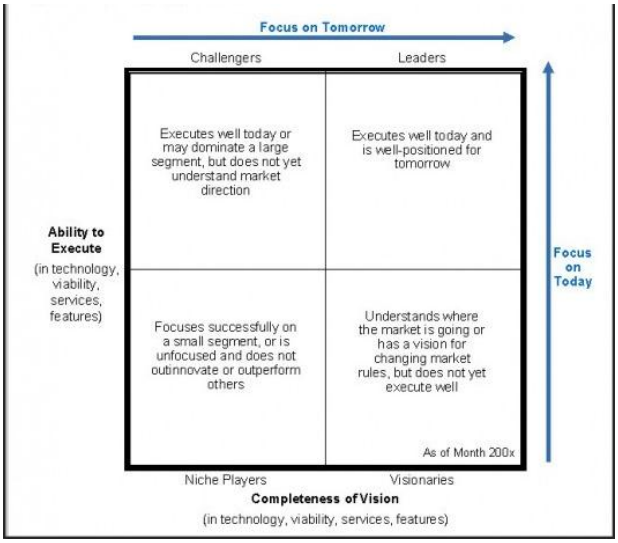
10,000 GB Results

Rank	Company	System	QphH	Price/QphH	Watts/KQphH	System Availability	Database	Operating System	Date Submitted
1	hp invent	DL580 G8	404,005	2.34 USD	NR	04/16/14	Microsoft SQL Server 2014 Enterprise Edition	Windows Server 2012 R2 Std Edition	04/15/14
2	ORACLE	SPARC T5-4 Server	377,594	4.65 USD	NR	11/26/13	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Solaris 11.1	11/25/13
3	hp invent	HP ProLiant DL980 G7	158,108	6.49 USD	NR	04/15/13	Microsoft SQL Server 2012 Enterprise Edition	Microsoft Windows Server 2012 Standard Edition	04/15/13

http://www.tpc.org/tpch/results/tpch_perf_results.asp?resulttype=noncluster&version=2%&currencyID=0

در واقع اگر بخواهیم سیستم‌های مدیریت پایگاه داده عملیاتی را رتبه بندی کنیم، به جز سرعت، باید عوامل مختلفی را در نظر بگیریم که چنین کاری نیاز به همکاری گروهی بزرگ دارد. خوشبختانه چنین گروه‌هایی وجود دارند و آن قدر معتبر هستند که اکثر شرکت‌های بزرگ به آمارهای آنها استناد می‌کنند. در فناوری‌های مربوط به آی تی، برای رسیدن به معتبرترین نتایج باید به گزارش‌های ارائه شده‌ی شرکت گارتنر رجوع کنیم. گارتنر، شرکت پژوهشی و مشاوره‌ی آمریکایی است، که در زمینه‌ی ارائه خدمات برون‌سپاری، تحقیق و پژوهش و مشاوره فناوری اطلاعات فعالیت می‌نماید. این شرکت در سال ۱۹۷۹ راه‌اندازی شد و در سال ۲۰۱۴ بیش از ۶۵۰۰ نفر کارمند داشته که در ۸۵ کشور بوده‌اند. در این بین حدود ۱۵۰۰ نفر از آنها در بخش تحقیق و توسعه فعالیت داشته‌اند. همچنین در این سال، درآمد شرکت گارتنر که عمدتا از طریق مشاوره دادن به شرکت‌های مختلف بوده، بیش از ۲ میلیارد دلار در سال ۲۰۱۴ بوده است.

شرکت گارتنر معمولا خلاصه‌ی نتیجه‌ی بررسی‌های خود را در نمودارهایی خاص به نام مربع جادویی گارتنر ارائه می‌کند. در این نمودار، قابلیت‌های اجرایی که بیانگر کیفیت فعلی محصول هستند، در محور عمودی نمایش داده می شوند و از پایین به بالا زیاد می شوند. یعنی هر چه محصولی بالاتر باشد، در حال حاضر کیفیت بهتری دارد. محور افقی نمودار بیانگر بصیرت و آینده نگری محصول می‌باشد و از چپ به راست زیاد می شود. به این ترتیب رهبران یک حوزه‌ی خاص، در ربع بالا و سمت راست مربع جای می‌گیرند.



حال که با نحوه‌ی تفسیر مربع جادویی گارتنر آشنا شدیم، به

بررسی نمودارهای مربوط به سیستم‌های مدیریت پایگاه داده عملیاتی در سه سال اخیر می پردازیم. در شکل ذیل می‌بینیم که در سال ۲۰۱۳ و پس از ارائه‌ی نسخه‌ی sql server 2012 توسط مایکروسافت، اوراکل همچنان پیشتاز است و شرکت‌های مایکروسافت، آی بی ام و SAP پس از آن قرار گرفته‌اند. البته در این سال شرکت مایکروسافت فاصله‌ی زیاد قبلی خود را با اوراکل، کم کرده است.

Source: Gartner (October 2013)

در سال ۲۰۱۴، شرکت مایکروسافت از نظر آینده نگری و بصیرت، از اوراکل پیشی گرفته ولی هنوز در قابلیت‌های اجرایی عقب‌تر از اوراکل قرار دارد.

اما چند روز پیش در تاریخ ۱۲ اکتبر ۲۰۱۵، شرکت گارتنر گزارشی ارائه کرد که خیلی از فعالان آی تی را شگفت زده کرد. این گزارش در حال حاضر در وب سایت شرکت گارتنر قابل دسترسی است؛ ولی معمولا گارتنر پس از مدتی آن را از حالت رایگان به پولی تغییر می‌دهد.

در گزارش سال ۲۰۱۵ و پس از ارائه‌ی نسخه‌ی sql server 2014 و کاربردی شدن و تست قابلیت‌های آن در عمل توسط شرکت‌های مختلف، بالاخره طلسم چند ده ساله‌ی اوراکل شکسته شده و اگرچه اوراکل نسبت به سال قبل رشد داشته است، ولی sql server مایکروسافت توانسته، هم در قابلیت اجرای فعلی و هم در بصیرت و آینده نگری بالاتر از محصول شرکت اوراکل بایستد. بنابراین عملا دوران پادشاهی مطلق اوراکل در حوزه‌ی پایگاه‌های داده‌ی عملیاتی به سر رسیده است.

در انتها لازم می‌بینم به نکاتی مهم اشاره کنم:

شرکت اوراکل بر خلاف تصور خیلی از افراد، همانند شرکت‌های مایکروسافت، آی بی ام و ... محصولات گسترده و مختلفی دارد و این بررسی و نتایج تنها در حوزه‌ی سیستم‌های مدیریت پایگاه داده عملیاتی بود.

- بالاتر بودن sql server مایکروسافت از اوراکل در سال ۲۰۱۵ به این معنا نیست که اوراکل نمی‌تواند به جایگاه قبلی خود برگردد؛ بلکه شاید در سال‌های آینده این رتبه بندی باز هم تغییر کند. در واقع این گزارش به این معنا است که فاصله‌ی زیاد

قدیم بین sql server و oracle db از بین رفته و در حال حا ضر این دو به رقیب سر سختی برای یکدیگر تبدیل شده‌اند.
وجود رقابت نزدیک بین شرکت‌های بزرگ باعث می شود که این شرکت‌ها حداکثر تلاش خود را برای بهتر کردن محصولات خود انجام بدهند و برندگان ا صلی این و ضعیت، ا ستفاده کنندگان از این محصولات هستند.

بررسی دستور Truncate Table و Delete

بررسی دستور Delete :
همانگونه که می‌دانیم از این د ستور برای حذف رکوردها ا ستفاده می‌کنند. با اجرای د ستور Delete به راحتی می‌توانید تعدادی از رکوردهای یک جدول را حذف کنید. ساده‌ترین شکل ا ستفاده از دستور Delete به صورت زیر می‌باشد.

```
DELETE FROM table_name
WHERE some_column=some_value
```

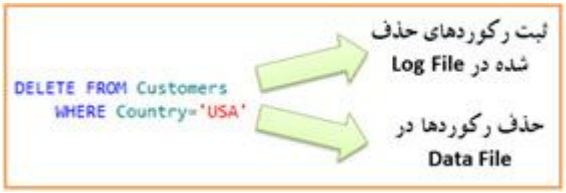
برای مثال در صـــورتیکه بخواهیم مشـــتریانی را حذف کنیم که ک شور (Country) آنها USA ا ست باید از د ستور زیر ا ستفاده کنیم.

```
DELETE FROM Customers
WHERE Country='USA'
```

با اجرای این د ستور هر رکوردی که در شرط مربوطه صدق کند حذف خواهد شد. (خوب این رو که همه می‌دانند)

اما نکته مهمی که د ستور Delete دارد این ا ست کار این د ستور به شـکل Transactional می‌باشـد. یعنی یا کلیه رکوردهایی که Country آنها USA اسـت حذف می‌شـود و یا هیچکدام از آنها. پس اگر شـما ۲۰۰۰۰۰ رکورد داشـته باشـید که در این شـرط صدق کند اگر وسط کار Delete (البته اگر عملیات حذف طولانی باشـد) منصـرف شـوید می‌توانید با Cancel کردن این دسـتور عملیات Rollback Transaction را به خودکار توسـط SQL Server داشـته باشـید. در صـورتیکه عملیات Cancel را انجام دهید SQL Server از Log File برای بازگرداندن مقادیر حذف شده استفاده خواهد کرد.

اما نکته دیگری که د ستور Delete دارد این است که این د ستور Log کلیه رکوردهایی را که قرار اسـت حذف کند در Log File می‌نویسد. این Log شامل ا صل رکورد، تاریخ و زمان حذف، نام کاربر و... می‌باشد. شاید الان متوجه شوید که دستور Delete چرا در برخی از مواقع که قرار ا ست حجم زیادی از اطلاعات را حذف نما ید به کُندی این کار را انجام می‌د هد. (چون با ید Log رکوردهای حذف شده در Log File نوشته شود.)



بررسی دستور Truncate Table:
Truncate در لغت به معنی بریدن و کوتاه کردن می‌باشـــد. با اســـتفاده دســـتور Truncate Table می‌توانید محتوای کلیه رکورد های موجود در یک جدول را در کســـری از ثانیه حذف کنید.
نکته مهمی که باید درباره دســـتور Truncate Table بدانید این است که تاثیر استفاده از این دستور بر روی کلیه رکوردها بوده و

به هیچ عنوان نمی‌توان برای این دستور شرط (Where Clause)اعمال نمود.

شکل کلی استفاده از این دستور به صورت زیر می‌باشد.

TRUNCATE TABLE table_name

برای مثال اگر بخواهیم کلیه رکوردهای موجود در جدول Customers را حذف نمایید کافی است با استفاده از این دستور اینکار را انجام دهید

TRUNCATE TABLE Customers

با اجرای این دستور در کسری از ثانیه کلیه رکوردهای جدول Customers حذف خواهد شد. (بهتر است از این دستور زمانی استفاده کنید که بخواهید ساختار جدول شما باقی بماند)

اما در مورد دستور Truncate Table و Delete باید به نکات زیر توجه کنید.

۱- دستور Truncate Table فاقد قسمت شرط (Where Clause) می‌باشد در صورتیکه دستور Delete دارای قسمت شرط (Where Clause) است

۲- دستور Truncate Table در Log File آدرس Page و مقدار فضای آزاد شده (کمترین میزان Log) را می‌نویسد اما در صورتیکه دستور Delete در Log هر رکوردی را که قرار است حذف شود را در Log File ثبت می‌نماید.

۳- دستور Truncate Table باعث می‌شود که Pageهای متعلق به جدول deallocate شوند. deallocate شدن Page ها این معنی را می‌دهد که رکورد های موجود در جدول واقعاً حذف نشوند بلکه Extentهای مربوط به آن Page ها علامت Empty خورده تا دفعات بعد مورد استفاده قرار گیرند اما دستور Delete به طور فیزیکی محتوای Pageها مربوط به جدول را خالی می‌کند.

نکته : پس از Truncate شدن رکوردها امکان بازگشت آنها وجود ندارد.

۴- در صورتیکه جدول شما دارای ایندکس باشد. دستور Truncate Table آزاد کردن فضای مربوط به ایندکس را در یک مرحله انجام می‌دهد(مطابق بند ۳) همچنین Log مربوط به این حذف به شکل حداقل (مطابق بند ۲) در Log File ثبت می‌شود. اما دستور Delete هر رکوردی را که از ایندکس حذف می‌کند در Log File ثبت می‌کند.

۵- Trigger مربوط به دستور Delete به هیچ عنوان هنگام اجرای دستور Truncate Table فعال نمی‌شود. در صورتیکه با اجرای دستور Delete تریگر آن فعال خواهد شد.

۶- در صورتیکه جدول شما دارایReference (Relation) باشد امکان استفاده از دستور Truncate Table وجود ندارد. لازم به ذکر است حتی اگر Reference را غیر فعال کنید باز هم امکان استفاده از دستور Truncate Table وجود نخواهد داشت و تلاش برای اجرای دستور Truncate Table باعث نمایش خطای زیر خواهد شد.

```
TRUNCATE TABLE CUSTOMERS
GO
100 %
Messages
Msg 4712, Level 16, State 1, Line 1
Cannot truncate table 'Customers' because it is being referenced by a FOREIGN KEY constraint.
```

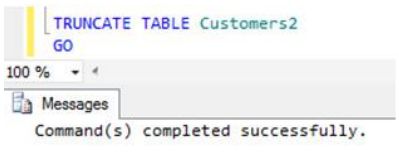
در صورتیکه در دستور Delete امکان حذف رکوردها به ازای جداولی که دارای Relation هستند وجود دارد. فقط باید به این نکته توجه کنید که ترتیب حذف رکوردها از جداول Master و Detail را رعایت کنید.

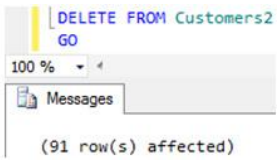
۷- دستور Truncate Table مقدار Identity را Reset کرده و آن را به Seed (هسته/مقدار اولیه) بر می‌گرداند. در صورتیکه دستور Delete تاثیری بر روی مقدار Identity ندارد

۸ - دستور Truncate Table تنها توسط کاربرانی قابل اجرا است که نقش DB_Owner و یا SysAdmin را داشته باشند در صورتیکه دستور Delete توسط هر کاربری که مجوز Delete بر روی جدول را داشته باشد قابل اجرا می‌باشد.

۹ - پس از اجرای دستور Truncate Table تعداد رکوردهای حذف شده نمایش داده نمی‌شود. در صورتیکه هنگام اجرای

دستور Delete تعداد رکوردهای حذف شده نمایش داده می‌شود.





Change Data Capture

یا CDC چیست ؟

یکی از قابلیت های ویژه SQL Server Enterprise Edition (که البته درنسخه های Developer و editions هم به خوبی کار می کند) اســت که روی جدولی در دیتابیس ما فعال میشــود و تغییرات ناشی از دستورات (Insert,Update,Delete DML) آن را نگهداری می کند و ما میتوانیم از این اطلاعات ا ستفاده کنیم. وقتی که ما cdc را روی جدولی فعال میکنیم، SQL Server جدولی مشــابه آن به همراه یکســری Metadata ایجاد می کند وتغییرات را در آن ذخیره می نماید. در کنار آن تعدادی Table-valued function در اختیار ما قرار میگیرند که به ما امکان استفاده از داده های ذخیره شده را می دهند.

CDC چگونه کار میکند؟

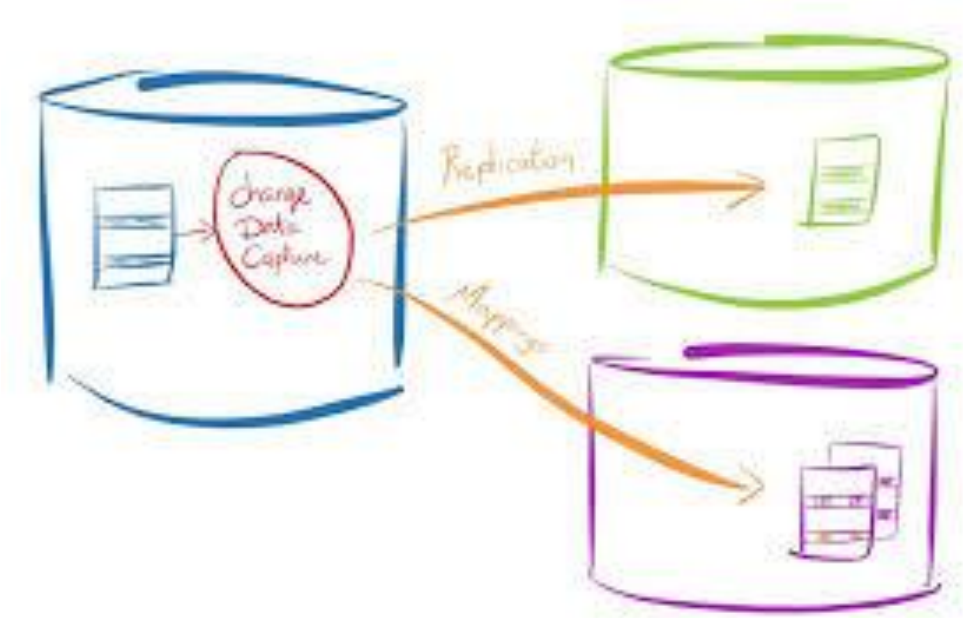
هنگامی که CDC روی یک جدول فعال می شود، SQL Server از مکانیزم نامتقارنی (Async) اســتفاده می کندکه به کمک آن تغییرات رخ داده در جداول را از فایل log می خواند و درجداولی که به منظور نگهداری تغییرات ایجاد کرده ذخیره میکند. در فایلهای log هر رکوردی که ذخیره می شود یک شناسه یکتا با نام Log Sequence Number دارد که به اختصــار LSN نامیده میشود. بسیاری از Metadataهایی که برای ما نگهداری می شود به خاطر قرابت نزدیکی که این تکنولوژی با Log دارد از همین اطلاعات Log استخراج می شوند.

برای فعال ســازی CDC روی یک جدول ابتدا به کمک دســتور sys.sp_cdc_enable_db باید آنرا در پایگاه داده مورد نظر فعال کنیم. با اجرای این د ستور سیستم CDC بر روی پایگاه داده ما فعال میشــود و زیرســاخت های مورد نیاز آن مانند meta-data table ها، CDC schema ، CDC database user و ... روی پایگاه داده ما ایجادمی شوند. در صورتی که نیاز دا شته با شید مــی تــوانــیــد در sys.databases و بــه کــمــک ســــتــون is_cdc_enabled بررسی کنید که آیا CDC برای دیتابیس شما

VS

فعال است یا خیر.
حالا شما میتوانید ازدستور sys.sp_cdc_enable_table برای فعال کردن CDC روی جدول خود استفاده کنید. با بررسی ستون is_tracked_by_cdc در sys.tables می توانید از فعال بودن یا نبودن CDC روی جدول خود مطلع شوید.

با فعال شدن این امکان، با هر تغییری در داده های جدول موردنظر شما، تغییرات آن در جدول CDC متناظر از Log استخراج شده و ذخیره می گردد. به صورت پیش فرض تمامی ستون های جدول نگهداری می شوند اما در صورت نیاز مثلا به دلایل امنیتی یا پرفورمنسی شمامی توانید تنها اطلاعات بعضی از ستون های جدول را نگهداری کنید. این کار را به کمک پارامتر @captured_column_list هنگام ایجاد CDC روی جدول انجام می دهیم. همچنین به صورت پیش فرض این اطلاعات روی FileGroup اصلی ما نگهداری می شوند که باز هم در صورت نیاز می توانید با استفاده از پارامتر @filegroup_name یک FileGroup دیگر را جهت نگهداری اطلاعات معرفی کنید.



فعال بودن SQL Server Agent برای فعال کردن این امکان اجباری نیست. اما فعال بودن آن برای کارکرد صحیح CDC لازم است. به همین منظور در صورتی که هنگام فعال کردن این امکان روی جدول اگر این SQL Server Agent استارت نباشد، با اینکه کار فعال سازی درست انجام می شود اما به شما پیامی مبنی بر عدم کارکرد صحیح سیستم داده میشود.

آیا جدول CDC بسیار بزرگ میشود؟

با توجه به اینکه هر تغییراتی که در جداول ما اتفاق می افتد در جدول CDC نگهداری می شود، احتمالا باید این سوال برای شما پیش آمده باشد که آیا این جدول بسیار بزرگ نمیشود؟! پاسخ این است خیر.

یک روال سیستمی وجود دارد که به صورت اتوماتیک هر ۳ روزیکبار داده ها قدیمی را پاک می کند. البته میتوانید این اندازه را به هر اندازه ای که نیاز دارید تغییر دهید و یا میتوانید به کمک روال ذخیره شده sys.sp_cdc_cleanup_change_table این کار را به صورت دستی انجام دهید.

در صورت تغییر در جدولی که تغییرات آن را نگهداری می کنیم چه اتفاقی می افتد؟

با تغییر در ساختار جدول روال CDC به کار خود ادامه می دهد، با این تفاوت که در صورتی که ستونی اضافه شود، دیگر تغییرات آن نگهداری نمی شود، و اگر ستونی هم حذف شود مقدار NULL برای آن نگهداری می شود. یعنی CDC توجهی به تغییرات اتفاق افتاده ندارد و کار خود را به همان شکل سابق ادامه می دهد. در صورت نیاز شما می توانید یک روال دیگر برای نگهداری تغییرات ستون های جدید ایجاد کنید. اما به خاطر داشته باشید که تنها ۲ نمونه از نگهداری سابقه را می توانید برای یک جدول فعال کنیم.