عنوان: **NOSQL قسمت اول**
نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۴ ۱۷:۴۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, Database, پایگاه‌داده, نوسی‌کوال, نواس‌کیوال

در این پست نگاهی کلی به ویژگی‌های پایگاه‌های داده NOSql خواهیم داشت و با بررسی تاریخچه و دلیل پیدایش این سیستم‌ها آشنا خواهیم شد.
با فراگیر شدن اینترنت در سال‌های اخیر و افزایش کاربران ، سیستم‌های RDBMS جوابگوی نیازهای برنامه‌نویسان در حوزه‌ی وب نبودند زیرا نیاز به نگهداری داده‌ها با حجم بالا و سرعت خواندن و نوشتن بالا از جمله نقط ضعف سیستم‌های RDBMS میباشد ، چرا که با افزایش شدید کاربران داده‌ها اصولا به صورت منطقی ساختار یکدست خود را جهت نگه‌داری از دست می‌دهند و به این ترتیب عملیات نرمال سازی منجر به ساخت جداول زیادی می‌شود که نتیجه آن برای هر کوئری عملیات Joinهای متعدد می‌باشد که سرعت خواندن و نوشتن را به خصوص برای برنامه‌های با گستره‌ی وب پایین می‌آورد و مشکلات دیگری در سیستم‌های RDBMS که ویژگی‌های سیستم‌های NoSql مشخص کننده آن مشکلات است که در ادامه به آن می‌پردازیم.
طبق تعریف کلی پایگاه داده NOSql عبارت است از:
نسل بعدی پایگاه داده (نسل از بعد RDBMS) که اصولا دارای چند ویژگی زیر باشد:
-۱ داده‌ها در این سیستم به صورت رابطه‌ای (جدولی) نمی‌باشند
-۲داده‌ها به صورت توزیع شده نگهداری می‌شوند.
-۳سیستم نرم‌افزاری متن باز می‌باشد.
-۴پایگاه داده مقیاس پذیر به صورت افقی می‌باشد(در مطالب بعدی توضیح داده خواهد شد.)

همان‌گونه که گفته شد این نوع پایگاه داده به منظور رفع نیازهای برنامه‌های با حجم ورود و خروج داده بسیار بالا (برنامه‌های مدرن وب فعلی) ایجاد شدند.
شروع کار پیاده‌سازی این سیستم‌ها در اوایل سال ۲۰۰۹ شکل گرفت و با سرعت زیادی رشد کرد و همچنین ویژگی‌های کلی دیگری نیز به این نوع سیستم اضافه شد.
که این ویژگی‌ها عبارتند از:
Schema-free : بدون شَما ! ، با توجه به برنامه‌های وبی فعلی ممکن است شمای نگه‌داری داده‌ها (ساختار کلی) مرتبا و یا گهگاهی تغییر کند. لذا در این سیستم‌ها اصولا داده‌ها بدون شمای اولیه طراحی و ذخیره می‌شوند. (به عنوان مثال می‌توان در یک سیستم که مشخصات کاربران وارد سیستم می‌شود برای یک کاربر یک سری اطلاعات اضافی و برای کاربری دیگر از ورود اطلاعات اضافی صرف‌نظر کرد ، و در مقایسه با RDBMS به این ترتیب از ورود مقادیر Null و یا پیوندهای بیمورد جلوگیری کرد.
کنترل اطلاعات الزامی توسط لایه سرویس برنامه انجام می‌شود. (در زبان جاوا توسط jsr-303 و یاBean Validation ها)
easy replication support : در این سیستم ، نحوه‌ی گرفتن نسخه‌های پشتیبان و sync بودن نسخه‌های مختلف بسیار ساده و سر راست می‌باشد و سرور پایگاه داده به محض عدم توانایی خواندن و یا نوشتن از روی دیسک سراغ نسخه‌ی پشتیبان می‌رود و آن نسخه را به عنوان نسخه‌ی اصلی در نظر می‌گیرد.
Simple API : به دلیل متن‌باز بودن و فعال بودن Community این سیستم‌ها APIهای ساده و بهینه‌ای برای اکثر زبان‌های برنامه‌نویس محبوب ایجاد شده است که در پست‌های بعدی با ارائه مثال آنها را بررسی خواهیم کرد.
eventually consistent : در سیستم‌های RDBMS که داده‌ها خاصیت ACID را (در قالب Transaction) پیاده می‌کنند ، در این سیستم‌های داده‌ها در وضعیت BASE قرار دارند که سرنام کلمات Eventual Consistency ، Soft State ، Basically Available می‌باشد.
huge amount of data : این سیستم‌ها به منظور کار با داده‌های با حجم بالا ایجاد شده‌اند ، یک تعریف کلی می‌گوید اگر مقدار داده‌های نگهداری شده در پایگاه‌های داده برنامه شما ظرفیتی کمتر از یک ترابایت داده دارد از پایگاه داده RDBMS استفاده کنید واگر ظرفیت آن از واحد ترابایت فراتر می‌رود از سیستم‌های NOSql استفاده کنید.
به طور کلی پایگاه داده‌ای که در چارچوب موارد ذکر شده قرار گیرد را می‌توان از نوع NoSql که سرنام کلمه (Not Only SQL) می‌باشد قرار داد. تاکنون پیاده‌سازی‌های زیادی از این سیستم‌ها ایجاد شده است که رفتار و نحوه‌ی نگه‌داری داده‌ها (پرس‌وجو ها) در این سیستم‌ها با یکدیگر متفاوت می‌باشد.

جهت پیاده سازی پایگاه داده با این سیستم‌ها تا حدودی نگرش کلی به داده‌ها و نحوه‌ی چیدمان آنها تغییر می‌کند ، به صورت کلی

مباحث مربوط به normalization و de-normalization و تصور داده‌ها به صورت جدولی کنار می‌رود.
سیستم NoSql به جهت دسته‌بندی نحوه‌ی ذخیره‌سازی داده‌ها و ارتباط بین آنها به ۴ دسته کلی تقسیم می‌شود که معرفی کلی آن دسته‌بندی‌ها موضوع مطلب بعدی می‌باشد.

عنوان: **NOSQL قسمت دوم**
نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۶ ۹:۲۵
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, Database, پایگاه‌داده, نوسی‌کوال, نواس‌کیوال, key-value, کلید-مقدار

در مطلب قبلی با تعاریف سیستم‌های NoSQL آشنا شدیم و به طور کلی ویژگی‌های یک سیستم NoSQL را بررسی کردیم.

در این مطلب دسته‌بندی کلی و نوع ساختار داده‌ای این سیستم‌ها و بررسی ساده‌ترین آنها را مرور می‌کنیم.

در حالت کلی پایگاهای داده NoSQL به ۴ دسته تقسیم می‌شوند که به ترتیب پیچیدگی ذخیره‌سازی داده‌ها عبارتند از:
Key/Value Store Databases

Document Databases
Graph Databases
Column Family Databases
در حالت کلی در پایگاه‌های‌داده NoSQL داده‌ها در قالب KEY/VALUE (کلید/مقدار) نگه‌داری می‌شوند ، به این صورت که مقادیر توسط کلید یکتایی نگاشت شده و ذخیره می‌شوند، هر مقدار صرفا توسط همان کلید نگاشت شده قابل بازگردانی می‌باشد و راهی جهت دریافت مقدار بدون دانستن کلید وجود ندارد . در این ساختارداده منظور از مقادیر، داده‌های اصلی برنامه هستند که نیاز به نگه‌داری دارند و کلیدها نیز رشته‌هایی هستند که توسط برنامه‌نویس ایجاد می‌شوند.
به دلیل موجود بودن این نوع ساختار داده‌ای در اکثر کتابخانه‌های زبان‌های برنامه‌نویسی (به عنوان مثال پیاده‌سازی‌های مختلف اینترفیس Map شامل HashMap ، HashTable و موارد دیگر در کتابخانه‌های JDK) این نوع ساختار برای اکثر برنامه‌نویسان آشنا بوده و فراگیری آن نیز ساده می‌باشد.

بدیهی است که اعمال فرهنگ داده‌ای (درج ، حذف ، جستجو) در این سیستم به دلیل اینکه داده‌ها به صورت کلید/مقدار ذخیره می‌شوند دارای پیچیدگی زمانی O(1) می‌باشد که بهینه‌ترین حالت ممکن به لحاظ طراحی می‌باشد. همان‌گونه که مستحضرید در الگوریتم‌هایی که دارای پیچیدگی زمانی با مقدار ثابت دارند کم یا زیاد بودن داده‌ها تاثیری در کارایی الگوریتم نداشته و همواره با هر حجم داده‌ای زمان ثابتی جهت پردازش نیاز می‌باشد.

:Key/Value Store Databases

این سیستم ساده‌ترین حالت از دسته‌بندی‌های NoSQL می‌باشد ، به طور کلی جهت استفاده در سیستم‌هایی است که داده‌ها متمایز از یکدیگر هستند و اصولا Availability و یا در دسترس بودن داده‌ها نسبت به سایر موارد نظیر پایائی اهمیت بالاتری دارد.

از موارد استفاده این گونه سیستم‌ها به موارد زیر می‌توان اشاره کرد:
در پلتفرم‌های اشتراک گذاری داده‌ها . هدف کلی صرفا هندل کردن آپلود محتوی (باینری) و به صورت همزمان بروز کردن در سمت دیگر می‌باشد.(اپلیکیشنی مانند اینستاگرام را تصور کنید) در اینگونه نرم‌افزارها با تعداد بسیار زیاد کاربر و تقاضا، استفاده از این نوع پایگاه داده به مراتب کارایی و سرعت را بالاتر می‌برد. و با توجه به عدم پیش‌بینی حجم داده‌ها یکی از ویژگی‌های این نوع پایگاه داده تحت عنوان Horizontal Scaling مطرح می‌شود که در صورت Overflow شدن سرور، داده‌ها را به سمت سرور دیگری می‌توان هدایت کرد وبدون مشکل پردازش را ادامه داد ، این ویژگی یک وجه تمایز کارایی این سیستم با سیستم‌های RDBMS می‌باشد که جهت مقابله با چنین وضعیتی تنها راه پیش‌رو بالا بردن امکانات سرور می‌باشد و به طور کلی داده‌ها را در یک سرور می‌توان نگه‌داری کرد (البته راه‌حل‌هایی نظیر پارتیشن کردن و غیره وجود دارد که به مراتب پیچیدگی و کارایی کمتری نسبت به Horizontal Scaling در پایگاه‌های داده NoSQL دارد.)

برای Cache کردن صفحات بسیار کارا می‌باشد ، به عنوان مثال می‌توان آدرس درخواست را به عنوان Key در نظر گرفت و مقدار آن را نیز معادل JSON نتیجه که توسط کلاینت پردازش خواهد شد قرار داد.

یک نسخه کپی شده از توئیتر که کاملا توسط این نوع پایگاه داده پیاده شده است نیز از این آدرس قابل مشاهده است. این برنامه به زبان‌های ruby , php و java نوشته شده است و سورس نیز در مخارن github می‌جود می‌باشد. (یک نمونه پیاده سازی ایده‌آل جهت آشنایی با نحوه‌ی مدیریت داده‌ها در این نوع پایگاه داده)

از پیاده‌سازی‌های این نوع پایگاه داده به موارد زیر می‌توان اشاره کرد:

Amazon SimpleDB

Memcached

Oracle Key/value Pair

Redis

هر یک از پیاده‌سازی‌ها دارای ویژگی‌های مربوط به خود هستند به عنوان مثال Memcached داده‌ها را صرفا در DRAM ذخیره می‌کند که نتیجه‌ی آن Volatile بودن داده‌ها می‌باشد و به هیچ وجه از این سیستم جهت نگهداری دائمی داده‌ها نباید استفاده شود. از طرف دیگر Redis داده‌ها را علاوه بر حافظه اصلی در حافظه جانبی نیز ذخیره می‌کند که نتیجه‌ی آن سرعت بالا در کنار پایائی می‌باشد.

همان‌گونه که در تعریف کلی عنوان شد یکی از ویژگی‌های این سیستم‌ها متن‌باز بودن انها می‌باشد که نتیجه‌ی آن وجود پیاده‌سازی‌های متنوع از هر کدام می‌باشد ، لازم است قبل از انتخاب هر سیستم به خوبی با ویژگی‌های اکثر سیستم‌های محبوب و پراستفاده آشنا شویم و با توجه به نیاز سیستم را انتخاب کنیم.

در مطلب بعدی با نوع دوم یعنی Document Databases آشنا خواهیم شد.

نظرات خوانندگان

نویسنده: مجید هزاری
تاریخ: ۱۳۹۱/۱۱/۲۸ ۱۵:۴۷

عالی است.
متشکرم.

نویسنده: احمد ولی پور
تاریخ: ۱۳۹۱/۱۱/۲۸ ۱۷:۵۵

یه سوال برام پیش اومده:
با رایج شدن nosql پایگاه داده هایی مثل Oracle یا Sql Server چی میشن؟

نویسنده: مجید هزاری
تاریخ: ۱۳۹۱/۱۱/۲۸ ۱۹:۵۰

اینها تداخلی با یکدیگر ندارند.
NOSQL تنها برای رفع نیاز هایی ظهور کرده است که RelDB در آنها ضعیف بوده. همانطور که NoSQL در زمینه هایی که RelDB قوی است ضعیف عمل خواهد کرد.
(البته من کاملا مختصر گفتم)

نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۸ ۲۰:۱۷

در حالت کلی هرکدام از پایگاه‌داده‌ها بسته به نیاز استفاده می‌شن ، توی برنامه‌های اینترپرایز وبی مفهوم Polyglot Persistence مطرحه (که می‌شه اونو نگه‌داری یا ذخیره سازی چند زبانی ترجمه کرد) که می‌گه توی یک سیستم از چندین نوع پایگاه داده~~می‌شه~~ (باید) استفاده کرد. به عنوان مثال برای نگه‌داری داده‌هایی جهت گزارش گیری و یا ایجاد Transactionها بهترین گزینه همان سیستم‌های RDBMS هستند ، در مطالب آتی به این موضوع اشاره بیشتری خواهم کرد ، مارتین فویلر در این مطلب مفهوم Polyglot Persistence را به خوبی توضیح داده‌اند.

نویسنده: saremi
تاریخ: ۱۳۹۲/۱۱/۲۹ ۱۷:۲۲

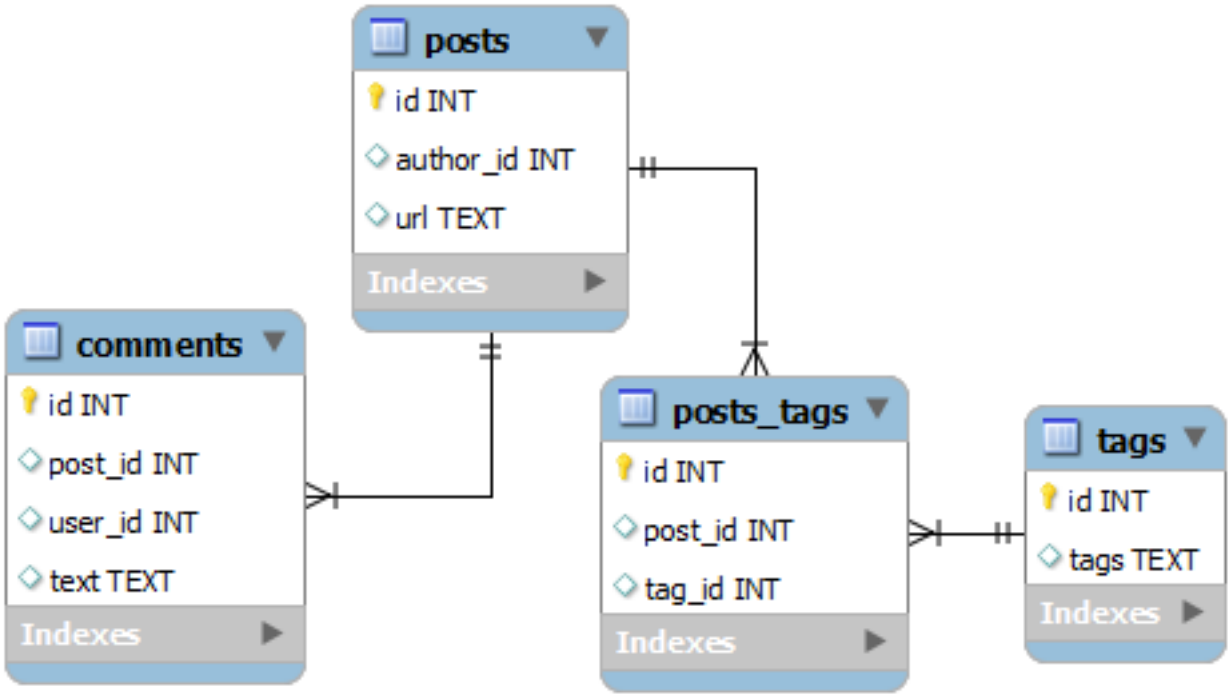
سلام
می خواستم بپرسم bucket در key value store دقیقا چیه؟
بعد خیلی از جاها راجع به hash table و hash code هم مطالبی گفته اند. آیا منظور فقط hash کردن کلید است یا فرآیند پیچیده‌تر از این حرفاست؟

عنوان: **NOSQL قسمت سوم**
نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۱۹:۱۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, Database, نوسی‌کوال, نواس‌کیوال, mongodb, پایگاه

در مطلب قبلی با نوع اول پایگاه‌های‌داده NoSQL یعنی Key/Value Store آشنا شدیم و در این مطلب به معرفی دسته دوم یعنی Document Database خواهیم پرداخت.

در این نوع پایگاه داده ، داده‌ها مانند نوع اول در قالب کلید/مقدار ذخیره می‌شوند و بازگردانی مقادیر نیز دقیقا مشابه نوع اول یعنی Key/Value Store بر اساس کلید م ی‌باشد. اما تفاوت این سیستم با نوع اول در دسته‌بندی داده‌های مرتبط با یکدیگر در قالب یک Document می‌باشد. سعی کردم در این مطلب با ذکر مثال مطالب را شفاف‌تر بیان کنم:

به عنوان مثال اگر بخواهیم جداول مربوط به پست‌های یک سیستم CMS را بصورت رابطه‌ای پیاده کنیم ، یکی از ساده‌ترین حالات پایه برای پست‌های این سیستم در حالت نرمال به صورت زیر می‌باشد.



جداول واضح بوده و نیازی به توضیح ندارد ، حال نحوه‌ی ذخیره‌سازی داده‌ها در سیستم Document Database برای چنین مثالی را بررسی می‌کنیم:

```
{
_id: ObjectID(‘4bf9e8e17cef4644108761bb’),
Title: ‘NoSQL Part3’,
url: ‘http://dotnettips.info/yyy/xxxx’,
author: ‘hamid samani’,
tags: [‘databases’,’mongoDB’],
comments:[
{user: ‘unknown user’,
 text:’unknown test’
},
{user:unknown user2’,
 text:’unknown text2
```

```
}
]
}
```

همانگونه که مشاهد می‌کنید نحوه‌ی ذخیره‌سازی داده‌ها بسیار با سیستم رابطه‌ای متفاوت می‌باشد ، با جمع‌بندی تفاوت نحوه‌ی نگه‌داری داده‌ها در این سیستم و RDBMS و بررسی این سیستم نکات اصلی به شرح زیر می‌باشند:

-۱فرمت ذخیره سازی داده‌ها مشابه فرمت JSON می‌باشد.

-۲به مجموعه داده‌های مرتبط به یکدیگر Document گفته می‌شود.

-۳در این سیستم JOIN ها وجود ندارند و داده‌های مرتبط کنار یکدیگر قرار می‌گیرند ، و یا به تعریف دقیق‌تر داده‌ها در یک داکیومنت اصلی Embed می‌شوند .

به عنوان مثال در اینجا مقدار comment ها برابر با آرایه‌ای از Document ها می‌باشد.

-۴مقادیر می‌توانند بصورت آرایه نیز در نظر گرفته شوند.

-۵در سیستم‌های RDBMS در صورتی که بخواهیم از وجود JOIN ها صرفنظر کنیم. به عدم توانایی در نرمال‌سازی برخواهیم خورد که یکی از معایب عدم نرمال‌سازی وجود مقادیر Null در جداول می‌باشد؛ اما در این سیستم به دلیل Schema free بودن می‌توان ساختارهای متفاوت برای Document ها در نظر گرفت.

به عنوان مثال برای یک پست می‌توان مقدار n کامنت تعریف کرد و برای پست دیگر هیچ کامنتی تعریف نکرد.

-۶در این سیستم اصولا نیازی به تعریف ساختار از قبل موجود نمی‌باشد و به محض اعلان دستور قرار دادن داده‌ها در پایگاه‌داده ساختار متناسب ایجاد می‌شود.

با مقایسه دستورات CRUD در هر دو نوع پایگاه داده با نحوه‌ی کوئری گرفتن از Document Database آشنا می‌شویم:

در SQL برای ایجاد جدول خواهیم داشت:

```
CREATE TABLE posts (
    id INT NOT NULL
        AUTO_INCREMENT,
    author_id INT NOT NULL,
    url VARCHAR(50),
    PRIMARY KEY (id)
)
```

دستور فوق در Document Database معادل است با:

```
db.posts.insert({id: “256” , author_id:”546”,url:"http://example.com/xxx"}) // با قرار دادن مقدار نوع ساختار مشخص می‌شود
```

در SQL جهت خواندن خواهیم داشت:

```
SELECT * from posts
WHERE author_id > 100
```

و معادل آن برابر است با:

```
db.posts.find({author_id:{$gt:”1000”}})
```

در SQL جهت بروزرسانی داریم:

```
UPDATE posts
SET author_id= "123"
```

که معادل است با:

```
db.posts.update({ $set: { author_id: "123" }})
```

در SQL جهت حذف خواهیم داشت:

```
DELETE FROM posts
WHERE author_id= "654"
```

که معادل است با:

```
db.posts.remove( { author_id: "654" } )
```

همانگونه که مشاهده می‌فرمایید نوشتن کوئری برای این پایگاه داده ساده بوده و زبان آن نیز بر پایه جاوا اسکریپت می‌باشد که برای اکثر برنامه‌نویسان قابل درک است.

تاکنون توسط شرکت‌های مختلف پیاده‌سازی‌های مختلفی از این سیستم انجام شده است که از مهم‌ترین و پر استفاده‌ترین آنها می‌توان به موارد زیر اشاره کرد:

MongoDB

CouchDB

RavenDB

نظرات خوانندگان

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۱۹:۵۷

با تشکر از مطلب زیباتون
یک سوال داشتم ایا این روش اونقدر به بلوغ رسیده که بشه در پروژه‌ها روش حساب کرد . یا اینکه فعلا از همون روش قبلی استفاده کنیم
سوال دیگر من هم این هست که به نظر شما در nosql اینده ایی دیده میشه ؟
با تشکر

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۱۹:۵۹

اگر هم امکان داره refrence ی در این زمینه هست link بدید

نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۲۱:۳

در رابطه با سوال اولتون عارضم که در حال حاضر همه‌ی شرکت‌های بزرگ و فعال در این صنعت مثل گوگل ، فیس ب و ک توئیتر و از این شیوه استفاده می‌کنند ، در حالت کلی این مبحث یک تکنولوژی خاص نیست که مصرفی باشه و بعد از مدتی تاریخش بگذره ، یک Movement و یا یک نگرش کلی در تعریف عامه از مجموعه‌ای از راه حل‌ها به منظور رفع مشکلاتRDBMS در پردازش داده‌های بزرگ (BigData) ، داده‌ها در حوزه‌ی وب هم که رشدی نمایی دارند.
در رابطه با سوال دوم هم بستگی به خود فرد و یا شرکت مربوطه داره ، در حوزه‌ی نرم‌افزارهای داخلی به دلیل پایین‌تر بودن حجم داده‌ها الزامی در استفاده از این روش‌ها نیست. (استفاده و یا عدم استفاده مستقیما به نوع نرم‌افزار و ساختار آن بستگی دارد)

نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۲۱:۵

از اینجا که شما شروع کنید به همه جا لینک می‌شوید :)

نویسنده: سعید یزدانی
تاریخ: ۱۳۹۱/۱۱/۲۹ ۲۱:۲۴

ممنون بابت جواب کاملتون

نویسنده: توحید عزیزی
تاریخ: ۱۳۹۱/۱۱/۳۰ ۳:۵۲

سلام

سپاسگزارم از موضوع جالبی که انتخاب کرده اید و مطالب خوبی که می‌نویسید.
آیا امکان دارد که در مورد هر کدام از انواع دیتابیس نوسیکوئل، مثالهای بیشتری بزنید.
از یک سرویس رایگان برای نوشتن مثال‌ها می‌تواند استفاده کرد که برای همه در دسترس باشد، مثل: cloudant.com
با تشکر

نویسنده: Meysam Navaei
تاریخ: ۱۳۹۱/۱۱/۳۰ ۹:۱۶

سلام

توحید این سایت که معرفی کردی خیلی جالب بود.می خاستم بینم محدودیت حجمی در استفاده ازش وجود داره یا نه نامحدود. ریسک محسوب نمیشه‌ی پروژه بزرگ داشته باشی و بخای دیتابیس رو از سرویس این سایت استفاده کنی؟منظورم اینکه از این سایت‌ها نباشه که یهو محدودیت ایجاد بکنه و یا پولی بشه و...

نویسنده: حمید سامانی
تاریخ: ۱۳۹۱/۱۱/۳۰ ۱۶:۱۵

سلام
سعی می‌کنم مثال‌های بیشتری را در مطالب آتی بگنجانم.
با سپاس :)

نویسنده: توحید عزیزی
تاریخ: ۱۳۹۱/۱۲/۰۳ ۹:۳

سلام.
نسخه رایگانش محدودیت داره: فکر کنم 2000 کوئری در روز.
اگر می‌خواهید پروژه‌ی بزرگ روش ببرید، باید از نسخه‌های تجاریش استفاده کنید. البته من خودم تستش نکرده ام هنوز.
https://cloudant.com/#home-pricing

نویسنده: masi
تاریخ: ۱۳۹۲/۰۲/۱۹ ۱:۵۶

سلام ، واقعا مطالب خوبی بود هر جا رو گشتم کاملتر و جامع‌تر از همه بودید ، موضوع پروژه‌ی من روی این موضوعه ، ای کاش میشد در مورد موضوع زیر صحبت کنید. key value store

نویسنده: جواد زبیدی
تاریخ: ۱۳۹۲/۰۵/۱۶ ۳:۳۳

سلام تشکر از مطلب بسیار مفیدتون .
می خواستم بدونم که کدام یک از روش‌ها بیشتر امتحان خودش رو توی داده‌های زیاد پس داده .و بشه راحتر باهاش کار کرد .
روش
Document store
Key value
روش هایی دیگری رو هم توی سایت دیدم اگر امکان داره مزایا و معایب هر کدوم رو توضیح دهید ممنون.

نویسنده: دادخواه
تاریخ: ۱۳۹۲/۰۶/۰۷ ۱۲:۱۰

سلام
تشکر از مطالب خوبتون
اما چند تا سوال دارم.
1- از این سه تا پایگاه داده که در اخر نوشتید فکر کنم فقط MongoDB مجانی باشه. درسته؟
2- آیا دستورات در همه این پایگاه داده‌ها به همین صورت است؟
3- آیا همه سرورها و هاست‌ها از این پایگاه داده‌ها مانند MS SQL پشتیبانی می‌کنند و یا سرورهای خاص را باید پیدا کرد؟
تشکر

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۰۶/۰۷ ۱۲:۲۵

اگر مطالب مقدماتی‌تر رو مطالعه می‌کردید، می‌دید که اصلا هدف از بانک اطلاعاتی NoSQL این نیست که باهاش سایت معمولی درست کنند اون هم روی سرور اجاره‌ای با 100 مگ فضا. هدفش توزیع شده بودن در سرورهایی متعدد و یا با پراکندگی جغرافیایی بالا است.

نتیجه گیری؟ ابزار زده نباشید. اول مفاهیم رو مطالعه کنید. اول تئوری کار مهمه.

نویسنده: saremi
تاریخ: ۱۳۹۲/۱۱/۲۴ ۱۶:۴۴

با سلام؛ میخواستم در مورد ،HQL، CQL ،UNQL، map reduce و... بپرسم. توی همون سایتی که لینکش رو دادین این‌ها جزو انواع query methodها هستند. من دقیق نمی‌فهمم الان ما دستورات مشابه sql رو معادلش رو با java script نوشتیم. در مورد تفاوت اینها و استفاده شون اگر میشه کمی توضیح بدین لطفا. دقیقا توی انواع مختلف پایگاه داده با چه زبانی کوئری نویسی می‌شه؟ با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۱۱/۲۴ ۱۶:۵۲

در مورد تفاوت این‌ها در مطلب مروری بر مفاهیم مقدماتی NoSQL بیشتر توضیح داده شده. برچسب NoSQL را بهتر است دنبال کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۱/۲۴ ۱۷:۴۹

در مورد MongoDB یک کتابچه‌ی فارسی 90 صفحه‌ای موجود است .

نویسنده: salam
تاریخ: ۱۳۹۳/۰۵/۰۹ ۱۱:۱۶

سلام
در قسمت دوم این مطلب اومده که "در حالت کلی پایگاهای داده NoSQL به ۴ دسته تقسیم می‌شوند " دسته 3 و 4 را توضیح نمی‌دین؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۵/۰۹ ۱۱:۲۶

برای دنبال کردن مطالب هم خانواده در این سایت، در ذیل هر مطلب یک سری گروه یا برچسب تعریف شده‌اند. برای مثال اگر برچسب NoSQL را دنبال کنید، در مطالب دیگری پاسخ خود را خواهید یافت.

عنوان: **اصول پایگاه داده - اندیس ها (indices)**
نویسنده: حامد خسروجردی
تاریخ: ۱۳۹۲/۰۸/۰۳ ۰:۵
آدرس: www.dotnettips.info
برچسب‌ها: Index, Database

با افزایش حجم بانک‌های اطلاعاتی دسترسی سریع به داده‌های مطلوب به یک معضل تبدیل می‌شود. بهمین دلیل نیاز به مکانیزم هایی برای بازیابی سریع داده‌ها احساس می‌شود. یکی از این مکانیزم‌ها اندیس گذاری (indexing) است. اندیس گذاری مکانیزمی است که به ما امکان دسترسی مستقیم (direct access) را به داده‌های بانک اطلاعاتی می‌دهد.

عمل اندیس گذاری وظیفه طراح بانک اطلاعاتی است که با توجه به دسترسی هایی که در آینده به بانک اطلاعاتی وجود دارد مشخص می‌کند که بر روی چه ستون هایی می‌خواهد اندیس داشته باشد. بعنوان مثال با تعیین کلید اصلی اعلام می‌کند که بیشتر دسترسی‌های آینده من بر اساس این کلید اصلی است و بنابراین بانک اطلاعاتی بر روی کلید اصلی اندیس گذاری را انجام می‌دهد. علاوه بر کلید اصلی می‌توان بر روی هر ستون دیگری از جدول نیز اندیس گذاشت که همانطور که گفته شد این مسئله بستگی به تعداد دسترسی آینده ما از طریق آن ستون‌ها دارد.

پس از اندیس گذاری بر روی یک ستون بسته به نوع اندیس فایلی در پایگاه اطلاعاتی ما ایجاد می‌شود که به آن فایل اندیس (index file) گفته می‌شود. این فایل یک فایل مبتنی بر رکورد (record-based) است که هر رکورد آن محتوی زوج *کلید جستجو* - *اشاره گر* می باشد. کلید جستجو را مقدار ستون مورد نظر و اشاره گر را اشاره گری به رکورد مربوط به ان می‌تواند در نظر گرفت.

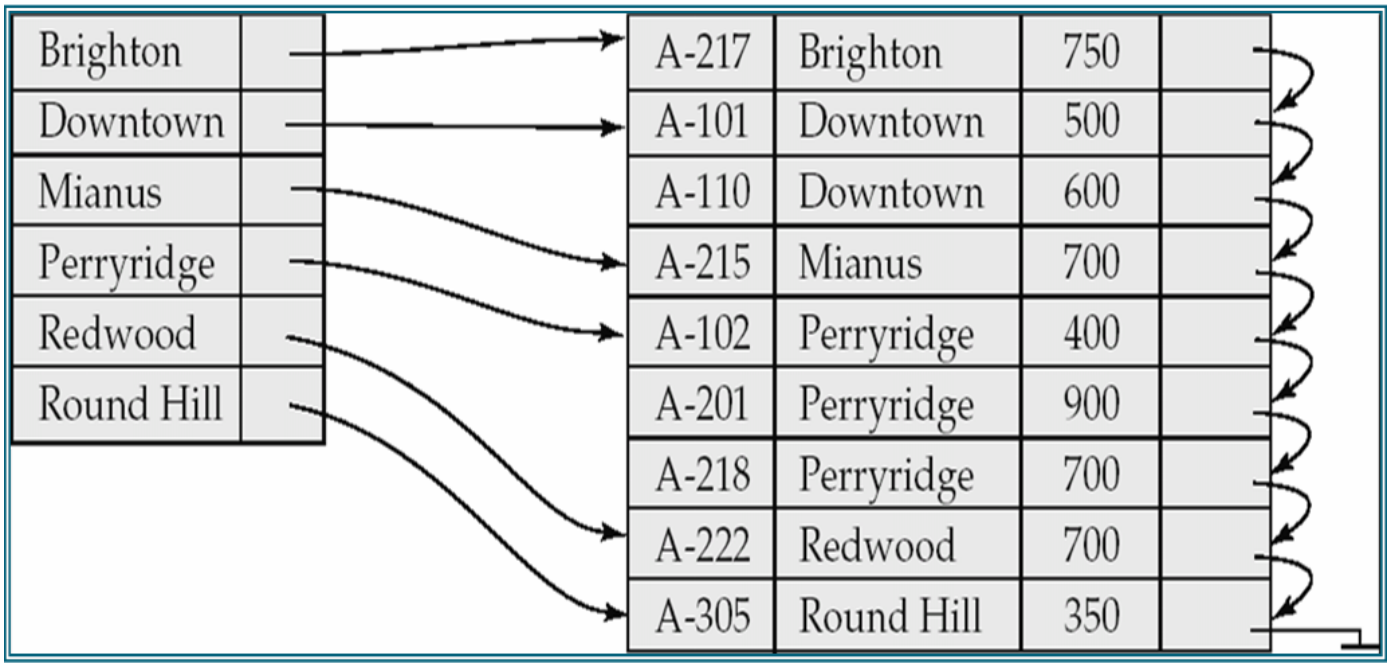
توجه داشته باشید که اندیس گذاری و مدیریت اندیس ها، همانطور که در این مقاله آموزشی گفته خواهد شد سر بار هایی (از نظر حافظه و پردازش) را بر سیستم تحمیل می‌نمایند. بعنوان مثال با اندیس گذاری بر روی هر ستونی یک فایل اندیس نیز ایجاد می‌شود بنابراین اگر اندیس‌های ما بسیار زیاد باشد حجم زیادی از بانک اطلاعاتی ما را خواهند گرفت. مدیریت و بروز نگهداری فایل‌های اندیس نیز خود مسئله ایست که سربار پردازشی را بدنبال دارد. بنابراین توصیه می‌شود در هنگام اندیس گذاری حتما بررسی‌ها و تحلیل‌های لازم را انجام دهید و تنها بر روی ستون هایی اندیس بگذرید که در آینده بیشتر دسترسی‌های شما از طریق ان ستون‌ها خواهد بود.

عموما در بانک‌های اطلاعاتی دو نوع اندیس می‌تواند بکار گیری شود که عبارتند از :
اندیس‌های مرتب (ordered indices) : در این نوع کلیدهای جستجو (search-key) بصورت مرتب نگداری می‌شوند.
اندیس‌های هش (Hash indices) : در این نوع از اندیس‌ها کلیدهای جستجو در فایل اندیس مرتب نیستند. بلکه توسط یک تابع هش (hash function) توزیع می‌شوند.

در این مقاله قصد داریم به اندیس‌های مرتب بپردازیم و بخشی از مفاهیم مطرح در این باره را پوشش دهیم.

اندیس‌های متراکم (dense index):

اولین و ساده‌ترین نوع از اندیس‌های مرتب **اندیس‌های متراکم (dense)** هستند. در این نوع از اندیس‌ها وقتی بر روی ستونی می‌خواهیم عمل اندیس گذاری را انجام دهیم می‌بایست به ازای هر کلید - جست و جو (search-key) غیر تکراری در ستون مورد نظر، یک رکورد در فایل اندیس مربوط به ان ستون اضافه کنیم. برای روشن شدن بیشتر موضوع به شکل زیر توجه کنید.



شکل 1 – اندیس متراکم (sparse index)

همانطور که در تصوری مشاهده می‌کنید بر روی ستون دوم از این جدول (جدول سمت راست)، اندیس متراکم (dense) گذاشته شده است. بر همین اساس به ازای هر کدام از اسامی خیابان‌ها یک رکورد در فایل اندیس (جدول سمت چپ) آورده شده است. در فایل اندیس می‌بینید که در کنار کلید جستجو یک اشاره گر نیز به جدول اصلی وجود دارد که در هنگام دسترسی مستقیم (direct access) از این اشاره گر استفاده خواهد شد. دقت کنید که کلیدهای جستجو در فایل اندیس بصورت مرتب نگهداری شده اند که نکته ای کلیدی در اندیس‌های مرتب می‌باشد.

مرتب بودن فایل اندیس موجب می‌شود که ما در هنگام جستجوی کلید مورد نظرمان در جدول اندیس بتوانیم از روش‌های جستجویی نظری جست و جوی دو دویی استفاده کنیم و در نتیجه سریع‌تر کلید مورد نظر را پیدا کنیم. این مسئله باعث بهبود کارایی می‌شود. بعنوان مثال فرض کنید در فایل اندیس یک ملیون رکورد داریم. در این صورت برای یافتن کلید مورد نظرمان در جدول اندیس بروش جست و جوی دو دویی تنها کافی است 20 عمل مقایسه انجام دهیم. بنابراین می‌بینید که مرتب نگهداشتن جدول اندیس چقدر در سرعت بازیابی، تاثیر دارد.

نکته مهمی که در اندیس‌های متراکم باید به آن دقت شود اینست که ما به ازای کلیدهای جستجوی غیر تکراری یک رکورد در جدول اندیس نگهداری می‌کنیم. برای مثال در شکل بالا در ستون مورد نظر ما دو رکورد برای Downtown و سه رکورد برای Perryridge وجود دارد. این در حالی است که در فایل اندیس فقط یک Downtown و Perryridge داریم.

در اندیس‌های متراکم ما امکان دو نوع دسترسی را داریم :
دسترسی مستقیم (direct access)
دسترسی ترتیبی (sequential access)

دسترسی مستقیم :

توجه داشته باشید که در هنگام کار با یک جدول، فایل‌های اندیس آن به حافظه اصلی آورده می‌شوند (البته ممکن است که بخشی از فایل‌های اندیس به حافظه اصلی نیایند). این در حالی است که فایل اصلی جدول در حافظه جانبی قرار دارد. بنابراین در هنگام

بازیابی یک رکورد از برای یافتن محل ان رکورد نیازی به مراجعه زیاد به حافظه جانبی نیست. بلکه در حافظه اصلی بسرعت با یک عمل جستجو اشاره گر مربوط به رکورد مورد نظر در حافظه جانبی پیدا شده و مستقیما به آدرس همان رکورد می‌رویم و آن را می‌خوانیم. به این دسترسی، دسترسی مستقیم (direct access) می گوییم.

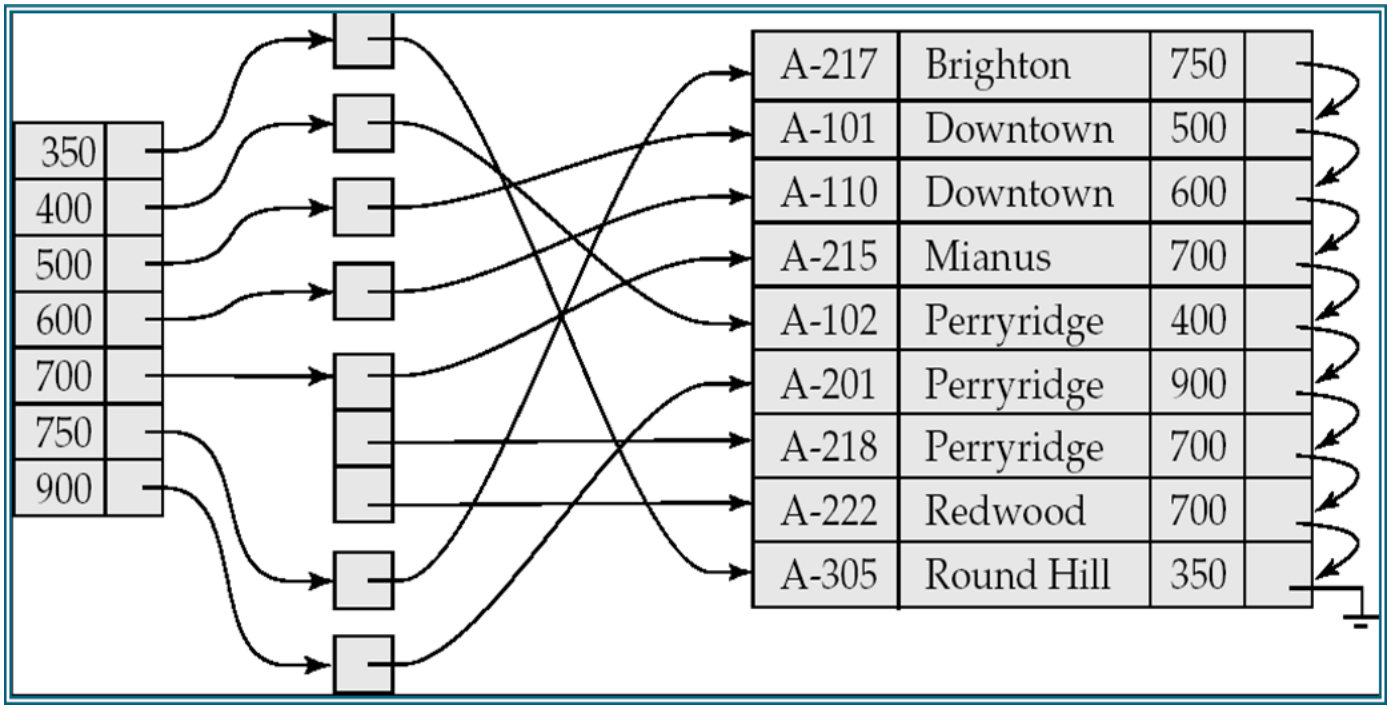
دسترسی ترتیبی :

در برخی از روش‌های اندیس گذاری علاوه بر دسترسی مستقیم امکان دسترسی بصورت ترتیبی نیز وجود دارد. در دسترسی ترتیبی این امکان وجود دارد که از یک رکورد خاص در جدول اصلی بتوانیم رکوردهای بعد از آن را به ترتیبی منطقی پیمایش کنیم. برای روشن‌تر شدن موضوع به شکل شماره 1 توجه کنید. در انتهای هر رکورد اشاره گری به رکورد منطقی بعدی مشاهده می‌کنید. این اشاره گرها امکان پیمایش و دسترسی ترتیبی را به ما می‌دهند. بعنوان مثال فرض کنید قصد داریم تمامی رکوردهای حاوی کلید Perryridge را بازیابی نماییم. از آنجایی که در جدول اندیس تنها برای یکی از رکوردهای حاوی این کلید اندیس داریم، برای بازیابی باقی رکوردها چه باید کرد؟ در چنین شرایطی ابتدا با دسترسی مستقیم اولین رکورد حاوی Perryridge را پیدا کرده و آن را بازیابی می‌کنیم. سپس از طریق اشاره گر انتهای آن رکورد، می‌توان به رکورد بعدی آن دست یافت و به همین ترتیب می‌توان یک به یک به رکوردهای دیگر دسترسی ترتیبی پیدا نمود.

دقت کنید که رکوردهای جدول ما بصورت فیزیکی مرتب نیستند. اما اشاره گرهای انتهای رکوردها طوری مقدار دهی شده اند که بتوان آنها را بصورت مرتب شده پیمایش نمود.

اندیس اولیه (primary index) و اندیس ثانویه (secondary index) :

بر روی ستون‌های یک جدول می‌توان چندین اندیس را تعریف نمود. اولین اندیسی که بر روی یک ستون از یک جدول گذاشته می‌شود اندیس اولیه (primary index) نامیده می‌شود. عموما این اندیس به کلید اصلی نسبت داده می‌شود، چراکه اولین اندیسی است که بر روی جدول زده می‌شود. توجه داشته باشید که رکوردهای جدول اصلی بر اساس کلیدهای جستجوی اندیس اولیه بصورت منطقی (با استفاده اشاره گرهای انتهای رکورد که توضیح داده شد) مرتب هستند. بنابراین امکان دسترسی بصورت ترتیبی وجود دارد. وقتی پس از اندیس اولیه اقدام به اندیس گذاری‌های دیگری می‌کنیم، اندیس‌های ثانویه را ایجاد می‌کنیم که اندکی با اندیس‌های اولیه متفاوت می‌باشند. در اندیس‌های ثانویه دیگر امکان پیمایش و دسترسی ترتیبی وجود ندارد چراکه اشاره گرهای انتهای رکوردها بر اساس اندیس اصلی (اولیه) مرتب شده اند. بنابراین ما در اندیس‌های ثانویه تنها دسترسی مستقیم خواهیم داشت. شکر زیر نمونه ای از یک اندیس ثانویه را نشان می‌دهد.

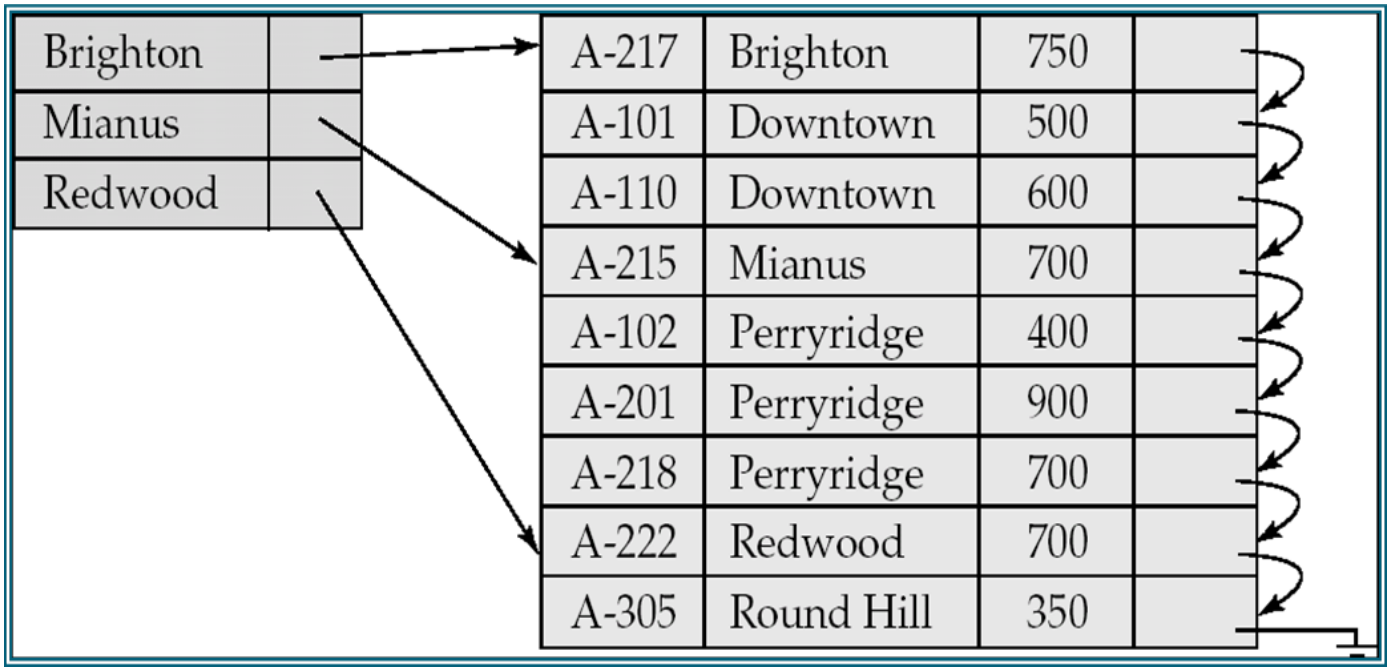


شکل 2 – اندیس ثانویه

همانطور که مشاهده می‌کنید علاوه بر اندیس اصلی (بر روی ستون 2) بر روی سومین ستون این جدول اندیس ثانویه متراکم زده شده است. دقت کنید که هر اشاره گر از جدول اندیس به یک باکت (bucket) اشاره دارد. در هر باکت اشاره گر هایی وجود دارد که به رکورد هایی از جدول اصلی اشاره می‌کنند. فلسفه وجود باکت‌ها اینست که در اندیس‌های ثانویه امکان دسترسی ترتیبی وجود ندارد. بنابراین برای مقادیری تکراری در جدول (مثلا عدد 700) نمی‌توان از اشاره گرهای انتهای رکوردها استفاده نمود. در چنین شرایطی در باکت‌ها اشاره گر مربوط به تمامی رکوردهای حاوی مقادیر تکراری یک کلید را نگهداری می‌کنیم تا بتوان به انها دسترسی مستقیم داشت. همانطور که مشاهده می‌کنید برای بازیابی رکوردهای حاوی مقدار 700 ابتدا از جدول اندیس (که مرتب است) باکت مربوطه را پیدا کرده و سپس از طریق اشاره گرهای موجود در این باکت به رکوردهای حاوی مقدار 700 دستیابی پیدا می‌کنیم.

اندیس‌های تنک (sparse index) :

در این نوع از اندیس‌ها بر خلاف اندیس‌های متراکم، تنها به ازای برخی از کلیدهای جستجو در جدول اندیس اشاره گر نگهداری می‌کنیم. بهمین دلیل فایل اندیس ما کوچکتر خواهد بود (نسبت به اندیس متراکم). در مورد اندیس‌های تنک نیز امکان دسترسی ترتیبی وجود دارد. در شکل زیر نمونه از اندیس تنک (sparse) را مشاهده می‌کنید.



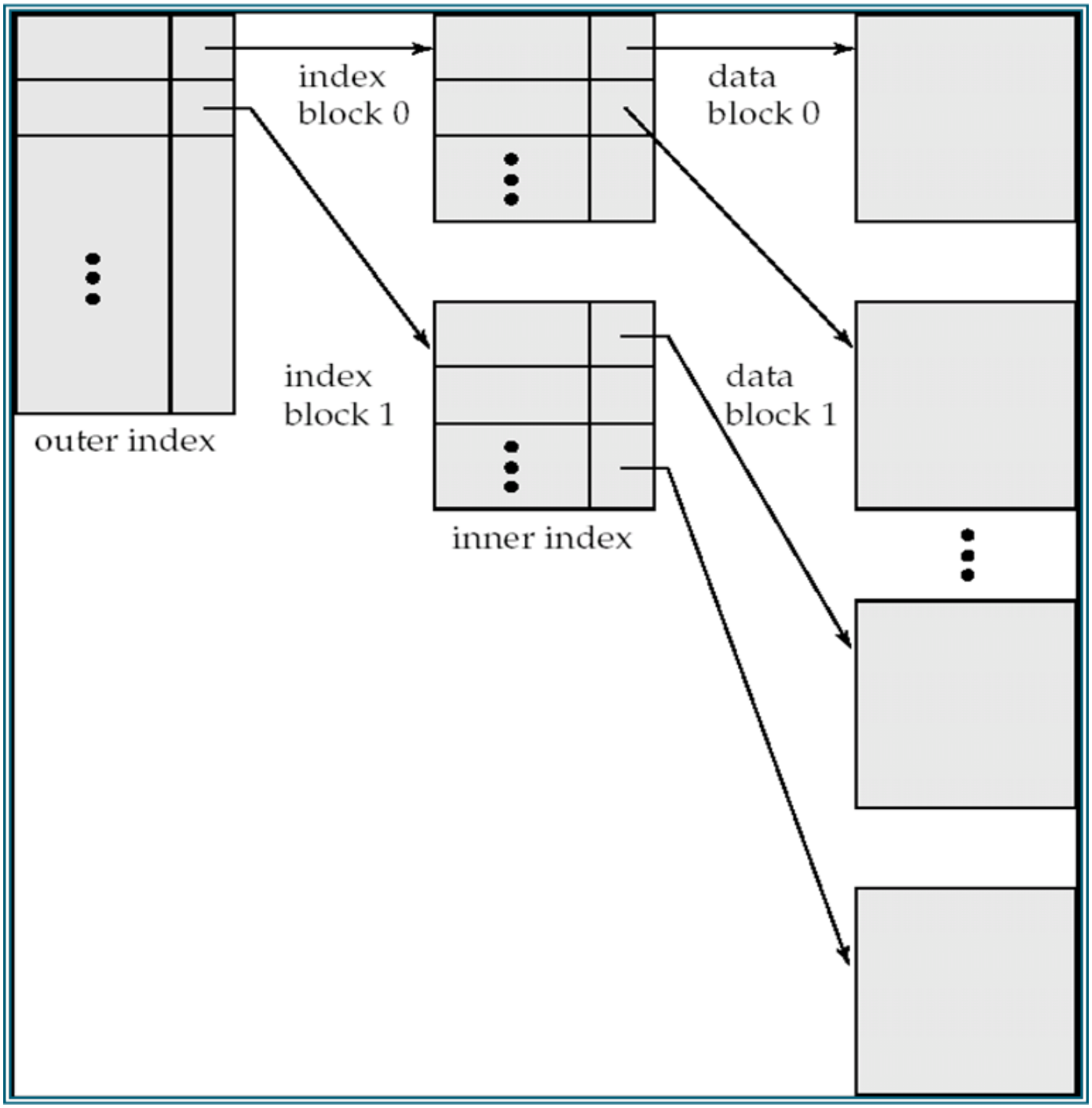
شکل 3 – اندیس تنک (sparse index)

همانند شکل 1، در این شکل نیز اندیس اولیه بر روی ستون دوم زده شده است. اما این بار از اندیس تنک استفاده گردیده است. مشاهده می‌کنید که از میان مقادیر مختلف این ستون تنها برای سه کلید Perryridge ، Brighton و Redwood در جدول اندیس رکورد درج شده است. بنابراین برای دست یابی به کلیدهای دیگر باید ابتدا محل تقریبی آن را با جستجو بر روی جدول اندیس پیدا نمود و سپس از طریق پیمایش ترتیبی به رکورد مورد نظر دست یافت. بعنوان مثال برای بازیابی رکورد حاوی مقدار Mianus ابتدا در جدول اندیس کلیدی که از Mianus کوچکتر باشد (یعنی Brighton) را پیدا می‌کنیم. سپس به رکورد حاولی Brighton می رویم و از آنجا با استفاده از اشاره گرهای انتهایی رکوردها به سمت رکورد حاوی Mianus حرکت می‌کنیم تا به آن برسیم.

نکته بسیار مهمی که در مورد اندیس‌های تنک مطرح می‌شود اینست که سیستم چگونه باید تشخیص دهد که کدام کلیدها را در جدول اندیس نگهداری کند. این تصمیم به مفهوم بلاک‌های حافظه و اندازه انها باز می‌گردد. می‌دانیم که واحد خواندن اطلاعات از حافظه بر اساس بلاک‌ها می‌باشد. این بدان معنی است که در هنگام خواندن رکوردهای جداول بانک اطلاعاتی، عمل خواندن بصورت بلاکی انجام می‌شود. هنگامی که بر روی یک جدول می‌خواهیم اندیس تنک بزنیم ابتدا باید ببینیم این جدول چند بلاک از حافظه را اشغال کرده است. سپس رکوردهای اول هر بلاک را پیدا کرده و به ازای هر بلاک آدرس و کلید جستجوی رکورد اول آن را در جدول اندیس نگهداری کنیم. بدین ترتیب ما به ازای هر بلاک از جدول یک رکورد در فایل اندیس خواهیم داشت و با تخصیص بلاک‌های جدید به ان، طبیعی است که اندیس‌های جدید نیز در فایل اندیس ذخیره خواهند شد.

اندیس‌های چند سطحی (multi-level index)

در دنیایی واقعی معمولا تعداد رکوردهای جداول مورد استفاده بسیار بزرگ است و این اندازه دائما در حال زیاد شدن می‌باشد. افزایش اندازه جداول باعث می‌شود که اندازه فایل‌های اندیس نیز رفته رفته زیاد شود. گفتیم برای کارایی هرچه بیشتر باید جدول اندیس مورد استفاده به حافظه اصلی آورده شود تا تعداد دسترسی‌های ما به حافظه جانبی تا حد امکان کاهش یابد. اما اگر اندازه فایل اندیس ما بسیار بزرگ باشد ممکن است حجم زیادی از حافظه اصلی را بگیرد یا اینکه در حافظه اصلی فضای کافی برای ان وجود نداشته باشد. در چنین شرایطی از اندیس‌های چند سطحی استفاده می‌شود. به بیان دیگر بر روی جدول اندیس نیز اندیس زده می‌شود. تعداد سطوح اندیس ما بستگی به اندازه جدول اصلی دارد و هر چه این اندازه بزرگ‌تر شود، ممکن است باعث افزایش تعداد سطوح اندیس شود. در شکل زیر ساختار یک اندیس دو سطحی را مشاهده می‌کنید.



نکته مهم در مورد اندیس‌های چند سطحی اینست که اندیس‌های سطوح خارجی (outer index) از نوع تنک هستند. این مسئله به این دلیل است که اندازه اندیس‌ها کوچک‌تر شود. چراکه اگر اندیس خارجی از نوع متراکم باشد به این معناست که به ازای هر رکورد غیر تکراری باید یک رکورد در فایل اندیس نیز آورده شود و این مسئله باعث بزرگ شدن اندیس می‌شود. بهمین دلیل سطوح خارجی را در اندیس‌های چند سطحی از نوع تنک می‌گیرند. تنها آخرین سطحی که مستقیما به جدول اصلی اشاره می‌کند از نوع متراکم است. به این سطح از اندیس، اندیس داخلی (inner index) گفته می‌شود.

بروز نگهداشتن اندیس‌ها :

با انجام عملیات درج و حذف بروی جداول، جداول اندیس مربوطه نیز باید بروز رسانی شوند. در این بخش قصد داریم به نحوه بروز رسانی جداول اندیس در زمان حذف و درج رکورد بپردازیم.

بروز رسانی در زمان حذف :

اندیس متراکم :

هنگامی که رکوردی از جدول اصلی حذف می‌شود، در صورتی که بر روی ستون‌های آن اندیس‌های متراکم داشته باشیم، پس از حذف رکورد اصلی باید ابتدا کلید جستجوی ستون مربوط را در جدول اندیس پیدا کنیم. در صورتی که از این کلید تنها یک مقدار در جدول اصلی وجود داشته باشد، اندیس آن را از فایل اندیس حذف کرده و اشاره گرهای انتهای رکوردها را بروز رسانی می‌کنیم. اما اگر از کلید مورد نظر چندین مورد وجود داشته باشد نباید رکورد مورد نظر در جدول اندیس پاک شود. بلکه تنها ممکن است نیاز به ویرایش اشاره گر اندیس باشد. ویرایش در زمانی رخ می‌دهد که اشاره گر جدول اندیس مستقیما به رکوردی اشاره کند که حذف شده باشد، در این صورت باید اشاره گر اندیس را ویراش نمود تا به رکورد بعدی اشاره نماید.

اندیس تنک :

همانند روش قبل ابتدا رکورد اصلی را از جدول حذف می‌کنیم. سپس در فایل اندیس بدنبال کلید جستجوی مربوط به رکورد حذف شده می‌گردیم. در صورتی که کلید مورد نظر در جدول اندیس پیدا شد کلید جستجوی رکورد بعدی در جدول اصلی را جایگزین آن می‌کنیم. چنانچه کلید مربوط به رکورد بعدی در جدول اندیس وجود داشته باشد نیازی به جایگزینی نیست و باید فقط عمل حذف اندیس را انجام داد.

اگر کلید مورد جستجو در جدول اندیس وجود نداشته باشد نیاز به انجام هیچ عملی نیست. در پایان باید اشاره گرهای انتهای رکوردها را ویرایش نمود تا ترتیب منطقی برای پیمایش ترتیبی حفظ شود.

بروز رسانی در زمان درج:

اندیس متراکم:

در هنگام درج یک رکورد جدید، ابتدا باید کلید موجود در رکورد جدید را در جدول اندیس جستجو نمود. در صورتی که کلید مورد نظر در جدول اندیس یافت نشد، باید رکوردی جدیدی در فایل اندیس درج کرد و اشاره گر آن طوری مقدار دهی نمود تا به رکورد جدید اشاره نماید. اگر کلید مورد نظر در جدول اندیس وجود داشته باشد دیگر نیازی بروز رسانی اندیس‌ها نیست و تنها کافی است اشاره گرهای انتهای رکوردها بروز رسانی شوند.

اندیس تنک :

در مورد اندیس‌های تنک کمی پیچیدگی وجود دارد. در صورتی که رکورد جدید باعث تخصیص بلاک (block) جدیدی از حافظه به جدول شود، باید به ازای آن بلاک یک اندیس در جدول اندیس‌ها ایجاد شود و آدر آن بلاک را (که در واقع آدرس رکورد جدید نیز می‌شود) در اشاره گرد اندیس قرار داد. اما درغیز این صورت (در صورتی که رکورد در بلاک‌های موجود ذخیره شود) نیازی به بروز رسانی جدول اندیس‌ها وجود ندارد.

نوع دیگری از اندیس‌های مرتب نیز وجود دارد که اندیس های B-Tree هستند که در سیستم‌های اطلاعاتی دنیای واقعی بیشتر از آنها استفاده می‌شود. به امید خدا در مطالب بعدی این اندیس‌ها را نیز مورد بررسی قرار خواهیم داد.

موفق و پیروز باشید.

نظرات خوانندگان

نویسنده: مجید_فاضلی نسب
تاریخ: ۱۳۹۲/۰۸/۰۳ ۱۳:۲۹

سلام و درود.
فرق Indices و Index چیست ؟

نویسنده: حامد خسروجردی
تاریخ: ۱۳۹۲/۰۸/۰۴ ۹:۵۱

سلام. indices همون جمع index هستش.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۸/۰۴ ۱۳:۳۸

Indices مستقیما از زبان لاتین گرفته شده است. آمریکایی‌ها بیشتر indexes را بکار می‌برند بجای Indices. هر دو هم صحیح هستند.

عنوان: **اصول پایگاه داده - تراکنش ها**
نویسنده: حامد خسروجردی
تاریخ: ۱۳۹۲/۰۸/۰۹ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Database, transaction

در این مقاله آموزشی قصد داریم به یکی از مهمترین و اساسی‌ترین مفاهیم تعریف شده در پایگاه داده بنام تراکنش‌ها بپردازیم.
بعنوان تعریف می‌توان اینگونه بیان نمود که تراکنش یک واحد کاری منطقی است که عملی را بر روی پایگاه داده انجام می‌دهد.
عموما تراکنش‌ها دنباله ای از عملیات پایگاه داده هستند که رویه هم رفته انجام یک کار یا وظیفه را بر عهده دارند. نکته مهمی که در مورد تراکنش‌ها مطرح می‌شود اینست که آنها باید به گونه ای مدیریت شوند که پایگاه داده را از یک وضعیت سازگار و درست (consistent) به وضعیت سازگار دیگری ببرند. به بیان دیگر اگر تراکنش از چند عملیات تشکیل شده باشد، پس از پایان اجرای تمامی عملیات مربوط به تراکنش نباید در داده‌های پایگاه داده هیچ تناقضی با قوانین پایگاه داده (integrity rules) بوجود بیاید.
مزیت استفاده از تراکنش نیز همین مسئله است که به توسعه دهنده نرم افزار این اطمینان را می‌دهد که صحت و درستی پایگاه داده در اثر اجرای دستورات او از بین نخواهد رفت. علاوه بر آن اگر در حین اجرای یکی از دستورات خللی ایجاد گردد، پایگاه داده دوباره به وضعیت سازگار قبلی خود باز گردانده خواهد شد. نسل‌های اولیه سیستم‌های مدیریت پایگاه داده فاقد پیاده سازی تراکنش بودند و بهمین دلیل توسعه دهندگان کار بسیار مشکلی در شبیه سازی این واحدهای یکپارچه منطقی داشتند. خوشبختانه اکثر DBMSهای امروزی این مفهوم مهم را پشتیبانی می‌کنند و نیازی به نگرانی در مورد پیاده سازی آن نیست. تنها کاری که لازم است انجام گیرد کسب مهارت در زمینه استفاده از آنهاست.
تعریف تراکنش‌ها و مشخص کردن عملیات موجود در آنها اغلب توسط خود توسعه دهنده برنامه صورت می‌گیرد. اوست که تعیین می‌کند تراکنشش باید چه عملیاتی را با چه ترتیبی انجام دهد. اما در کنار این قسم از تراکنش‌ها که توسط کاربران تعریف می‌شود، تراکنش‌های دیگری نیز وجود دارند که توسط خود سیستم مدیریت پایگاه داده تعریف می‌شوند. به این قبیل تراکنش‌ها که واحدهای کاری بسیار کوچک و عموما تجزیه ناپذیری هستند تراکنش‌های خودکار یا auto transactions گفته می‌شود. بعنوان مثال اگر ما تراکنشی را تعریف کرده باشیم که شامل یک عمل خواندن و یک عمل درج باشد، در هنگام اجرا سیستم این تراکنش را به دو تراکنش کوچکتر می‌شکند که در یکی عمل خواندن و در دیگری عملی نوشتن و درج را انجام می‌دهد. البته توجه داشته باشید که اگر چه این دو عملیات جدا و مستقل از هم اجرا می‌شوند اما رابطه منطقی آنها با یکدیگر حفظ می‌شود و در صورت خللی در یکی از آنها اثر دیگری نیز بازگردانده شده و پایگاه داده دوباره به حالت قبل از جرا برگردانده می‌شود. به این کار عمل undo شدن تراکنش گفته می‌شود.
گفتیم که تعریف تراکنش توسط کاربر صورت می‌پذیرد و مدیریت آن بر عهده پایگاه داده قرار می‌گیرد. در این میان نکته حائز اهمیتی وجود دارد که در اینجا باید به آن اشاره شود. اندازه تراکنش نقشی بسیار موثر در کارایی پایگاه داده ایفا می‌کند. توجه داشته باشید که اندازه تراکنش‌ها نباید خیلی بزرگ باشد. چراکه منجر به بزرگ شدن بیرویه فایل مربوط به ثبت وقایع پایگاه داده (log file) می‌گردد. تمامی علیات تاثیر گذار بر روی پایگاه داده در این فایل ثبت می‌شوند تا در موقع لزوم بتوان با استفاده از عمل بازیابی و ترمیم پایگاه داده (recovery) را انجام داد. بزرگ بودن این فایل در هنگام ترمیم می‌تواند بر روی کارایی تاثیر گذار باشد. علاوه بر این موضوع اندازه تراکنش‌ها اثر سوء دیگری نیز می‌تواند در پی داشته باشد و آن محدود نمودن درجه همروندی است. یعنی اگر اندازه تراکنش بیش از حد معمول باشد ممکن است بر روی تعداد تراکنش هایی که می‌توانند بطور موازی و همزمان اجرا شوند تاثیر منفی بگذارد. چرا که معمولا در آغاز تراکنش بر روی منابعی که مورد استفاده تراکنش قرار می‌گیرد قفل گذاری می‌شود تا بگونه ای مسئله نواحی بحرانی حل شود. این قفل‌ها زمانی آزاد می‌شوند که تمامی عملیات داخل تراکنش بطور کامل اجرا شده باشند یا اینکه مشکلی در حین اجرا بوجود آید. در این صورت هرچه تراکنش بزرگ‌تر باشد اجرای آن بیشتر طول خواهد کشید و در نتیجه قفل‌های آن نیز دیرتر آزاد می‌شوند. بدین ترتیب سایر تراکنش هایی که می‌خواهند از منابع مشترک استفاده کنند باید تا پایان اجرای تراکنش بزرگ ما منتظر بمانند. این مسئله یعنی کاهش درجه اجرای موازی با همروندی که اگر در سیستم‌های بزرگ به آن دقت نشود به گلوگاهی تبدیل خواهد شد و کارایی را به نحو قابل توجهی کاهش می‌دهد. **تعریف تراکنش‌ها :**

بدنه اصلی هر تراکنش را چهار کلمه کلیدی تشکیل می‌دهند که البته ممکن است صریحا در تعریف توسط کاربر لحاظ نشوند اما این چهار کلمه کلیدی باید در تمامی تراکنش‌ها چه بصورت صریح و چه بصورت ضمنی آورده شوند. این کلمات عبارتند از BEGIN TRANSACTION، END TRANSACTION، ROLLBACK و COMMIT. کلمات کلیدی BEGIN TRANSACTION و END TRANSACTION همانطور که از نامشان پیداست آغاز و پایان یک تراکنش را نشان می‌دهد. اینکه تراکنش از چه نقطه ای آغاز و در چه نقطه ای به پایان رسیده است برای مدیریت آن بسیار مهم و حیاتی است بخصوص در مواقعی که در حین انجام مشکلی پیش بیاید. از کلمه کلیدی

ROLLBACK هنگامی استفاده می‌کنیم که بخواهیم تغییراتی که تا این لحظه بر روی پایگاه داده صورت گرفته است را مجددا بی اثر کنیم و پایگاه داده را به حالت پیش از شروع تراکنش بازگردانیم. توجه داشته باشید که در برخی از مواقع ممکن است این کلمه را خودمان در بدنه تراکنش مستقیما قرار دهیم. بعنوان مثال یک خطای منطقی را در بخشی از روال انجام تراکنش با یک عبارت شرطی تشخیص می‌دهیم و با استفاده از ROLLBACK به مدیریت پایگاه داده اعلام می‌کنیم که عملیات بازگردانی را انجام بده. گاهی ممکن است ما صریحا این کلمه را در تراکنش نیاورده باشیم اما درحین انجام تراکنش خطایی رخ دهد، در این صورت خود سیستم مدیریت پایگاه داده خطا را شناسایی کرده و عملیات مربوط به ROLLBACK را انجام می‌دهد تا صحت و سازگاری پایگاه داده حفظ گردد. کلمه کلیدی COMMIT نیز باید در انتهای تراکنش آورده شود تا به مدیریت پایگاه داده اعلام شود که عملیات کامل شده است و تغییرات باید در پایگاه داده بطور فیزیکی اعمال شوند. توجه داشته باشید که تا زمانی که مدیریت پایگاه داده به دستور COMMIT نرسیده باشد، تغییرات را جهت اعمال بر روی حافظه فیزیکی به واحد مدیریت حافظه نمی‌دهد و بنابراین این تغییرات تا پیش از COMMIT از چشم سایر کاربران مخفی خواهد ماند.

نکته ای که در اینجا وجود دارد این است که فرمان COMMIT به معنی این نیست که بلافاصله تغییرات بر روی دیسک و حافظه جانبی نوشته می‌شود. بلکه به این معنی است که تمامی عملیات تراکنش با موفقیت انجام شده است و سیستم مدیریت پایگاه می‌تواند آنها را برای نوشته شدن در حافظه جانبی به واحد مدیریت حافظه تحویل دهد. در اینجاست که یکی دیگر از پیچیدگی‌های طراحی سیستم مدیریت پایگاه داده روشن می‌شود و آن اینست که این سیستم باید بنحوی این داده‌ها را در فاصله بین COMMIT و نوشته شدن در حافظه برای سایر کاربران قابل مشاهده نماید.

در ادامه نمونه ای از یک تراکنش را مشاهده می‌کنید :

```
BEGIN TRANSACTION;
INSERT INTO SP RELATION {S#  S#('S5'), P#  P#('P1'),
                    QTY  QTY(1000)}};
IF any error occurred THEN GOTO UNDO; END IF;
UPDATE P WHERE P# = P#('P1')
    TOTAL:=TOTAL + QTY(1000);
IF any error occurred THEN GOTO UNDO; END IF;
COMMIT;
GOTO FINISH;
UNDO:  ROLLBACK;
FINISH: RETURN;
```

همانطور که مشاهده می‌کنید تراکنش بالا دارای تمامی بخش‌های اصلی تراکنش که ذکر شد می‌باشد. البته این امکان وجود دارد که صراحتا این کلمات را در تعریف بدنه تراکنش نیاوریم. بعنوان مثال می‌توان از آوردن COMMIT صرف نظر کرد. در این صورت خود سیستم مدیریت پایگاه داده پس از اجرای آخرین دستور تراکنش در صورتی که هیچ خطایی رخ نداده باشد بطور خودکار عمل COMMIT را انجام می‌دهد. این امر در مورد ROLLBACK و END نیز صادق است. اما در مورد BEGIN TRANSACTION نکته ای وجود دارد و آن اینست که ما باید به پایگاه داده اعلام کنیم که بطور خودکار در پایان یک تراکنش برای شروع تراکنش بعدی BEGIN TRANSACTION را لحاظ کند. این کار را باید با دستور SET IMPLICIT TRANSACTION ON انجام دهیم.

گفتیم که وقوع خطا می‌تواند توسط برنامه نویس شناسایی شود و یا توسط سیستم. یک نمونه از تشخیص خطا توسط برنامه نویس را در مثال بالا مشاهده می‌کنید. عموما دراین قبیل خطاها پس از انجام عمل ROLLBACK تراکنش UNDO شده و اجرای آن متوقف می‌شود که اصطلاحا می‌گوییم تراکنش ABORT می‌شود. اما در مورد خطاهایی که خود سیستم تشخیص می‌دهد وضع به این منوال نیست. در شرایط خطا، سیستم پس از UNDO کردن تراکنش عموما آن را ABORT نمی‌کند بلکه مجددا اجرا می‌کند که به این عمل REDO گفته می‌شود. در بخش‌های بعدی بطور کامل در مورد دو عمل REDO و UNDO بحث خواهیم کرد. **ویژگی‌های تراکنش‌ها :**

هر تراکنشی که در سیستم اجرا میشود باید دارای چهار ویژگی باشد. در حقیقت این ویژگی‌ها باید به نحوی تامین شوند تا مقصود و هدف کلی تراکنش‌ها که بردن پایگاه داده از یک وضعیت صحیح به وضعیت صحیح دیگری است برآورده شود. در ادامه هر کدام را یک به یک شرح می‌دهیم : **Atomicity:**

اولین ویژگی ای که یک تراکنش باید داشته باشد اینست که اثری که بر روی پایگاه داده ما می‌گذارد اثری کامل و بدون نقص باشد. به این معنا که اگر قرار است مجموعه از عملیات تغییراتی را اعمال کنند باید تمامی آن تغییرات بر روی جداول اعمال شوند. در صورتی که حتی یکی از عملیات با مشکل مواجه شود باید تاثیرات عملیات قبلی بازگردانده شوند. به بیانی ساده‌تر در تراکنش یا تمامی عملیات باید بطور کامل انجام شوند و یا هیچ یک از آنها نباید اجرا شده و اثرگذار باشند. به این ویژگی Atomicity گفته می‌شود.

توجه داشته باشید که در حین اجرای یک تراکنش احتمالا پایگاه داده به وضعیت غیر سازگار و نادرست خواهد رفت. یکی از وظایف سیستم مدیریت پایگاه داده اینست که این وضعیت ناسازگار را از دید سایر تراکنش‌ها مخفی بسازد تا زمانی که تراکنش

COMMIT شود.

در مورد Atomicity در برخی مقالات و مطالب آموزشی گفته می‌شود که این مفهوم یعنی تراکنش نباید قابل شکسته شدن باشد که این تعریف چندان صحیحی از Atomicity نمی‌باشد. چراکه یک تراکنش در حین اجرا ممکن است بارها و بارها شکسته شود و یا از یک تراکنش بر روی تراکنش دیگری سوئیچ شود. بنابراین مراد از Atomicity همان واحد کاری کامل است نه واحد کاری غیر قابل شکسته شدن. **Consistency** :

تراکنش باید تغییرات را به گونه ای اعمال کند که پایگاه داده را از وضعیت صحیح به وضعیت صحیح دیگری ببرد.از آنجا که صحت پایگاه داده را قوانین جامعیت پایگاه داده (integrity rules) تضمین می‌کنند بنابراین تراکنش باید تغییرات را بگونه ای اعمال کند که این قوانین نقض نشوند. به این خاصیت از تراکنش‌ها Consistency گفته می‌شود. **Isolation:**

عموما برنامه‌های مبتنی بر پایگاه در دنیای واقعی برنامه هایی چند کاربره هستند که در برخی از آنها ممکن است میلیون‌ها تراکنش بطور همزمان با یکدیگر در حال اجرا باشند. در چنین حجم بالایی یکی از مسائلی که مطرح می‌شود اینست که تراکنش‌های موازی تاثیر سوئی بر روی یکدیگر نداشته باشند. بعنوان مثال یکی از مشکلاتی که در اجرای همروند و موازی تراکنش‌ها ممکن است رخ دهد مشکل lost update می‌باشد. بر همین اساس یکی دیگر از ویژگی هایی که یک تراکنش باید داشته باشد که اینست که اثر سوئی بر روی تراکنش‌های همروند دیگر نداشته باشد. به این ویژگی Isolation گفته می‌شود.

در مورد ایزولاسیون (isolation) تراکنش‌ها باید گفت که ایزولاسیون سطوح و درجه بندی هایی دارد که هر کدام از این سطوح مشخص می‌کنند که تراکنش‌ها تا چه حدی اجازه دارند بر روی هم تاثیر گذار باشند. در واقع این سطوح، میزان عایق بندی تراکنش‌ها را نسبت به یکدیگر مشخص می‌کنند. هرچه درجه ایزولاسیون بالاتر باشند به این معنی است که تراکنش‌ها تاثیر کمتری بر روی یکدیگر خواهند داشت. خوب در ظاهر ممکن است این قضیه بسیار خوب در نظر بیاید چرا که به ما اطمینان می دهد که اثر ناخواسته ای بر روی یکدیگر نخواهند داشت. اما باید این نکته را نیز در نظر بگیریم که هر چه درجه ایزولاسیون بالاتر باشد درجه همروندی (concurrency) پایین می‌آید و این به معنای کاهش امکان پردازش موازی تراکنش‌ها می‌باشد. این مسئله در مورد پایگاه‌های داده بسیار بزرگ که میلیون‌ها تراکنش همزمان در خواست اجرا داده می‌شوند به یک مسئله بحرانی و یک گلوگاه می‌تواند تبدیل شود. بنابراین تعیین درجه ایزولاسیون بسیار مهم است و باید با درنظر گرفته شرایط پروژه انجام گیرد.

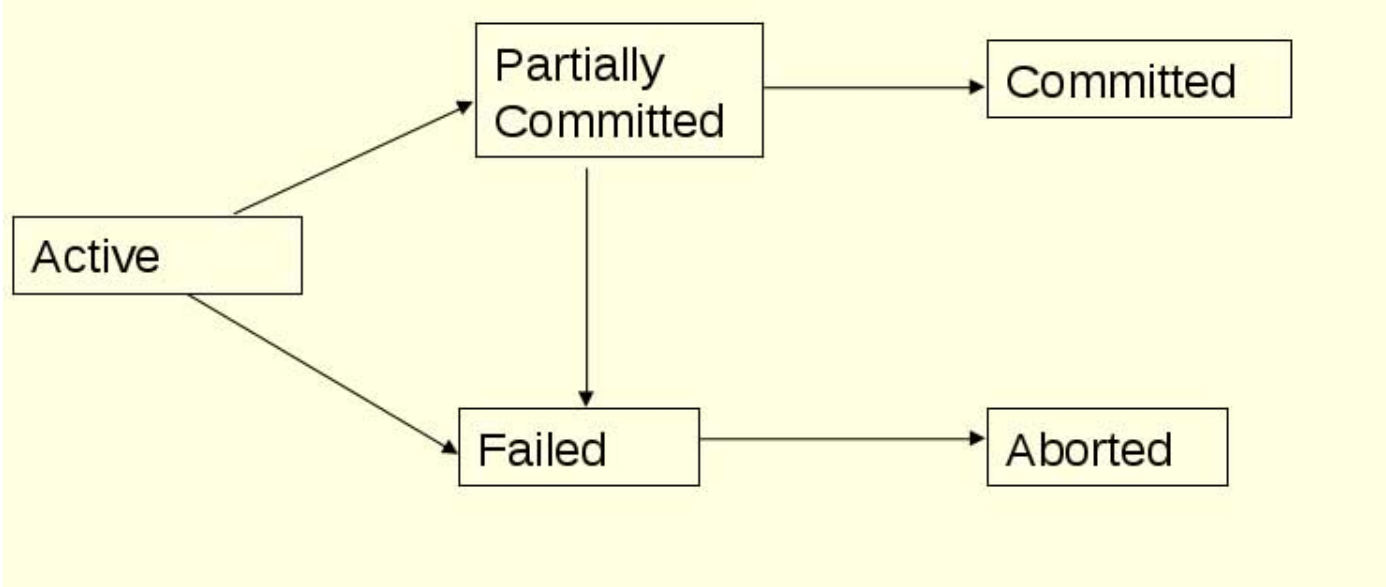
اینکه پایگاه داده ما در چه سطحی از ایزولاسیون باید عمل نماید توسط کاربر تعیین می‌شود. البته بحث در مورد ارجای موازی تراکنش‌ها و ایزولاسیون آنها بسیار مفصل است و انشاالله در مطلبی دیگر به آن خواهیم پرداخت. **Durability** :

تغییراتی که تراکنش‌ها بر روی پایگاه داده می‌گذارند باید بعد از COMMIT شدن آن پایدار و قابل مشاهده باشند. به این خاصیت durability گفته می‌شود. **وضعیت‌های یک تراکنش :**

تراکنش‌ها در سیستم همانند یک موجودیت (entity) فعال است هستند. همانطور که می‌دانید ساده‌ترین موجودیت فعال در سیستم فرآیندها (process) می‌باشند که cpu را بعنوان یک ابزار در اختیار گرفته و وظایفی را انجام می‌دهند. تراکنش نیز یک موجودیت فعال می‌باشد و همانند سایر موجودیت‌های فعال دارای وضعیت هایی (state) می‌باشند که در ادامه هریک شرح داده شده اند :

- فعال (Active) : تراکنشی که در حالت اجرا است در وضعیت فعال می‌باشد.
- کامیت جزئی (Partially Committed): پس از اجرای آخرین دستور تراکنش به وضعیت کامیت جزئی می‌رود.
- شکست (Failed): در این وضعیت، در روند اجرا خطایی رخ داده و اجرای ادامه تراکنش امکان پذیر نمی‌باشد.
- خاتمه (Aborted): پس از تشخیص خطا تراکنش می‌تواند به وضعیت Aborted که در انجا اجرا متوقف شده و تغییرات ROLLBACK می‌شوند.
- Committed: در این وضعیت اجرای تراکنش با موفقیت انجام شده و تراکنش پایان می‌پذیرد.

در ادامه نمودار حالت تراکنش‌ها نشاد داده شده است :



نکته ای که در اینجا لازم به ذکر است اینست که در حالت پس از حالت شکست به دو شکل امکان ادامه کار وجود دارد. در صورتی که خطای منطقی در تراکنش دیده شود که عموما توسط کاربر تشخیص داده می‌شود تراکش پس از شکست به حالت خاتمه برده می‌شود و کار تمام است. اما در برخی از شرایط خطایی سیستم توسط خود سیستم رخ می‌دهد. که در چنین حالاتی پس از شکست تراکنش مجددا تراکنش ممکن است به حالت فعال برگردانده شود و اجرای ان دوباره از ابتدای تراکنش شروع شود. به این وضعیت اصطلاحا REDO شدن تراکنش گفته می‌شود که در بخش RECOVERY و ترمیم پایگاه داده باید به آن پرداخته شود. **اعمال زمان COMMIT:**

در زمان COMMIT (بصورت صریح و یا ضمنی) باید اعمالی انجام شود که در اینجا به آن می‌پردازیم. اولین کاری که صورت می‌گیرد اینست که سیگنالی به DBMS ارسال می‌شود مبنی بر اینکه تراکنش با موفقیت به پایان رسیده است. پس از اینکار سیستم مدیریت پایگاه داده شروع به آزاد کردن قفل هایی می‌کند که در طول اجرای تراکنش بر روی منابع مختلف پایگاه داده زده شده است تا از تاثیر سوء تراکنش‌ها بر روی یکدیگر جلوگیری به عمل آید. علاوه بر کار ذکر شده تغییراتی که توسط تراکنش داده شده است باید پایدار و قابل رویت توسط سایر تراکنش‌ها گردد.

همانطور که در بخش ابتدایی این مطلب آموزشی اشاره کردیم COMMIT به معنی نوشته شدن تغییرات بر روی دیسک سخت نیست. سیستم مدیریت پایگاه داده تنها درخواست نوشتن داده‌ها را به سیستم مدیریت حافظه می‌دهد و نوشتن ان بر عهده مدیریت حافظه می‌باشد. سیستم مدیریت پایگاه داده باید اطلاع داشته باشد که چه تغییراتی نوشته شده است و چه تغییراتی هنوز در حافظه نوشته نشده است. بنابراین یکی دیگر از پیچیدگی‌های طراحی سیستم‌های مدیریت پایگاه داده اینست که تغییراتی را برای سایرین قابل رویت کند که هنوز در حافظه سخت نوشته نشده است. **اعمال زمان ROLLBACK:**

در زمان ROLLBACK ناموفق بودن تراکنش باید به DBMS اطلاع داده شود. پس از انکه سیستم مدیریت پایگاه داده مطلع شد تمامی تغییرات اعمال شده تا آن لحظه را UNDO می‌کند. البته توجه داشته باشید که در این زمان همانند زمان COMMIT قفل‌ها نیز آزاد می‌شوند تا سایر تراکنش‌ها بتوانند از منابع در اختیار این تراکنش استفاده کنند و درجه همروندی پایین نیاید. **پردازش پیام‌ها در زمان اجرای تراکنش‌ها :**

به مثال زیر توجه کنید.

```
Read Sav_Amt
  Sav_Amt := Sav-Amt - 500
    if Sav-Amt <0 then do
       put (“insufficient fund”)
       rollback
       end
    else do
      Write Sav_Amt
      Read Chk_Amt
      Chk_Amt := Chk_Amt + 500
      Write Chk-Amt
      put (“transfer complete”)
End transaction
```

در تراکنش بالا مبلغ 500 دلار از حساب فردی برداشته شده و به حساب دیگر او منتقل می‌شود. همانطور که مشاهده می‌کنید در خلال اجرای یک تراکنش ممکن است پیام هایی را به کاربر نمایش دهیم. حال در نظر بگیرید که در حین اجرا ما پیامی را در خروجی نمایش می‌دهیم و پس از آن تراکنش با شکست مواجه شده و ROLLBACK می‌گردد. در این شرایط پیامی به کاربر مبنی بر انتقال موفق نمایش داده شده است در حالی که در عمل تراکنش با شکست رو به رو شده است. برای حل این مشکل در ضمن کار پیام‌های مختلفی که در خروجی باید نمایش داده شوند بافر می‌شوند تا پس از COMMIT یا ROLLBACK شدن به کاربر نمایش داده شوند. توجه داشته باشید که در زمان بافر کردن پیام ها، انها در دو گره پیام‌های مربوط به COMMIT و پیام‌های زمان ROLLBACK تقسیم می‌شوند تا هرکدام در شرایط خود نمایش داد شوند. این عمل توسط زیر سیستمی از DBMS بنام سیستم مدیریت ارتباطات داده ای (Data Communication Manager) انجام می‌گیرد. **انواع تراکنش‌ها :**

تراکنش‌ها انواع و اقسام مختلفی دارند که به سبب پیچیدگی بعضی از آنها به لحاظ پیاده سازی ممکن است آنها را در برخی از پایگاه داده‌ها نداشته باشیم. **Flat Transactions:**

ساده‌ترین نوع تراکنش‌ها می‌باشند که در تمامی پایگاه‌های داده پشتیبانی می‌شوند و مثال هایی که تا کنون در این نقاله زده شد از این دست می‌باشند. **Distributed Transactions:**

این قبیل تراکنش‌ها مربوط به پایگاه داده‌های توزیع شده می‌باشند که داده‌های آنها بر روی ماشین‌های مختلفی قرار دارند. بر روی هریک از این ماشین‌ها ممکن است DBMSهای مختلفی نیز نصب شده باشد که هر یک سیستم مدیریتی مربطو به خود را دارند. از آنجایی که هر یک از این ماشین‌ها یک سیستم مدیریت پایگاه داده مستقل دارند بنابراین قوانین جامعیتی محلی ای را نیز باید لحاظ نمایند. البته باید توجه داشت که علاوع بر این قوانین محلی یک سری قوانین سراسری نیز وجود خواهد داشت که مربوط به کل پایگاه داده توزیع شده می‌باشد. بعنوان مثال سیستم در یکی سیستم دانشگاهی که در شهرهای مختلفی توزیع شده است، ممکن است بخواهیم تعداد کل دانشجویان ثبت نام شده در سیستم از هزار نفر بیشتر نباشد. عموما درچنین سیستم هایی یک DBMS مدیریت کننده نیز وجود دارد که مسئول برقراری هماهنگی بین سایر DBMSها و نیز اعمال اینگونه قوانین جامعیتی سراسری می‌باشد.

تراکنش‌های توزیع شده یک یا چند تراکنش جزئی تشکیل شده اند که ممکن است هریک از آنها مربوط به یکی از DBMSهای سیستم باشد. چنین تراکنش هایی معمولا ابتدا توسط سیستم مدیریتی مرکزی دریافت می‌شوند و سپس هرکدام از پرس و جوهای داخلی آن به DBMS مربوطه ارسال می‌گردد. اجرای هرکدام از پرس و جوهای جزئی (که خود می‌توانند تراکنشی مستقل نیر باشند) بطور مستقل و محلی بر روی ماشین مربوطه اجرا شده و در انتها نیز نتیجه اجرا به سیستم مدیریتی باز گردانده می‌شود. سیستم مدیریتی مرکزی منتظر می‌ماند که تمامی تراکنش‌ها اعلام COMMIT کنند تا از انجام موفقیت آمیز همه انها اطمینان حاصل نماید. پس از کسب اطمینان کل تراکنش توسط این سیستم مرکزی COMMIT شده و در نتیجه تغییرات بر روی پایگاه داده توزیه شده اعمال می‌شوند. به این سیاست COMMIT کردن، کامیت دو مرحله ای یا Two-phase Commit گفته می‌شود. توجه داشته باشید که در صورتی که هریک از DBMSها اعلام شکست نمایند تمامی تراکنش توزیع شده ROLLBACK می‌گردد.

```
tx_begin();
            execute T1  //at site D
            execute T2  //at site C
            Execute T3  //at site B
            …
tX_commit ();
```

همانطور که در مثال بالا مشاهده می‌کنید تراکنش اصلی از سه تراکنش T1، T2 و T3 تشکیل شده که مر بوط به سه سایت متفاوت می‌باشند. در زمانی تراکنش اصلی COMMIT خواهد شد که هر سه سایت اعلام موفقیت کنند. **تراکنش‌های تو در تو (Nested Transaction):**

این نوع از تراکنش نسبت به دو نوع تراکنش قبلی پیچیدگی بیشتری به لحاظ پیاده سازی و مدیریت دارند. این گونه تراکنش‌ها عموما واحدهای کاری بزرگی هستند که در داخل آنها درختی از تراکنش‌های تو در تو را داریم که مجموعه تمامی انها در نهایت یک کار واحد بلحاظ منطقی را انجام می‌دهند. هر یک از تراکنش‌های داخلی بعنوان یک گره در این ساختار درختی قرار دارند که می‌توانند پدر و یا فرزندانی داشته باشند.

- در تراکنش‌های تو در تو شرایطی حاکم است.
- هر گره در ساختار درختی تراکنش تنها قادر به دیدن برادرهای خود می‌باشد. به بیان دیگر فرزندان برادران خود را نمی‌بیند و نسبت به انها هیچ اطلاعی ندارد.
- در تراکنش‌های تو در تو امکان اجرای موازی فرزندان یک گره وجود دارد.

• امکان اجرای موازی تراکنش‌ها منجر می‌شود به این که تراکنش‌های داخلی قادر به دیدن خروجی حاصل از اجرا همدیگر نباشند.
• هر تراکنشی به طور مستقل ویژگی atomicity را دارد اما پایداری (durability) و کامیت شدن آنها وابسته به پدرانشان می‌باشد.
• در صورتی که پدری تصمیم بگیرد می‌تواند تمامی زیر تراکنش هایش را خاتمه (abort) دهد.

در تراکنش‌های موازی COMMIT شدن یک گره پدر به دو صورت امکان پذیر است. **حالت AND** : در این حالت یک تراکنش در صورتی کامیت خواهد شده که تمامی فرزندان آن با موفقیت اجرا و COMMIT شده باشند. **حالت OR:** در این حالت اگر حتی یکی از تراکنش‌های فرزند نیز موفق به COMMIT شده باشد تراکنش پدر نیز COMMIT خواهد شد. **تراکنش‌های چند سطحی (Multi-level Transactions) :**

این نوع نیز همانند تراکنش‌های تو در تو پیچیده است. از نظر ساختاری تراکنش‌های چند سطحی مشابه تراکنش‌های تو در تو می‌باشند ولی به لحاظ مفهومی با یکدیگر متفاوت هستند. اولین تفاوت موجود بین این دو نوع اینست که هر زیر تراکنشی قادر است خروجی زیر تراکنش‌های دیگر را ببیند. این مسئله باعث می‌شود که تنوانیم زیر تراکنش‌ها را بصورت همروند و موازی اجرا کنیم که این دومین تفاوت مفهومی بین این دو می‌باشد. هنگامی که زیر تراکنش کامل شد (COMMIT) تمامی قفل‌های مربوط به خود را آزاد می‌کند که این مورد نیز در مورد تراکنش‌های تو در تو صادق نمی‌باشد. یکی از مهمترین تفاوت‌های دیگر بین این دو نوع در اینست که در تراکنش‌های چند سطحی تمامی برگ‌ها در یک سطح از درخت قرار دارند و تنها تراکنش‌های برگ هستند که مستقیما به پایگاه داده مراجعه می‌کنند. در مورد کایت شدن نیز شروط مربوط به تراکنش‌های تو در تو در اینجا وجود ندارند و زیر تراکنش‌ها می‌توانند بدون هیچ شرطی کامیت شوند. **تراکنش‌های زنجیره ای (Chained Transaction):**

همانطور که از نام این نوع از تراکنش‌ها پیداست، این تراکنش‌ها از زنجیره ای از زیر تراکنش‌های پی در پی تشکیل شده اند. تا زمانی که تمامی حلقه‌های این زنجیر با موفقیت اجرا نشوند سیستم به حالت سازگاری نخواهد نرفت. دراین نوع از تراکنش‌های COMMIT هر حلقه باعث پایداری شدن (durable) داده‌های در پایگاه داده خواهد شد. این مسئله ممکن است پایگاه داده را به وضعیت ناسازگاری ببرد. در هنگام کامیت شدن هر حلقه قفل‌های مربوط به آن نیز آزاد می‌شود.

حلقه‌های مختلف زنجیره تراکنشی می‌توانند با یکدیگر تبادل اطلاعات کنند. البته توجه داشته باشید که منابعی که هر کدام از آنها بر روی آن کار می‌کنند با دیگری متفاوت می‌باشد. بعنوان نمونه تراکنشی را نظر بگیرد که قصد دارد متوسط مبلغ مکالمه تلفن همراه مشترکان یک مخابرات را محاسبه کند. بدلیل تعداد بالای مشترکان ممکن است این تراکنش را در قالب یک تراکنش زنجیره ای پیاده سازی کنیم که هر حلقه از آن مسئول محاسبه این مبلغ برای ده هزار نفر از کاربران باشد. توجه داشته باشید که برای بدست آوردن مقدار متوسط نیاز داریم که هر زیر تراکنش‌ها قادر به تبادل اطلاعات باشند. از طرفی منابع مورد استفاده آنها (رکورد ها) با یکدیگر متفاوت خواهد بود و نمی‌توانند تغییرات یکدیگر را ببینند. سوالی که مطرح می‌شود اینست که مبادله اطلاعات بین حلقه‌های تراکنش به چه صورت باید انجام شود؟ در جواب این سوال باید گفت که مبادله اطلاعات بین تراکنش‌ها از طریق متغیرهای رابطه ای که هما متغیرهای پایگاه داده هستند انجام می‌گیرد. **SavePoint:**

در برخی شرایط ممکن است بخواهیم در هنگام ROLLBACK مجددا به ابتدای تراکنش باز نگردیم تا مجبور باشیم دوباره کار را از ابتدا از سر بگیریم. بعنوان مثال تا قسمتی از تراکنش پیش رفتیم، به خطایی بر خورد می‌کنیم و می‌خواهیم از نقطه ای خاص از تراکنش کا را از سر بگیریم. در چنین کاربرد هایی از ابزاری بنام SavePoint استفاده می‌کنم.

برای روشن‌تر شدن مفهوم SavePoint فرض کنید قصد داریم بلیطی از تهران به سیدنی رزرو کنیم. برای این منظور ابتدا عمل رزرواسیون را از تهران به دوبی انجام می‌دهیم و سپس از دوبی به سنگاپور و در نهایت از سنگاپور به سیدنی. حال در این بین می‌توانیم در نقطه تهران – دوبی SavePoint قرار دهیم تا در صورت بروز هرگونه خطا مجددا رزرواسیون را از ابتدا آغاز نکنیم. اگر در هنگام رزرو بلیط دوبی – سنگاپور خطایی بروز دهد می‌توانیم به نقطه تهران – دوبی ROLLBACK کنیم و از آنجا مسیر دیگری را انتخاب کنیم. توجه داشته باشید که ROLLBACK به SavePoint وضعیت پایگاه داده به همان نقطه بازگردانده می‌شود.

```
begin transaction();
            s1;
            sp1:= create savepoint(0);
            s2;
            sp2:= create savepoint(0);
            if (condition)
            rollback (spi);
            …
            …
            commit
```

Auto Transaction:

این قبیل تراکنش‌ها تراکنش‌های کوچکی هستند که توسط سیستم تعریف می‌شوند. بعنوان مثال سیستم برای انجام دستورات زیر تراکنش تعریف می‌کند :

```
Alter table, Create, delete, insert, open, drop, fetch, grant, revoke, select, truncate table, update
```

یکی از علت‌های این امر اینست که در صورت بروز خطا در حین این تراکنش‌های خود کار امکان اجرای مجدد هر کدام فراهم گردد. **شروع تراکنش‌ها :**
همانطور که گفته شد برای شروع تراکنش‌ها می‌توانیم صراحتا از BEGIN TRANSACTION استفاده کنیم. البته راهکار دیگری نیز وجود دارد که در آن می‌توانیم به DBMS اعلام کنیم که با پایان یک تراکنش پیش از شروع تراکنش بعدی BEGIN TRANSACTION را قرار بده.
برای این منظور از دستور زیر استفاده می‌کنیم :

```
Set implicit_transaction on
```

برخی از ویژگی‌های تراکنش‌ها را می‌توان تغییر داد. بعنوان مثال می‌توان گفت که تراکنش جاری تنها اجازه خواندن از پایگاه داده را دارد. در این حالت از دستور زیر می‌توان استفاده نمود :

```
SET TRANSACTION READ ONLY
```

همچنین میتوان اجازه تغییر را به آن داد :

```
SET TRANSACTION READ WRITE
```

علاوه بر موارد بالا می‌توان سطح ایزولاسیون تراکنش را با دستود SET تغییر داد. این سطوح در زیر آورده شده اند که بحث در مورد آنها را به مقاله دیگر در مقوله همروندی موکول می‌کنیم.

```
READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE
```

موفق و پیروز باشید

عنوان: **معماری پایگاه داده چند مستاجری (Multi-Tenant Data Architecture)**
نویسنده: محمد پهلوان
تاریخ: ۱۳۹۲/۰۹/۱۱ ۹:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: database design, Database, Cloud computing, SaaS

اعتماد و یا فقدان آن، عامل شماره یک مسدود کردن استفاده از نرم افزار به عنوان خدمات است. معماری پایگاه داده چند مستاجری برای رسیدگی به مشکل نرم افزار به عنوان سرویس (SaaS) که می‌تواند خدمات به تعدادی کلاینت ارائه کند استفاده می‌شود . معماری دیتابیس چند مستاجری وقتی مفید است که یک نمونه از دیتابیس به تعدادی کلاینت خدمات دهد. وقتی که نرم افزارهای محلی نصب می‌کنید نرم افزارهای به عنوان یک سرویس با مشتریان متمرکز، دسترسی به داده‌ها مبتنی بر شبکه با سربار کمتر را فراهم می‌کنند. اما به منظور برخورداری بیشتر از مزیت‌های یک نرم افزار سرویس، یک سازمان باید از سطحی از کنترل روی داده صرفنظر کند و به فروشنده نرم افزار جهت نگهداری و امنیت به دور از چشم آنها اعتماد کند.

برای به درست آوردن این اعتماد، یکی از بالاترین الویت ها، آینده نگری معماری نرم افزار و ساخت یک معماری داده است که باید هر دو قوی و به اندازه کافی ایمن باشد، این دو برای راضی کردن مستاجران و کلاینت هایی که علاقمند هستند کنترل داده‌های حیاتی تجارت خود را به شخص سومی واگذار نمایند، موثر است در حالی که برای اداره کردن و نگهداری مقرون به صرفه است.

سه روش مدیریت چند مستاجری داده

دیتابیس‌های جداگانه برای هر مستاجر

دیتابیس مشترک و schema جداگانه برای هر مستاجر

دیتابیس مشترک و schema مشترک

انتخاب روش مناسب برای برنامه شما به عوامل زیر بستگی دارد :
سایز دیتابیس هر مستاجر

تعداد مستاجران

تعداد کاربران هر مستاجر

نرخ رشد مستاجر

نرخ رشد دیتابیس مستاجر

امنیت

هزینه

1) دیتابیس‌های جداگانه برای هر مستاجر :
ذخیره سازی داده‌های مستاجران در دیتابیس‌های جداگانه ساده‌ترین روش است. در این روش هر مستاجر یک دیتابیس دارد. منابع و کدهای برنامه معمولا در سرور بین همه مستاجران مشترک است اما هر مستاجر مجموعه ای از داده دارد که بطور منطقی از سایر مستاجران جدا شده است.

مزایا :
امنیت بیشتر

سهولت سفارشی سازی برای هر مستاجر

سهولت نگهداری (Backup و Restore) برای هر مستاجر

معایب:
برای نگهداری سخت افزار قوی مورد نیاز است

این روش هزینه بیشتری برای تجهیزات (Backup و Restore) برای هر مستاجر دارد

2) دیتابیس مشترک و schema جداگانه برای هر مستاجر :
خدمات دهی به چندین مستاجر در یک دیتابیس مشترک اما هر مستاجر یک مجموعه از جداول گروهبندی شده دارد که با Schema جدا شده است که برای هر مستاجر الزامی است.

مزایا :
برای دیتابیس برنامه‌های کوچک مناسب است. وقتی تعداد جداول برای هر مشتری کم است

هزینه کمتری نسبت به روش اول دارد

برای مشتریانی که نگران امنیت هستند، سطح منطقی مناسبی برای جداسازی داده ه وجود دارد

معایب:
اطلاعات مستاجران در صورت بروز خطا به سختی restore می‌شود

مدیریت آن برای دیتابیس‌های بزرگ مشکل است

3) دیتابیس مشترک و schema مشترک :

این روش شمامل یک دیتابیس و یک مجموعه از جداول برای چندین مستاجر است. داده‌های جدول می‌تواند شامل رکوردهای هر مستاجر باشد

مزایا :
در مقایسه با روش قبلی، کمترین هزینه سخت افزاری را دارد

می‌تواند مستاجران بیشتری رادر هر سرور پشتیبانی کند

قابلیت بروز رسانی آسان در یک جا برای همه مستاجران

مدیریت آسان دیتابیس و خطا و Backup و Restore

معایب:
امنیت بیشتری مورد نیاز است تا مطمئن شوید هیچکس به اطلاعات سایر مستاجران دسترسی ندارد.

می‌تواند روی کارایی کوئری‌ها تاثیر بگذارد چون تعداد رکوردها زیاد است.

بروزرسانی و سفارشی کردن فقط برای یک مستاجر سخت است

منابع :

http://msdn.microsoft.com/en-us/library/aa479086.aspx

http://www.codeproject.com/Articles/51334/Multi-Tenants-Database-Architecture

نظرات خوانندگان

نویسنده: XPlan
تاریخ: ۱۳۹۲/۰۹/۱۲ ۱۵:۲

با تشکر از شما
در صورت امکان برای پیاده سازی با روش 2 و 3 یک مثال ساده بیان کنید.

نویسنده: محمد پهلوان
تاریخ: ۱۳۹۲/۰۹/۱۲ ۱۷:۵۷

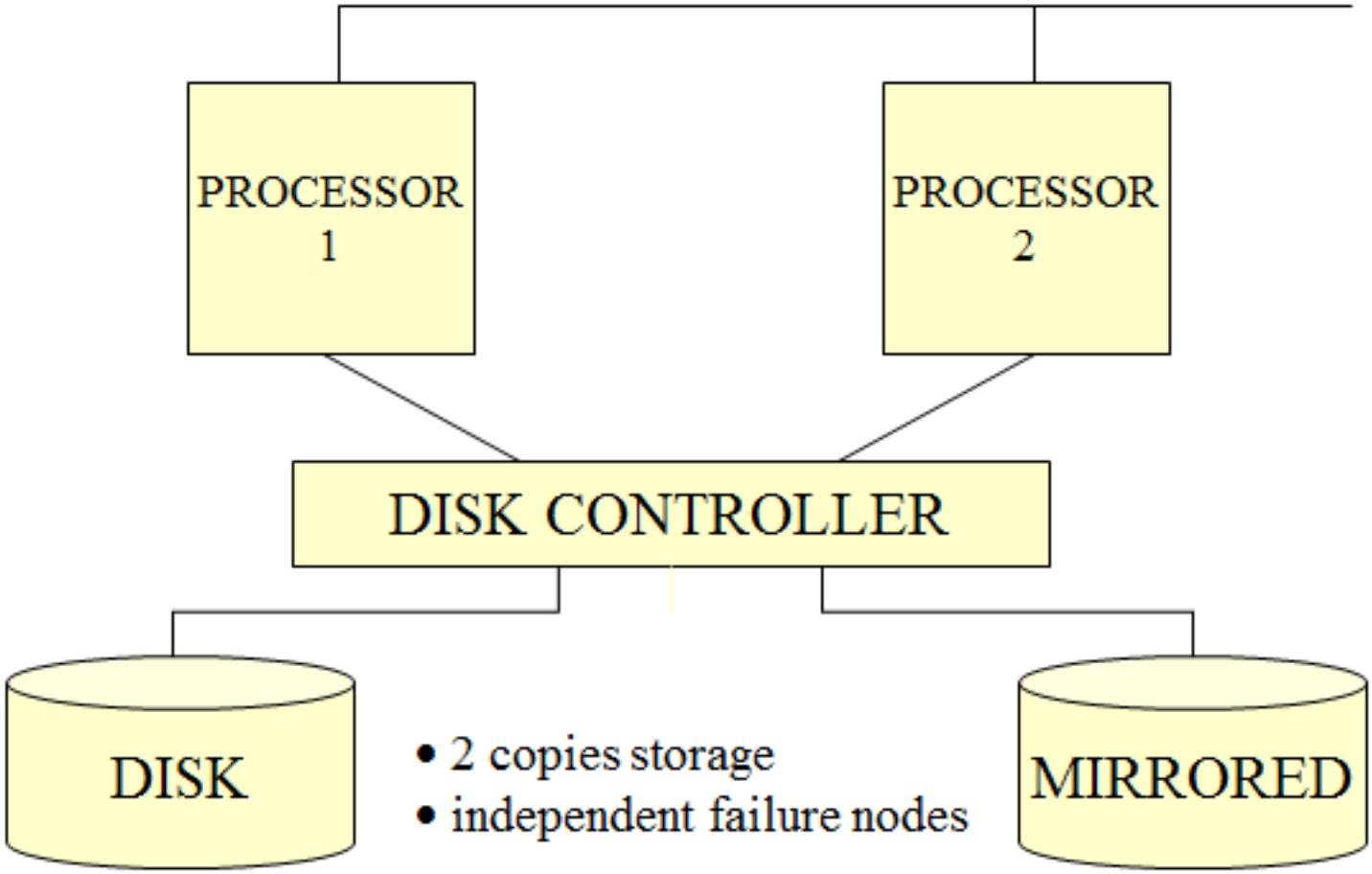
در لینک MSDN که در قسمت منابع آمده روش 2 و 3 به صورت اجمالی و دستورات SQL نحوه کار با schema ذکر شده است.

عنوان: **بازیابی پایگاه داده (database recovery)**
نویسنده: حامد خسروجردی
تاریخ: ۱۳۹۲/۱۲/۱۷ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Database, پایگاه‌داده

در این مقاله آموزشی که یکی دیگر از سری مقالات آموزشی اصول و مبانی پایگاه داده پیشرفته می‌باشد، قصد داریم به یکی دیگر از مقوله‌های مهم در طراحی سیستم‌های مدیریت پایگاه داده (DBMS) بپردازیم. همانطور که در مباحث قبلی بیان کردیم یکی از وظایف سیستم مدیریت پایگاه داده، حفظ سازگاری (consistency) داده‌ها می‌باشد. برای مثال یکی از راهکار هایی که برای این منظور ارائه می‌دهد انجام عملیات در قالب تراکنش هاست که در مبحث مربوط به تراکنش ها مفصل در مورد آن بحث کردیم. با این حال گاهی خطاها و شکست هایی (failure) در حین عملیات ممکن است پیش بیاید که منجر به خروج سیستم از وضعیت سازگار خود گردد. بعنوان مثال ممکن است سخت افزار سیستم دچار مشکل شود، مثلا دیسک از کار بیفتد (disk crash) یا آنکه برق قطع شود. خطاهای نرم افزاری نیز می‌توانند جزو موارد شکست و خرابی بحساب آیند که خطای منطق برنامه (logic) از این نمونه می‌باشد. در چنین شرایطی بحثی مطرح می‌شود تحت عنوان **بازیابی (recovery)** و ترمیم پایگاه داده که در این مقاله قصد داریم در مورد آن صحبت کنیم. بنا به تعریف بازیابی به معنای بازگرداندن یک پایگاه داده به وضعیت سازگار گذشته خود، بعد از وقوع یک شکست یا خرابی است. توجه داشته باشید که اهمیت بازیابی و ترمیم پایگاه داده تا آنجایی است که حدود 10 درصد از سیستم‌های مدیریت پایگاه داده را به خود اختصاص می‌دهند.

آنچه که در اینجا در مورد آن صحبت خواهیم کرد بازیابی بصورت نرم افزاری است که از آن تحت عنوان fail soft نام برده می‌شود. دقت داشته باشید در بیشتر مواقع می‌توان از طریق نرم افزاری عمل بازیابی را انجام داد، اما در کنار راهکارهای نرم افزاری باید حتما اقدامات سخت افزاری ضروری نیز پیش بینی شود. بعنوان مثال گرفتن نسخه‌های پشتیبان یک امر ضروری در سیستم‌های اطلاعاتی است. چرا که گاهی اوقات خرابی‌های فیزیکی باعث از دست رفتن تمامی اطلاعات می‌گردند که در این صورت نسخه‌های پشتیبان می‌توانند به کمک آیند و با کمک آنها سیستم را مجدد بازیابی کرد. در شکل زیر نمونه ای از روش‌های پشتیبان گیری بنام mirroring نشان داده شده است که روش رایجی در سیستم‌های بانک اطلاعاتی بشمار می‌رود. همانطور که در شکل نشان داده شده است در کنار نسخه اصلی (DISK) ، نسخه (MIRROR) آن قرار داده شده است. این دو نسخه کاملا مشابه یکدیگرند و هر عملی که در DICK انجام می‌شود در MIRROR ان نیز اعمال می‌شود تا در مواقع خرابی DISK بتوان از نسخه MIRROR استفاده نمود.

در شکل زیر نمونه بسیار ساده از نحوه لاگ کردن در حین اجرای تراکنش‌ها را مشاهده می‌کنید.



نیازمندی‌های اصلی در بازیابی پایگاه داده

برای آنکه وارد بحث اصلی شویم باید بگوییم در یک نگاه کلی می‌توان گفت که ساختار زیر سیستم بازیابی پایگاه داده بر پایه سه عملیات استوار است که عبارتند از **redo** ، **log** و **undo** . برای آنکه بتوان در هنگام رخ دادن خطا عمل ترمیم و بازیابی را انجام داد، سیستم پایگاه داده با استفاده از مکانیزم لاگ کردن (logging) خود تمامی عملیاتی را که در پایگاه داده رخ می‌دهد و بنحوی منجر به تغییر وضعیت ان می‌گردد را در جایی ثبت و نگهداری می‌کند. اهمیت لاگ کردن وقایع بسیار بالاست، چرا که پس از رخ دادن شکست در سیستم ملاک ما برای بازیابی و ترمیم **فایل‌های لاگ (log files)** می باشند.

سیستم دقیقا خط به خط این لاگ‌ها را می‌خواند و بر اساس وقایعی که رخ داده است تصمیمات لازم را برای بازیابی اتخاذ می‌کند. در حین خواندن فایل‌های لاگ، سیستم برخی از وقایع را باید بی اثر کند. یعنی عمل عکس آنها را انجام دهد تا اثر آن‌ها بر روی پایگاه داده از بین برود. به این عمل undo کردن می‌گوییم که همانطور که در بالا گفته شد یکی از عملیات اصلی در بازیابی است. عمل دیگری وجود دارد بنام انجام مجدد یا redo کردن که در برخی از مواقع باید صورت بگیرد. انجام مجدد همانطور که از اسمش پیداست به این معنی است که عملی که از لاگ فایل خوانده شده است باید مجدد انجام گیرد. بعنوان مثال در فایل لاگ به تراکنشی برخورد می‌کنیم و سیستم تصیم می‌گیرد که آن را مجدد از ابتدا به اجرا در آورد. دقت داشته باشید که سیستم بر اساس قوانین و قواعدی تصمیم می‌گیرد که تراکنشی را redo و یا undo نماید که در ادامه این بحث آن قوانین را باز خواهیم کرد.

در کنار لاگ فایل ها، که مبنای کار در بازیابی هستند، فایل دیگری نیز در سیستم وجود دارد که به DBMS در بازیابی کمک می‌کند. این فایل **raster file** نام دارد که در بخش‌های بعدی این مقاله در مورد آن و کارایی آن بیشتر صحبت خواهیم نمود.

Recovery Manager

مسئولیت انجام بازیابی بصورت نرم افزاری (fail soft) بر عهده زیر سیستمی از DBMS بنام **مدیر بازیابی** (recovery manager) می باشد و همانطور که اشاره شد این زیر سیستم چیزی در حدود 10 در صد DBMS را به خود اختصاص می‌دهد. برای آنکه این زیر سیستم بتواند مسئولیت خود را بنحو احسن انجام دهد بطوری که عمل بازیابی بدون نقص و قابل اعتماد باشد، باید به نکاتی توجه نمود. اولین نکته اینست که در لاگ کردن و همچنین خواندن لاگ فایل به جهت بازیابی و ترمیم پایگاه داده هیچ تراکنشی نباید از قلم بیفتد. تمامی تراکنش‌ها در طول حیات سیستم باید لاگ شود تا بازیابی ما قابل اعتماد و بدون نقص باشد. نکته دوم اینست که اگر تصمیم به اجرای مجدد (redo) تراکنشی گرفته شد، طوری باید عمل Redo انجام شود که بلحاظ منطقی آن تراکنش یک بار انجام شود و تاثیرش یکبار بر دیتابیس اعمال گردد. بعنوان مثال فرض کنید که در طی یک تراکنش مبلغ یک میلیون تومان به حساب شخصی واریز می‌شود. مدتی بعد از اجرای و تمکیل تراکنش سیستم دچار مشکل می‌شود و مجبور به انجام بازیابی می‌شویم. در حین عمل بازیابی سیستم مدیریت بازیابی و ترمیم تصمیم به اجرای مجدد تراکنش مذکور می‌گیرد. در اینجا سیستم نباید مجدد یک میلیون تومان دیگر به حساب ان شخص واریز کند. چرا که در این صورت موجودی حساب فرد دو میلیون تومان خواهد شد که این اشتباه است. سیستم باید طوری عمل کند که پس از انجام مجدد تراکنش باز هم موجودی همان یک میلیون تومان باشد. یعنی مثلا ابتدا یک میلیون کسر و سپس یک میلیون به آن اضافه کند. این مسئله نکته بسیار مهمی است که طراحان DBMS باید حتما آن را مد نظر قرار دهند.

لاگ کردن:

همانطور که گفته شد هر تغییری که در پایگاه داده رخ می‌دهد باید لاگ شود. لاگ کردن به این معنی است که هر گونه عملیاتی که در پایگاه داده انجام می‌شود در فایل هایی به نام فایل لاگ (log file) ذخیره شود. توجه داشته باشید لاگ فایل‌ها در بسیاری از سیستم‌های نرم افزاری دیگر نیز استفاده می‌شود. بعنوان مثال در سیستم عامل ما انواع مختلفی فایل لاگ داریم. بعنوان نمونه یک فراخوانی سیستمی (system call) که در سیستم عامل توسط کاربر انجام می‌شود در فایلی مخصوص لاگ می‌شود. یکی از کاربرد این لاگ فایل شناسایی کاربران بد و خرابکار (malicious users) می تواند باشد که کارهای تحقیقاتی زیادی هم در این رابطه انجام شده و میشود. بدین صورت که می‌توان با بررسی این فایل لاگ و آنالیز فراخوانی‌های یک کاربر بدنبال فراخوانی هایی غیر عادی گشت و از این طریق تشخیص داد که کاربر بدنبال خرابکاری بوده یا خیر. مشابه چنین فایل هایی در DBMS نیز وجود دارد که هدف نهایی تمامی انها حفظ صحت، سازگاری و امنیت اطلاعات می‌باشد.

حال ببینیم در لاگ فایل مربوط به بازیابی اطلاعات چه چیز هایی نوشته می‌شود. در طول حیات پایگاه داده عملیات بسیار گوناگونی انجام می‌گیرد که جزئیات تمامی آنها باید لاگ شود. بعنوان مثال هنگامی که رکوردی درج می‌شود در لاگ فایل باید مشخص شود که در چه زمانی، توسط چه کاربری چه رکوردی، با چه شناسه ای به کدام جدول از دیتابیس اضافه شد. یا اینکه در موقع حذف باید مشخص شود چه رکوردی از چه جدولی حذف شده است. در هنگام بروز رسانی (update) باید علاوه بر مواردی که در درج لاگ می‌کنیم نام فیلد ویرایش شده، مقدار قبلی و مقدار جدید آن نیز مشخص شود. تمامی عملیات ریز لاگ می‌شوند و هیچ عملی نباید از قلم بیفتد. بنابراین فایل لاگ با سرعت زیاد بزرگ خواهد و اندازه دیتابیس نیز افزایش خواهد یافت. این افزایش اندازه مشکل ساز می‌تواند باشد. چراکه معمولا فضایی که ما بر روی دیسک به دیتابیس اختصاص می‌دهیم فضایی محدود است. بهمین دلیل به لحاظ فیزیکی نمی‌توان فایل لاگی با اندازه نامحدود داشت. این در حالی است که چنین فایل هایی باید نامحدود باشند تا همه چیز را در خود ثبت نمایند. برای پیاده سازی ظرفیت نامحدود به لحاظ منطقی یکی از روش‌ها پیاده سازی فایل‌های حلقه ای (circular) است. بدین صورت که هنگامی که سیستم به انتهای فایل لاگ می‌رسد مجددا به ابتدا آن بر می‌گردد و از ابتدا شروع به نوشتن می‌کند. البته چنین ساختار هایی بدون اشکال نیستند. چرا که پس از رسیدن به انتهای فایل و شروع مجدد از ابتدا ما برخی از تراکنش‌های گذشته را از دست خواهیم داد. این مسئله یکی از دلایلی است که بر اساس آن پیشنهاد می‌شود تا جایی که امکان دارد تراکنش‌ها را کوچک پیاده سازی کنیم. گاهی اوقات بر روی لاگ فایل عمل فشرده سازی را نیز انجام می‌دهند. البته فشرده سازی بمعنای رایج ان مطرح نیست. بلکه منظور از فشرده سازی آنست که رکورد هایی که غیر ضروری هستند را حذف کنیم. بعنوان مثال فرض کنید رکوردی را از 50 به 60 تغییر داده ایم. مجددا همان رکورد را از 60 به 70 تغییر می‌دهیم. در این صورت برای این عملیات دو رکورد در فایل لاگ ثبت شده است که در هنگام فشرده سازی در صورت امکان می‌توان ان دو را به یک رکورد تبدیل نمود (تغییر از 50 به 70 را بجای ان دو لاگ کرد). بعنوان مثال دیگر فرض کنید تراکنشی در گذشته دور انجام شده است و با موفقیت کامیت شده است. می‌توان رکوردهای لاگ مربوط به این تراکنش را نیز بنا به شرایط حذف کرد.

دقت داشته باشید که ما عملیاتی مانند عملیات محاسباتی را در این لاگ فایل ثبت نمی‌کنیم. بعنوان مثال اگر دو فیلد با هم باید جمع شوند و نتیجه در فیلدی باید بروز گردد، جمع دو فیل را در سیستم لاگ نمی‌کنیم بلکه تنها مقدار نهایی ویرایش شده را ثبت می‌کنیم. چرا که عملیات محاسباتی در بازیابی ضروری نیستند و ثبت انها تنها باعث بزرگ شدن فایل می‌شود.

در برخی از سیستم‌های حساس، ممکن است برای فایل‌های لاگ هم یک کپی تهیه کنند تا در صورت بروز خطا در لاگ فایل بتوان آن را نیز بازیابی نمود.

انواع رکوردهای لاگ فایل :

در فایل لاگ رکوردهای مختلفی ممکن است درج شود که در این جا به چند نمونه از انها اشاره می‌کنیم:

[start-transaction, T]

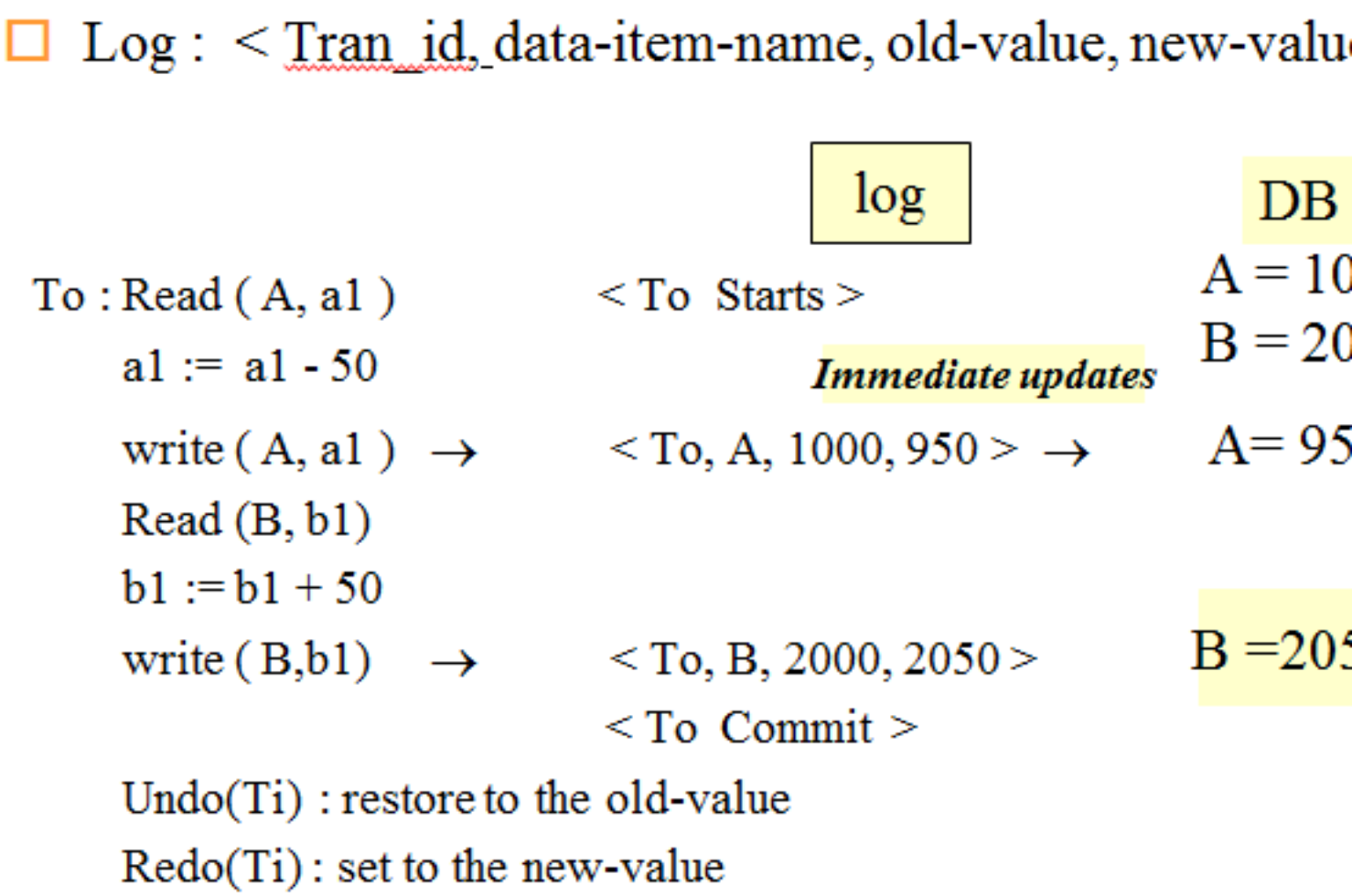
[write-item, T, X, old-value, new-value]

[read-item, T, X]

[commit, T]

در آیتم‌های بالا منظور از T شناسه تراکنش است، X نیز می‌تواند شامل نام دیتابیس، نام جدول، شماره رکورد و فیلدها باشد. البته توجه داشته باشید که این‌ها تنها نمونه هایی از رکوردهای فایل‌های لاگ هستند که در اینجا آورده شده اند. بعنوان مثال رکورد مربوط به عملیات نوشتن خود شامل سه رکورد درج، حذف و بروز رسانی می‌شود.

در شکل زیر نمونه بسیار ساده از نحوه لاگ کردن در حین اجرای تراکنش‌ها را مشاهده می‌کنید.



در این شکل نکته ای وجود دارد که به آن اشاره ای می‌کنیم. همانطور که میبینید در شکل از اصطلاح immediate update استفاده شده است. در برخی از سیستم‌ها تغییرات تراکنش‌ها بصورت فوری اعمال میشوند که اصطلاحا می‌گوییم immediate updates دارند. در مقابل این اصطلاح ما deffered را داریم. در این مدل تغییرات در انتهای کار اعمال می‌شوند (در زمان commit).

: (Write-Ahead Log (WAL

بر اساس آنچه تابحال گفته شد هر تغییری در پایگاه داده شامل دو عمل می‌شود. یکی انجام تغییر (اجرای تراکنش) و دیگری ثبت آن در لاگ فایل. حال سوالی که ممکن است مطرح شود اینست که کدامیک از این دو کار بر دیگری تقدم دارد؟ آیا اول تراکنش را باید اجرا کرد و سپس لاگ آن را نوشت و یا برعکس باید عمل کرد. یعنی پیش از هر تراکنشی ابتدا باید لاگ آن را ثبت کرد و سپس تراکنش را اجرا نمود. بر همین اساس سیاستی تعریف می‌شود بنام سیاست write-ahead log یا WAL که سوال دوم را تایید می‌کند. یعنی می‌گوید هنگامی که قرار است عملی در پایگاه داده صورت گیرد ابتدا باید ان عمل بطور کامل لاگ شود و سپس آن را اجرا نمود. این سیاست هدفی را دنبال می‌کند.

پیش از آنکه هدف این سیاست را توضیح دهیم لازم است نکته ای در مورد عملیات redo و undo بیان شود. شما با این دو عملیات در برنامه‌های مختلفی مانند آفیس، فتوشاپ و غیره آشنایی دارید. اما توجه داشته باشید که در DBMS این دو عملیات از پیچیدگی بیشتری برخوردار می‌باشند. اصطلاحا در پایگاه داده گفته میشود که عملیات redo و undo باید idempotent باشند. معنی idempotent بودن اینست که اگر قرار است تراکنشی در پایگاه داده undo شود، اگر بارها و بارها عمل undo را بر روی آن تراکنش انجام دهیم مانند این باشد این عمل را تنها یکبار انجام داده ایم. در مورد redo نیز این مسئله صادق است.

در تعریف idempotent بودن ویژگی‌های دیگری نیز وجود دارد. بعنوان مثال گفته می‌شود undo بر روی عملی که هنوز انجام نشده هیچ تاثیری نخواهد داشت. این مسئله یکی از دلایل اهمیت استفاده از سیاست WAL را بیان می‌کند. بعنوان مثال فرض کنید می‌خواهیم رکوردی را در جدولی درج کنیم. همانطور که گفتیم دو روش برای این منظور وجود دارد. در روش اول ابتدا رکورد را در جدول مورد نظر درج می‌کنیم و سپس لاگ آن را می‌نویسیم. در این صورت اگر پس از درج رکورد سیستم با مشکل مواجه شود و مجبور به انجام عمل بازیابی شویم، بدلیل آنکه برای بازیابی بر اساس لاگ فایل عمل می‌کنیم و برای درج آن رکورد لاگی در سیستم ثبت نشده است، آن عمل را از دست می‌دهیم. در نتیجه بازیابی بطور کامل نمی‌تواند سیستم را ترمیم نماید. چراکه درج صورت گرفته اما لاگی برای آن ثبت نشده است. در روش دوم فرض کنید بر اساس سیاست WAL عمل می‌کنیم. ابتدا لاگ مربوط به درج رکورد را می‌نویسم. سپس پیش از آنکه عمل درج را انجام دهیم سیستم crash می کند و مجبور به بازیابی می‌شویم. دراین صورت هنگامی که Recovery Manager به رکورد مربوط به عمل درج در لاگ فایل می‌رسد یا باید آن را redo کند و یا undo (بعدا می‌گوییم بر چه اساس تصمیم گیری می‌کند). اگر تصمیم به undo کردن بگیرد بدلیل ویژگی گفته شده، عمل undo بر روی عملی که انجام نشده است هیچ تاثیری در پایگاه داده نخواهد گذاشت. اگر عمل redo را بخواهد انجام دهد نیز بدلیل آنکه لاگ مربوط به عمل درج در سیستم ثبت شده بدون هیچ مشکلی این عمل مجددا انجام می‌گیرد. بنابراین بر خلاف روش قبل هیچ تراکنشی را از دست نمی‌دهیم و سیستم بطور کامل بازیابی و ترمیم می‌شود. به این دلیل است که توصیه می‌شود در طراحی DBMS ها سیاست WAL بکار گیری شود.

نکته بسیار مهمی که در اینجا ذکر آن ضروری بنظر می‌رسد اینست که در هنگام لاگ کردن تراکنش ها، علاوه بر آنکه خود تراکنش لاگ می‌شود و این لاگ‌ها نیز در فایل فیزیکی باید نوشته شوند، عملیات لازم برای Redo کردن و یا undo کردن آن نیز لاگ می‌شود تا سیستم در هنگام بازیابی بداند که چه کاری برای redo و undo کردن باید انجام دهد. توجه داشته باشید در این سیاست، COMMIT تراکنشی انجام نمی‌شود مگر انکه تمامی لاگ‌های مربوط به عملیات redo و undo آن تراکنش در **لاگ فایل فیزیکی** ثبت شود.

قرار دادن checkpoint در لاگ فایل:

گفتیم که در هنگام رخ دادن یک خطا، برای بازیابی و ترمیم پایگاه داده به لاگ فایل مراجعه می‌کنیم و بر اساس تراکنش هایی که در آن ثبت شده است، عمل ترمیم را انجام می‌دهیم. علاوه بر آن، این را هم گفتیم که لاگ فایل، معمولا فایلی بزرگ است که از نظر منطقی با ظرفیت بینهایت پیاده سازی می‌شود. حال سوال اینجاست که اگر بعد گذشت ساعت‌ها از عمر پایگاه داده و ثبت رکوردهای متعدد در لاگ فایل خطایی رخ داد، آیا مدیر بازیابی و ترمیم پایگاه داده باید از ابتدای لاگ فایل شروع به خواندن و بازیابی نماید؟ اگر چنین باشد در بانک‌های اطلاعاتی بسیار بزرگ عمل بازیابی بسیار زمان بر و پر هزینه خواهد بود. برای جلوگیری از این کار مدیر بازیابی پایگاه داده وظیفه دارد در فواصل مشخصی در لاگ فایل نقاطی را علامت گذاری کند تا اگر خطایی رخ داد عمل undo کردن تراکنش را تنها تا همان نقطه انجام دهیم (نه تا ابتدای فایل). به این نقاط checkpoint گفته می‌شود که انتخاب صحیح آنها تاثیر بسیاری در کیفیت و کارایی عمل بازیابی دارد.

نکته بسیار مهمی که در مورد checkpoint ها وجود دارد اینست که آنها چیزی فراتر از یک علامت در لاگ فایل هستند. هنگامی که DBMS به زمانی میرسد که باید در لاگ فایل checkpoint قرار دهد، باید اعمال مهمی ابتدا انجام شود. اولین کاری که در زمان checkpoint باید صورت بگیرد اینست که رکورد هایی از لاگ فایل که هنوز به دیسک منتقل نشده اند، بر روی لاگ فایل فیزیکی بر روی دیسک نوشته شوند. به این عمل flush کردن لاگ رکوردها نیز گفته می‌شود. دومین کاری که در این زمان باید صورت بگیرید اینست که رکوردی خاص بعنوان checkpoint record در لاگ فایل درج گردد. در این رکورد در واقع تصویری از وضعیت دیتابیس در زمان checkpoint را نگهداری می‌کنیم. دقت داشته باشید که در زمان DBMS ، checkpoint برای یک لحظه تمامی تراکنش‌های در حال اجرا را متوقف می‌کند و لیستی از این تراکنش‌ها را در رکورد مربوط به checkpoint نگهداری می‌کند تا در زمان بازیابی بداند چه تراکنش هایی در آن زمان هنوز commit نشده و تاثیرشان به پایگاه داده اعمال نشده است. سومین کاری که در این لحظه بایدا انجام گیرد ایسنت که اگر داده هایی از پایگاه داده هستند که عملیات مربوط به آنها COMMIT شده اند اما هنوز به دیسک منتقل نشده اند بر روی دیسک نوشته شوند.آخرین کاری که باید انجام شود اینست که آدرس رکورد مربوط به checkpoint در فایلی بنام raster file ذخیره شود. علت این کار آنست که در هنگام بازیابی بتوانیم بسرعت آدرس آخرین checkpoint را بدست آوریم.

عمل UNDO :

در اینجا قصد داریم معنی و مفهوم عمل undo را بر روی انواع مختلف تراکنش‌ها را بیان کنیم.
هنگامی که می‌گوییم یک عمل بروز رسانی (update) را می‌خواهیم undo کنیم منظور اینست که مقدار قبلی فیلد مورد نظر را به جای مقدار جدید آن قرار دهیم.
هنگامی که عمل undo را بر روی عملیات حذف می‌خواهیم انجام دهیم منظور اینست که مقدار قبلی جدول (رکورد حذف شده) را مجددا باز گردانیم.
هنگامی که عمل undo را بر روی عملیات درج (insert) می خواهیم انجام دهیم منظور این است که مقدار جدید درج شده در جدول را حذف کنیم.

البته این موارد ممکن است کمی بدیهی بنظر برسد اما برای کامل‌تر شدن این مقاله آموزشی بهتر دانستیم که اشاره ای به آنها کرده باشیم.

انجام عمل بازیابی و ترمیم :

تا اینجا مقدمات لازم برای ترمیم پایگاه داده را گفتیم. حال می‌خواهیم بسراغ چگونگی انجام عمل ترمیم برویم. هنگامی که می‌خواهیم پایگاه داده ای را ترمیم کنیم اولین کاری که باید انجام گیرد اینست که بوسیله raster file ، آدرس آخرین checkpoint لاگ فایل را پیدا کنیم. سپس فایل لاگ را **از نقطه checkpoint به پایین** اسکن می‌کنیم. در هنگام اسکن کردن باید تراکنش‌ها را به دو گروه تفکیک کنیم، تراکنش هایی که باید undo شوند و تراکنش هایی که باید عمل redo بر روی انها انجام گیرد. علت این کار اینست که در هنگام undo کردن از انتهای لاگ فایل به سمت بالا باید حرکت کنیم و برای Redo کردن بصورت عکس، از بالا به سمت پایین می‌آییم. بنابراین جهت حرکت در لاگ فایل برای این دو عمل متفاوت است. بهمین دلیل باید ابتدا تراکنش‌ها تفکیک شوند. اما چگونه این تفکیک صورت می‌گیرد؟

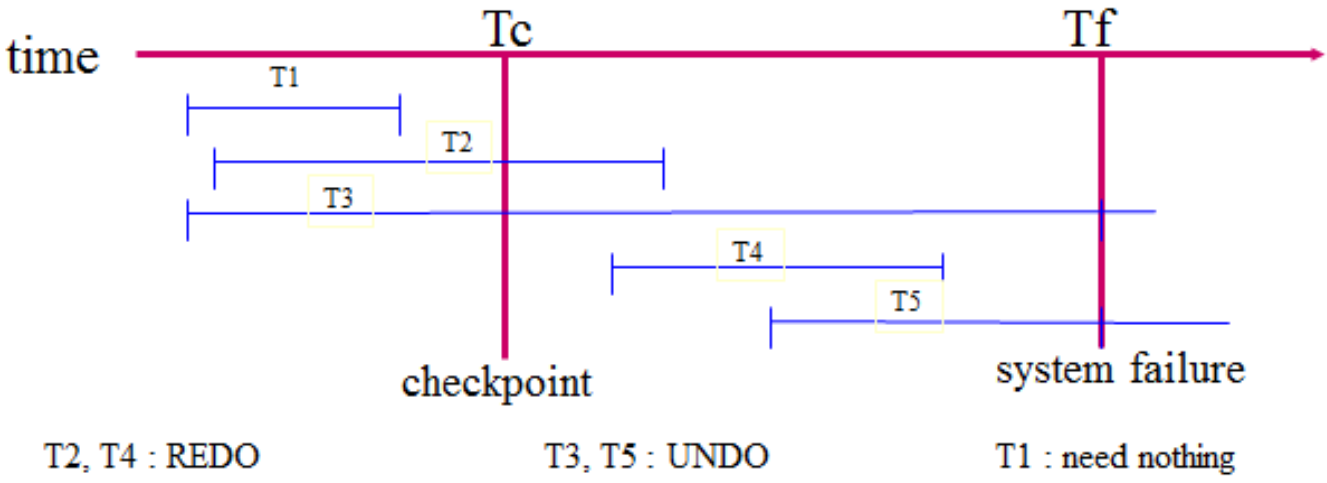
هنگام اسکن کردن (از نقطه checkpoint به سمت انتهای لاگ فایل (لحظه خطا))، هر تراکنشی که رکورد لاگ مربوط به commit آن دیده شود باید در گروه redo قرار گیرد. بعبارت دیگر تراکنش هایی که در این فاصله commit شده اند را در گروه redo قرار می‌دهیم. در مقابل هر تراکنشی که commit آن دیده نشود (commit نشده اند) باید undo شود. باز هم تاکید می‌کنیم که این عمل تنها در فاصله بین آخرین checkpoint تا لحظه وقوع خطا انجام می‌شود.

دقت داشته باشید که در شروع اسکن کردن اولین رکوردی که خوانده می‌شود رکورد مربوط به checkpoint می باشد که حاوی تراکنش هایی است که در زمان checkpoint در حال انجام بوده اند، یعنی هنوز commit نشده اند. بنابراین تمامی این تراکنش‌ها را ابتدا در گروه تراکنش هایی که باید undo شوند قرار می‌دهیم. بمرور که عمل اسکن را ادامه می‌دهیم اگر به تراکنشی رسیدیم که رکورد مربوط به شروع ان ثبت شده باشد، باید آن تراکنش را در لیست undo قرار دهیم. تراکنش هایی که commit آنها دیده شود را نیز باید از گروه undo حذف و به گروه Redo اضافه نماییم. پس از خاتمه عمل اسکن ما دو لیست از تراکنش‌ها داریم. یکی تراکنش هایی که باید Redo شوند و دیگری آنهایی که باید undo گردند.

پس از مشخص شدن دو لیست Redo و Undo ، باید دو کار دیگر انجام شود. اولین کار اینست که تراکنش هایی که باید undo شوند را از پایین به بالا undo کنیم. یکی از دلایل اینکه ابتدا عملیات undo را انجام می‌دهیم ایسنت هنگامی که تراکنش ها commit نشده اند، قفل هایی را که بر روی منابع پایگاه داده زده اند هنوز آزاد نکرده اند. با عمل undo کردن این قفل‌ها را آزاد می‌کنیم و بدین وسیله کمک می‌کنیم تا درجه همروندی پایگاه داده پایین نیاید. پس از خاتمه عملیات undo ، به نقطه checkpoint می رسیم. در این لحظه مانند اینست که هیچ تراکنشی در سیستم وجود ندارد. حالا بر اساس لیست redo از بالا یعنی نقطه checkpoint به سمت پایین فایل لاگ حرکت می‌کنیم و تراکنش‌های موجود در لیست redo را مجدد اجرا می‌کنیم. پس از خاتمه این گام نیز عملیات بازیابی خاتمه می‌یابد می‌توان گفت سیستم به وضعیت پایدار قبلی خود باز گشته است.

برای روشن‌تر شدن موضوع به شکل زیر توجه کنید. در این شکل نقطه Tf زمان رخ دادن خطا را در پایگاه داده نشان می‌دهد. اولین کاری که برای بازیابی باید انجام گیرد، همانطور که گفته شده اینست که آدرس مربوط به زمان Tc) checkpoint) از raster file خوانده شود. پس از این کار از لحظه Tc به سمت Tf شروع به اسکن کردن لاگ فایل می‌کنیم. بدلیل آنکه در زمان Tc دو تراکنش T2 و T3 در حال اجرا بودند (و نام آنها در checkpoint record نیز ثبت شده است)، این دو تراکنش را در لیست redo قرار می‌دهیم. سپس عمل اسکن را به سمت پایین ادامه می‌دهیم. در حین اسکن کردن ابتدا به رکورد start trasnaction مربوط

به تراکنش T4 می رسیم. بهمین دلیل این تراکنش را به لیست undo ها اضافه می‌کنیم. پس از آن به commit تراکنش T2 می رسیم. همانطور که گفته شد باید T2 را از لیست undo ها خارج و به یست تراکنش هایی که باید redo شوند اضافه گردد. سپس به تراکنش T5 می رسیم که تازه آغاز شده است. ان را نیز در گروه undo قرار می‌دهیم. بعد از ان رکورد مربوط به commit تراکنش T4 دیده می‌شود و ان را از لیست undo حذف و لیست redo اضافه می‌کنی. اسکن را ادامه می‌دهیم تا به نقطه Tf می رسیم. در ان لحظه لیست undo ها شامل دو تراکنش T3 و T5 و لیست Redo ها شامل تراکنش های T2 و T4 می باشند. در مورد تراکنش T1 نیز چون پیش از لحظه Tc کامیت شده است عملی صورت نمی‌گیرد.



موفق و پیروز باشید

عنوان: **آموزش PouchDB : قسمت اول (معرفی)**
نویسنده: محمد رضا صفری
تاریخ: ۱۳۹۳/۰۱/۲۵ ۱۷:۴۰
آدرس: www.dotnettips.info
گروه‌ها: NoSQL, Database, PouchDB, IndexedDB

آموزش PouchDB : معرفی

هدف این مقاله بر این است که شما را با دیتابیس PouchDB آشنا سازد .

در مطلب اول هدف فقط آشنایی و نحوه نصب PouchDB قرار خوهد داشت و در مطالب بعدی نحوه آشنایی با نحوه کدنویسی و استفاده به صورت آفلاین یا آنلاین بررسی خواهد شد .

فهرست مطالب :
بخش اول : معرفی PouchDB
شروع به کار با PouchDB
نحوه استفاده از API ها
سوالات متداول در مورد PouchDB
خطاهای احتمالی
پروژه‌ها و پلاگین های PouchDB

PouchDB یک دیتابیس NoSQL می‌باشد که به وسیله Javascript نوشته شده و هدف آن این است که برنامه نویس‌ها بتوانند برنامه‌هایی را توسعه و ارائه کنند که بتواند هم به صورت آفلاین و هم آنلاین سرویس دهی داشته باشند.
برنامه اطلاعات خودش را به صورت آفلاین ذخیره می‌کند و کاربر می‌تواند زمانیکه به اینترنت متصل نیست، از آنها استفاده کند. اما به محض اتصال به اینترنت، دیتابیس خودش را با دیتابیس آنلاین همگام (Sync) می‌کند. اینجاست که قدرت اصلی PouchDB مشخص می‌شود. بزرگترین برتری PouchDB همین است. دیتابیسی است که به صورت توکار قابلیت‌های همگام سازی را دارا می‌باشد و به صورت اتوماتیک این کار را انجام می‌دهد. PouchDB یک پروژه‌ی اوپن سورس است که توسط این افراد به روز می‌شود. البته باید گفت که PouchDB از CouchDB الهام گرفته شده است. اگر شما هم قصد همکاری در این پروژه را دارید بهتر است که راهنمای همکاری را مطالعه کنید .

پشتیبانی مرورگرها

PouchDB پیش زمینه‌های مختلفی دارد که به آن این امکان را می‌دهد تا روی همه مرورگرها و صد البته روی NodeJs کار کند. از IndexedDB بر روی Firefox/Chrome/Opera/IE و WebSql بر روی Safari و همچنین LevelDB بر روی NodeJs استفاده می‌کند.
در حال حاظر PouchDB بر روی مرورگرهای زیر تست شده است:
فایرفاکس 12 و بالاتر
گوگل کروم 19 و بالاتر
اپرا 12 و بالاتر
سافاری 5 و بالاتر
اینترنت اکسپلورر 10 و بالاتر
NodeJs 0.10 و بالاتر
و به صورت شگفت انگیزی در Apache Cordova

برای اطلاعات بیشتر در مورد مرورگرهایی که IndexdDB و WebSql را پشتیبانی می‌کنند به لینک‌های زیر مراجعه کنید:
Can I use IndexedDB

Can I use Web SQL Database

نکته : در صورتی که برنامه شما نیاز دارد تا از اینترنت اکسپلورر نسخه پایینتر از 10 استفاده کند می‌توانید از دیتابیس‌های آنلاین استفاده کنید، که البته دیگر قابلیت استفاده آفلاین را نخواهد داشت.

وضعیت فعلی PouchDB

PouchDB برای مرورگر، فعلا در وضعیت بتا به سر می‌برد و به صورت فعالی در حال گذراندن تست هایی می‌باشد تا باگ‌های آن برطرف شود و به صورت پایدار (Stable) ارائه گردد. البته فقط ممکن است که شما باگی را در قسمت Apiها پیدا کنید که البته Apiها هم در حال حاضر پایدار هستند و گزارشی مبنی بر باگ در آن‌ها موجود نیست. اگر هم باگی پیدا بشود شما می‌توانید PouchDB را بدون ریسک از دست رفتن اطلاعات آپگرید کنید.

PouchDB برای NodeJs فعلا در وضعیت آلفا است و آپگرید کردن ممکن است به اطلاعات شما آسیب بزند. البته با آپدیت به صورت دستی خطری شما را تهدید نخواهد کرد .

نحوه‌ی نصب PouchDB

PouchDB به صورت یک کتابخانه‌ی کوچک و جمع و جور طراحی شده است تا بتواند همه نیازها را برطرف و روی همه نوع Device اعم از موبایل، تبلت، مرورگر و کلا هر چیزی که جاوا اسکریپت را ساپورت می‌کند کار خود را به خوبی انجام بدهد.

برای استفاده از PouchDB میبایست این فایل را با حجم فوق العاده 97 کیلوبایت دانلود کنید و آن را به یک صفحه وب اضافه کنید :

```
<script src="pouchdb-2.1.0.min.js"></script>
```

آخرین نسخه و بهترین نسخه : pouchdb-2.1.0.min.js
برای اطلاع از آخرین آپدیتها و نسخه‌ها به این صفحه در گیت هاب مراجعه کنید .
برای کسانی هم که از NodeJS استفاده می‌کنند نحوه نصب به این صورت است :

```
$ npm install pouchdb
```

نظرات خوانندگان

نویسنده: سعیدزمان
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۱:۲۰

سلام مهندس مطلب بسیار جالبی بود فقط من یک سوال برام پیش اومده که این اطلاعات که میخواد در حالت افلاین استفاده بشه کجا قرار میگیره ؟ ایا امنیت داده‌های حساس رو پایین نمی‌یاره ؟
با تشکر

نویسنده: محمد رضا صفری
تاریخ: ۱۳۹۳/۰۱/۲۶ ۱۵:۱۵

پیشنهاد می‌کنم این اسلاید هارو ببینید : http://www.slideshare.net/wurbanski/nosql-no-security
http://www.slideshare.net/gavinholt/nosql-no-security-16514872
واقعا مفیده و خیلی از سوالات شمارو پاسخ میده .
موفق باشید ./

عنوان: Resource Governor در SQL Server 2008
نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۲ ۱۷:۵۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, Database, SQL Server 2008

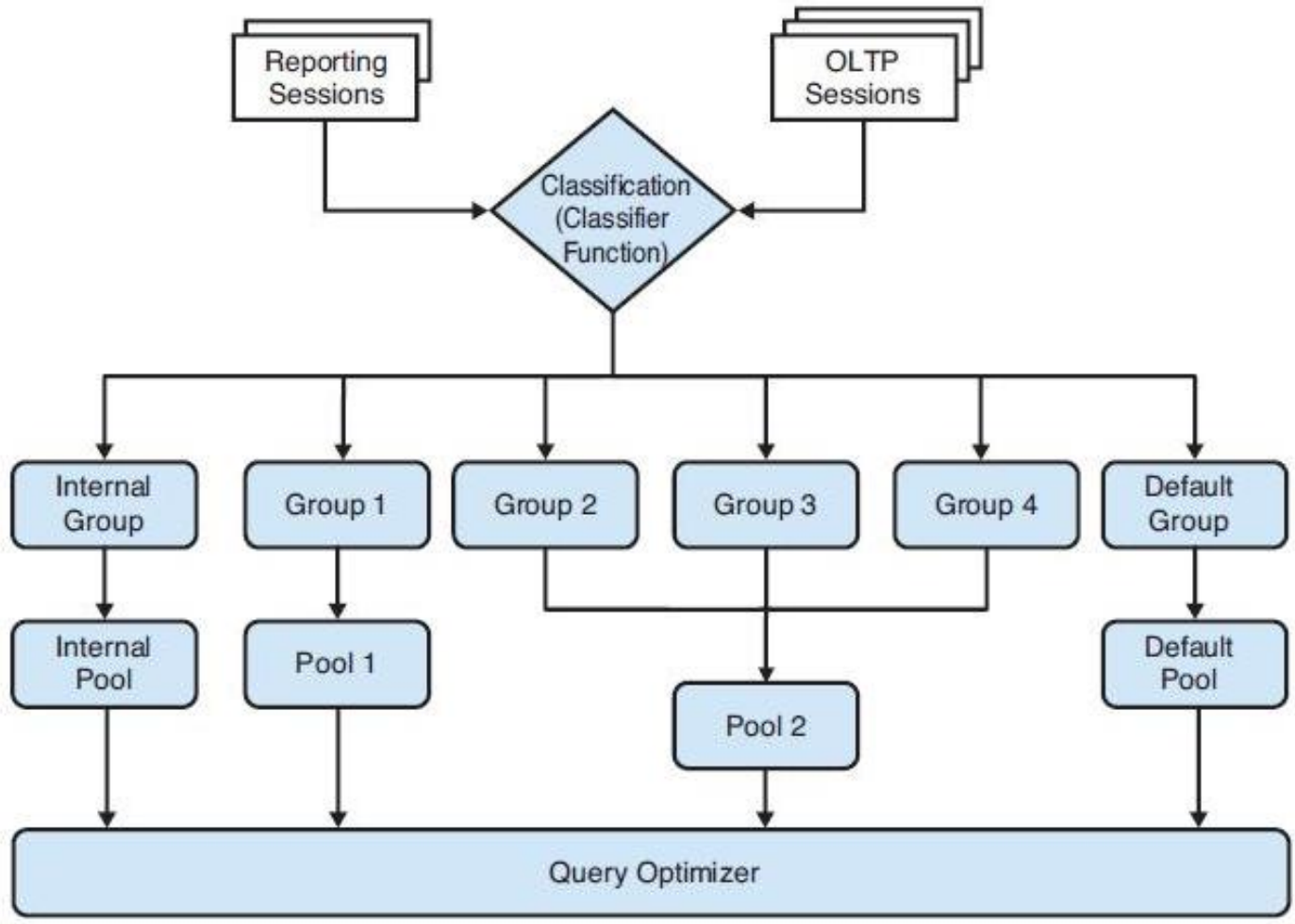
مقدمه

Resource Governor، اجازه می‌دهد تا انواع مختلف Session را بر روی Server طبقه بندی کنید که به نوبه خود چگونگی کنترل تخصیص منابع سرور به فعالیت داده شده را به شما اعطا می‌کند. این قابلیت کمک می‌کند که ادامه فرآیندهای OLTP تضمین شود و یک عملکرد قابل پیش بینی فراهم می‌کند تا توسط فرآیندهای غیر قابل پیش بینی، تحت تاثیر منفی قرار نگیرد. با استفاده از Resource Governor، قادر خواهید بود نحوه دستیابی به Session را به منظور محدود کردن منابع خاص برای SQL Server مشخص کنید. به عنوان مثال می‌توانید مشخص کنید که بیش از 20 درصد از پردازنده یا منابع حافظه به گزارش‌های در حال اجرا اختصاص داده نشود. هنگامیکه این ویژگی فعال باشد، مهم نیست چه تعداد گزارش در حال اجرا است، آنها هرگز نمی‌توانند از تخصیص منابع تعیین شده تجاوز کنند. البته این موضوع عملکرد گزارش گیری را کاهش می‌دهد ولی عملکرد فرآیندهای OLTP حداقل توسط گزارش ها، دیگر تحت تاثیر منفی قرار نمی‌گیرد.

1- بررسی اجمالی Resource Governor:

Resource Governor، با کنترل تخصیص منابع بر حسب Workload کار می‌کند. هنگامی که یک درخواست اتصال به موتور بانک اطلاعاتی ارسال می‌شود درخواست براساس یک تابع رده بندی (Classification function) طبقه بندی می‌شود. تابع رده بندی یک تابع اسکالر است که از طریق T-SQL تعریف می‌شود. تابع رده بندی، اطلاعات را درباره یک اتصال (برای مثال، login ID، application name، hostname، server role) ارزیابی می‌کند، به منظور تشخیص اینکه چگونه آنها را دسته بندی کند. پس از دسته بندی درخواست اتصال، آنها به گروه‌های حجم کاری (Workload Group) که برای رده بندی تعریف شده اند، شکسته می‌شوند. هر Workload Group مرتبط با یک مخزن منابع (Resource Pool) است.

یک Resource Pool، منابع فیزیکی SQL Server را نمایش می‌دهد (در حال حاضر در SQL Server 2008، تنها منابع فیزیکی موجود برای پیکربندی پردازنده و حافظه است) و مقدار حداکثر پردازنده و یا منابع حافظه را که به نوع خاصی از Workload اختصاص داده می‌شود، تعیین می‌کند. هنگامی که یک اتصال طبقه بندی شده و در Workload Group صحیح خود قرار می‌گیرد به این اتصال، پردازنده و منابع حافظه به اندازه نسبت داده شده به آن تخصیص داده می‌شود و سپس Query به Query Optimizer برای اجرا داده می‌شود.



2- اجزای Resource Governor:

Resource Governor، از سه قسمت اصلی تشکیل شده است: Workload Groups، Classification و Resource Pools. درک این سه قسمت و چگونگی تعامل آنها به درک و استفاده از Resource Governor کمک می‌کند.

2-1- Classification:

Classification، فرآیند ارزیابی اتصالات ورودی کاربر و اختصاص آن به یک Workload Group است که توسط منطق موجود در یک تابع تعریف شده توسط کاربر (user-defined function) انجام می‌شود. تابع نام یک Workload Group را برمی گرداند که Resource Governor از آن برای مسیر دهی Session به Workload Group مناسب استفاده می‌کند.

هنگامی که Resource Governor پیکربندی می‌شود فرآیند ورود به سیستم برای یک Session شامل گام‌های زیر است:

- Login authentication
- LOGON trigger execution
- Classification

2-2- Workload Groups:

Workload Groups، ظروفی برای اتصالات مشابه هستند که با توجه به معیارهای طبقه بندی برای هر اتصال گروه بندی می‌شوند. Workload Groups همچنین مکانیسمی برای تجمیع نظارت بر روی منابع مصرفی فراهم می‌کند.

Resource Governor دو Workload Group از پیش تعریف شده دارد: یک گروه داخلی (internal group) و یک گروه پیش فرض (default group).

Internal Workload Group ، تنها توسط فرآیندهای داخلی موتور بانک اطلاعاتی استفاده می‌شود. معیارهای طبقه بندی را برای گروه‌های داخلی نمی‌توانید تغییر دهید و همچنین هیچ یک از درخواست‌های کاربران را برای انتقال به گروه داخلی نمی‌توانید رده بندی کنید، با این حال بر گروه داخلی می‌توانید نظارت کنید.

درخواست‌های اتصال به طور خودکار هنگامی که شرایط زیر وجود دارد، به *Default Workload Group* رده بندی می‌شوند:

- معیاری برای طبقه بندی درخواست وجود ندارد.
- کوششی برای رده بندی درخواستی به گروهی که وجود ندارد.
- خرابی کلی Classification

Resource Governor، در مجموع 20 عدد Workload Group را پشتیبانی می‌کند. از آنجائی که دو عدد از آنها برای Workload Groupهای داخلی و پیش فرض ذخیره شده اند در مجموع 18 عدد Workload Group تعریف شده توسط کاربر (user-defined) می‌توان تعریف نمود.

2-3- Resource pools:

Resource Pool (مخزن منابع)، نشان دهنده تخصیص منابع فیزیکی به SQL Server است. یک Resource Pool از دو بخش تشکیل شده است:

- در بخش نخستین حداقل رزرو منابع را مشخص می‌کنیم، این بخش از مخزن منابع با مخازن دیگر همپوشانی نمی‌کند.
- در بخش دیگر حداکثر ممکن رزرو منابع را برای مخزن مشخص می‌کنیم، تخصیص منابع با مخازن دیگر مشترک است.

در SQL Server 2008 مخزن منابع با تعیین حداقل و حداکثر تخصیص CPU و حداقل و حداکثر تخصیص حافظه تنظیم می‌گردد. با تنظیم حداقل، در دسترس بودن منبع از مخزن تضمین می‌شود. از آنجائی که در هر رزرو حداقل منابع تداخلی نمی‌تواند وجود داشته باشد، مجموع مقادیر حداقل در تمام مخازن از 100% کل منابع Server نمی‌تواند تجاوز کند.

مقدار حداکثر در محدوده بین حداقل و شامل 100% مقدار می‌تواند تنظیم گردد. تنظیم حداکثر نشان دهنده مقدار حداکثری است که یک Session می‌تواند مصرف کند، مادامی که منابع در دسترس باشند و توسط مخزن دیگر که با حداقل مقدار غیر صفر پیکربندی شده، استفاده نشود. هنگامی که یک مخزن با حداقل مقدار غیر صفر تعریف شده، مقدار حداکثر موثر از مخزن‌های دیگر دوباره تنظیم می‌شوند، در صورت لزوم حداکثر مقدار موجود از جمع کل حداقل منابع مخازن دیگر کسر می‌گردد.

برای مثال، دو مخزن تعریف شده توسط کاربر (user-defined) را در نظر بگیرید. مخزن اول Pool1 با مقدار حداقل 20% و مقدار حداکثر 100% تعریف شده، مخزن دیگری Pool2 با مقدار حداقل 50% و مقدار حداکثر 70% تعریف شده است. حداکثر مقدار موثر برایPool1 برابر 50% است (100% منهای مقدار حداقل 50% مخزن Pool2) و حداکثر مقدار موثر برای Pool2، 70% است زیرا حداکثر مقداری است که پیکربندی شده است، گر چه 80% باقی می‌ماند.

بخش مشترکی از مخزن (مقدارش بین مقدار حداقل و مقدار حداکثر موثر است) که برای تعیین مقدار منابع مورد استفاده است، توسط مخزن می‌تواند مصرف شود اگر منابعی موجود باشد و توسط مخازن دیگر مصرف نشده باشد. هنگامی که منابعی توسط یک مخزن مصرف می‌شوند، آنها به یک مخزن مشخص نسبت داده می‌شوند، به بیان دیگر اشتراکی نیستند تا زمانی که فرآیند در آن مخزن به اتمام برسد.

برای توضیح بیشتر یک سناریو که در آن سه مخزن تعریف شده توسط کاربر (user-defined) وجود دارد، را در نظر بگیرید:

PoolA با حداقل مقدار 10% و حداکثر مقدار 100% تعریف می‌شود.

PoolB با حداقل مقدار 35% و حداکثر مقدار 90% تعریف می‌شود.

PoolC با حداقل مقدار 30% و حداکثر مقدار 80% تعریف می‌شود.

مقدار موثر PoolA و مجموع در صد منابع به اشتراک گذاشته PoolA به شرح زیر محاسبه خواهد شد:

(حداکثر مقدار PoolA) - (حداقل مقدار PoolB) - (حداقل مقدار PoolC) = (حداکثر مقدار موثر PoolA)

(حداکثر مقدار موثر PoolA) – (حداقل مقدار PoolA) = (اشتراک PoolA)

جدول زیر مقدار حداکثر موثر و اشتراکی را برای هر مخزن در این پیکربندی نمایش می‌دهد:

Resource Pool	MIN %	MAX %	Effective MAX %	Shared %
Internal	0	100	100	100
Default	0	100	25	25
PoolA	10	100	35	25
PoolB	35	90	50	15
PoolC	30	80	35	5

Internal Pool، منابع مصرف شده توسط فرآیندهای داخلی موتور بانک اطلاعاتی را نشان می‌دهد. این مخزن تنها شامل گروه‌های داخلی است و به هیچ وجه قابل تغییر نیست. مخزن داخلی مقدار ثابت حداقل صفر و حداکثر 100% را دارد و مصرف منابع توسط مخزن داخلی، از طریق تنظیمات در هر مخزن دیگر محدود یا کاسته نمی‌شود.
به عبارت دیگر حداکثر مقدار موثر مخزن داخلی همیشه 100% است. هر workloads در مخزن داخلی برای عملکرد Server حیاتی در نظر گرفته می‌شود و Resource Governor در صورت لزوم اجازه می‌دهد تا مخازن داخلی 100% منابع موجود را مصرف کند حتی اگر به معنی نقض نیازمندیهای منابع از سایر مخازن باشد.
Default Pool، اولین مخزن تعریف شده کاربر است. قبل از هرگونه پیکربندی، Default Pool تنها حاوی Default group است. Default Pool نمی‌تواند ایجاد یا حذف شود اما می‌تواند تغییر کند. Default Pool علاوه بر Default group می‌تواند شامل گروه‌های تعریف شده توسط کاربر (user-defined) نیز باشد.

3- پیکر بندی Resource Governor :

پیکربندی Resource Governor شامل مراحل زیر است:

- فعال کردن Resource Governor
- ایجاد مخازن منابع (Resource Pools) تعریف شده توسط کاربر (user-defined)
- تعریف Workload Groups و نسبت دادن آن به مخازن
- ایجاد Classification function
- ثبت Classification function به Resource Governor

3-1- فعال کردن Resource Governor

پیش از اینکه بتوانید یک Resource Pool را ایجاد کنید، نیاز است تا نخست Resource Governor را فعال کنید.

3-2- تعریف Resource Pool

ویژگی‌های موجود برای یک Resource Pool عبارتند از:
%Name، Minimum CPU %، Maximum CPU%، Min Memory%، Max Memory

3-3- تعریف Workload Group

پس از اینکه Resource Pool را تعریف کردید، گام بعدی ایجاد یک Workload Group و اختصاص آن به Resource Pool مناسب است. چندین workgroup را می‌توان به مخزن (Pool) یکسان نسبت داد اما یک workgroup را به چندین Resource Pool نمی‌توان نسبت داد. خواص انتخابی موجود برای Workload Groups به شما اجازه می‌دهد سطح بهتری از کنترل را روی اجرای دستورات یک Workload Group تنظیم کنید. انتخاب‌های موجود عبارتند از:

3-3-1- Importance :

اهمیت نسبی (کم، متوسط یا بالا) Workload Group درون Resource Pool را تعیین می‌کند. اگر چندین Workload Group را در یک Resource Pool تعریف کنید این تنظیمات تعیین می‌کند که درخواست‌ها در عرض یک Workload Group در اولویت بالاتر یا پائین‌تری از Workload Groupهای دیگر درون همان Resource Pool اجرا شوند، مقدار متوسط تنظیم پیش فرض است. در حال حاضر فاکتورهای وزنی برای هر تنظیم کم برابر 1، متوسط برابر3 و زیاد برابر 9 است. به این معنی که زمانبند به اجرای Sessionهای درون workgroup هائی با اهمیت بالا، سه برابر بیشتر از workgroupهای با اهمیت متوسط و نه برابر بیشتر از

workgroupهای کم اهمیت، مبادرت خواهد کرد.

3-3-2- Maximum Request :

حداکثر تعداد درخواست‌های همزمان که اجازه دارند در یک Workload Group اجرا شوند را مشخص می‌کند. تنظیم پیش فرض، صفر، تعداد نامحدود دستور را اجازه می‌دهد.

3-3-3- CPU Time :

حداکثر مقدار زمان پردازنده در ثانیه را مشخص می‌کند که یک درخواست درون Workload Group می‌تواند استفاده کند. تنظیم پیش فرض، صفر، به معنی نامحدود است.

3-3-4- Memory Grant %:

به صورت در صد، حداکثر مقدار اعطا حافظه برای اجرا (Execution grant memory)، که یک تک دستور از Resource Pool می‌تواند اخذ کند را مشخص می‌کند. این درصد نسبی است از مقدار حافظه ای که به Resource Pool نسبت داده می‌شود. محدوده مجاز مقادیر از 0 تا 100 است. تنظیم پیش فرض 25 است.

Execution grant memory، مقدار حافظه ای است که برای اجرای query استفاده می‌شود (نه برای Buffer کردن یا cache کردن) که می‌تواند صرفه نظر از Resource Pool یا Workload Group توسط تعدادی از Sessionها به اشتراک گذاشته شود. توجه شود که تنظیم این مقدار به صفر از اجرای عملیات Hash Join و دستورات مرتب سازی در Workload Groupهای تعریف شده توسط کاربر (user-defined)جلوگیری می‌کند. همچنین این مقدار توصیه نمی‌شود بیشتر از 70 باشد زیرا ممکن است Server قادر نباشد، اگر Queryهای همزمان در حال اجرا باشند، حافظه آزاد کافی اختصاص دهد.

3-3-5- Grant Time-out :

حداکثر زمان، به ثانیه، که یک query برای یک منبع منتظر می‌ماند تا در دسترس شود را مشخص می‌کند. اگر منبع در دسترس نباشد، فرآیند ممکن است با یک خطای time-out مواجه شود. تنظیم پیش فرض، صفر، به معنی این است که سرور time-out را با استفاده از محاسبات داخلی بر مبنای هزینه پرس و جو (query cost) با تعیین حداکثر زمان برآورد می‌کند.

3-3-6- Degree of Parallelism :

حداکثر درجه موازی سازی (DOP) را برای پرس و جوهای موازی تعیین می‌کند. محدوده مجاز مقادیر از 0 تا 64 است. تنظیم پیش فرض، صفر، به معنی این است که فرآیندها از تنظیمات عمومی استفاده می‌کنند.

3-4- ایجاد یک Classification function

پس از تعریف Resource Pool و Workload Group، به یک Classification function نیاز است که شامل منطق ارزیابی اتصالات و نسبت دادن آنها به Workload Group مناسب است. Classification function برای هر اتصال Session جدید به SQL Server بکار می‌رود. هر Session در Workload Group نسبت داده شده به آن باقی می‌ماند تا زمانی که به پایان برسد، مگر اینکه صراحتاً به یک گروه متفاوت دوباره نسبت داده شود. فقط یک Classification function فعال در هر زمان می‌تواند وجود داشته باشد. در صورت عدم تعریف شدن یا عدم فعال بودن Classification function همه اتصالات به Workload Group Default نسبت داده می‌شوند. Classification function یک نام workgroup که نوع آن SYSNAME است (که یک نام مستعار برای دیتا تایپ nvarchar 128 است.) برمی گرداند. اگر تابع تعریف شده مقدار 'Default'، NULL یا نام گروهی که وجود ندارد را برگرداند، Session به Workload Group Default نسبت داده می‌شود. همچنین اگر به هر دلیلی تابع با موفقیت خاتمه نیابد Session به Workload Group Default نسبت داده می‌شود.

منطق Classification function معمولاً مبتنی بر ویژگی‌های اتصال است و اغلب از طریق مقدار بازگشتی توابع سیستمی از قبیل:

HOST_NAME()، IS_SERVERROLEMEMBER()، IS_MEMBER()، SUSER_SNAME()، SUSER_NAME() و یا APP_NAME()، نام Workload Group اتصال مشخص می‌شود. علاوه بر این توابع می‌توانید از ویژگی‌های توابع دیگر برای ساخت منطق رده بندی استفاده کنید. تابع LOGINPROPERTY() شامل دو ویژگی (DefaultDatabase و DefaultLanguage) می‌باشد که می‌تواند برای Classification function استفاده شود. بعلاوه تابع CONNECTIONPROPERTY() پروتکل‌ها و دسترسی به نقل و انتقالات در شبکه، همچنین جزئیات طرح احراز هویت، Local IP address و TCP Port و Client’s IP Address را برای استفاده اتصالات فراهم می‌کند. برای مثال می‌توانید برای یک اتصال، یک Workload Group نسبت دهید، مبتنی بر اینکه subnet یک اتصال ازکجا می‌آید.

نکته: اگر قصد دارید از هر یک از توابع HOST_NAME() و یا APP_NAME() در تابع رده بندی تان استفاده کنید، توجه داشته باشید این امکان وجود دارد مقادیر بازگردانده شده توسط این توابع توسط کاربران تغییر داده شوند، گر چه به طور کلی گرایش به استفاده از تابع APP_NAME() برای رده بندی اتصالات بیشتر است.

4- بررسی نمونه ای از پیکربندی Resource Governor

برای سادگی، در این قسمت مثالی ارائه می‌شود که از تابع SUSER_NAME() استفاده می‌کند: در گام نخست، دو Resource Pool ایجاد می‌شود (ReportPool و OLTPPool)

```
CREATE RESOURCE POOL [ReportPool] WITH(
min_cpu_percent=0,
        max_cpu_percent=20,
        min_memory_percent=0,
        max_memory_percent=30)
GO
CREATE RESOURCE POOL [OLTPPool] WITH(
min_cpu_percent=80,
        max_cpu_percent=100,
        min_memory_percent=75,
        max_memory_percent=100)
GO
```

در گام بعدی، دو Workload Group ایجاد می‌شود (ReportWG1 و OLTPWG1)

```
CREATE WORKLOAD GROUP [ReportWG1] WITH(
group_max_requests=0,
        importance=Medium,
        request_max_cpu_time_sec=0,
        request_max_memory_grant_percent=25,
        request_memory_grant_timeout_sec=0,
        max_dop=0) USING [ReportPool]
GO
CREATE WORKLOAD GROUP [OLTPWG1] WITH(
group_max_requests=0,
        importance=High,
        request_max_cpu_time_sec=0,
        request_max_memory_grant_percent=25,
        request_memory_grant_timeout_sec=0,
        max_dop=0) USING [OLTPPool]
GO
```

سپس دو Login ایجاد می‌شود (report_user و oltp_user) که در تابع رده بندی استفاده خواهند شد برای مشخص کردن این که اتصالات Seesion به کدام Workload Group نسبت داده شوند. پس از اضافه کردن Loginها به عنوان Userها به Database مورد نظر مان، در بانک اطلاعاتی Master تابع رده بندی (Classification function) را ایجاد می‌کنیم:

```
use master
go
CREATE FUNCTION dbo.WorkgroupClassifier ()
RETURNS SYSNAME WITH SCHEMABINDING
AS
BEGIN
DECLARE @WorkloadGroup SYSNAME = N'Unidentified';
SET @WorkloadGroup = CASE suser_name()
        WHEN N'report_user' THEN N'ReportWG1'
        WHEN N'oltp_user' THEN N'OLTPWG1'
        ELSE N'Unidentified'
        END;
RETURN @WorkloadGroup;
END;
Go
GRANT EXECUTE on dbo.WorkgroupClassifier to public
Go
```

می توان تابع ()WorkgroupClassifier را در محیط SSMS با اجرای دستور زیر برای Loginهای متفاوت تست نمود:

```
select dbo.WorkgroupClassifier()
```

در ادامه دستور زیر برای پیکربندی تابع رده بندی به Resource Governor استفاده می‌شود:

```
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = dbo.WorkgroupClassifier);
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

5- اصلاح پیکربندی Resource Governor:

می‌توانید درمحیط SSMS تنظیمات Resource Pool و Workload Group را تغییر دهید (برای مثال حداکثر استفاده CPU برای یک Resource Pool و یا درجه اهمیت یک Workload Group). متناوباً می‌توان از دستورات T-SQL استفاده نمود.

نکته: پس از اجرای دستورات ALTER RESOURCE POOL یا ALTER WORKLOAD GROUP، برای اعمال کردن تغییرات اجرای دستور ALTER RESOURCE GOVERNOR RECONFIGURE نیاز می‌باشد.

5-1- حذف Workload Group :

یک Workload Group را اگر هر نوع Session فعال نسبت داده شده به آن وجود داشته باشد، نمی‌توان حذف نمود. اگر یک Workload Group شامل Sessionهای فعال باشد، حذف Workload Groupو یا جابجائی آن به یک Resource Pool متفاوت، هنگامی که دستور ALTER RESOURCE GOVERNOR RECONFIGURE برای اعمال نمودن تغییرات فراخوانی می‌شود، با خطا مواجه خواهد شد.

5-2- حذف Resource Pools:

یک Resource Pool را اگر هر نوع Workload Group نسبت داده شده به آن وجود داشته باشد، نمی‌توان حذف نمود. نخست نیاز

دارید Workload Group حذف شود و یا به Resource Pool دیگری جابجا گردد.

5-3- اصلاح Classification function:

اگر نیاز دارید تغییراتی در تابع رده بندی ایجاد نمائید، مهم است توجه داشته باشید که تابع رده بندی تا زمانی که مشخص شده (marked) برای Resource Governor است، نمی‌توان آنرا حذف و یا تغییر داد. پیش از اینکه بتوان تابع رده بندی را اصلاح و یا حذف نمود نخست نیاز دارید Resource Governor را غیر فعال نمائید. متناوباً می‌توان تابع رده بندی را جایگزین کرد با اجرای دستور ALTER RESOURCE GOVERNOR و فرستادن (passing) یک اسم متفاوت برای CLASSIFIER_FUNCTION،همچنین می‌توان با اجرای دستور زیر تابع رده بندی جاری را غیر فعال نمود:

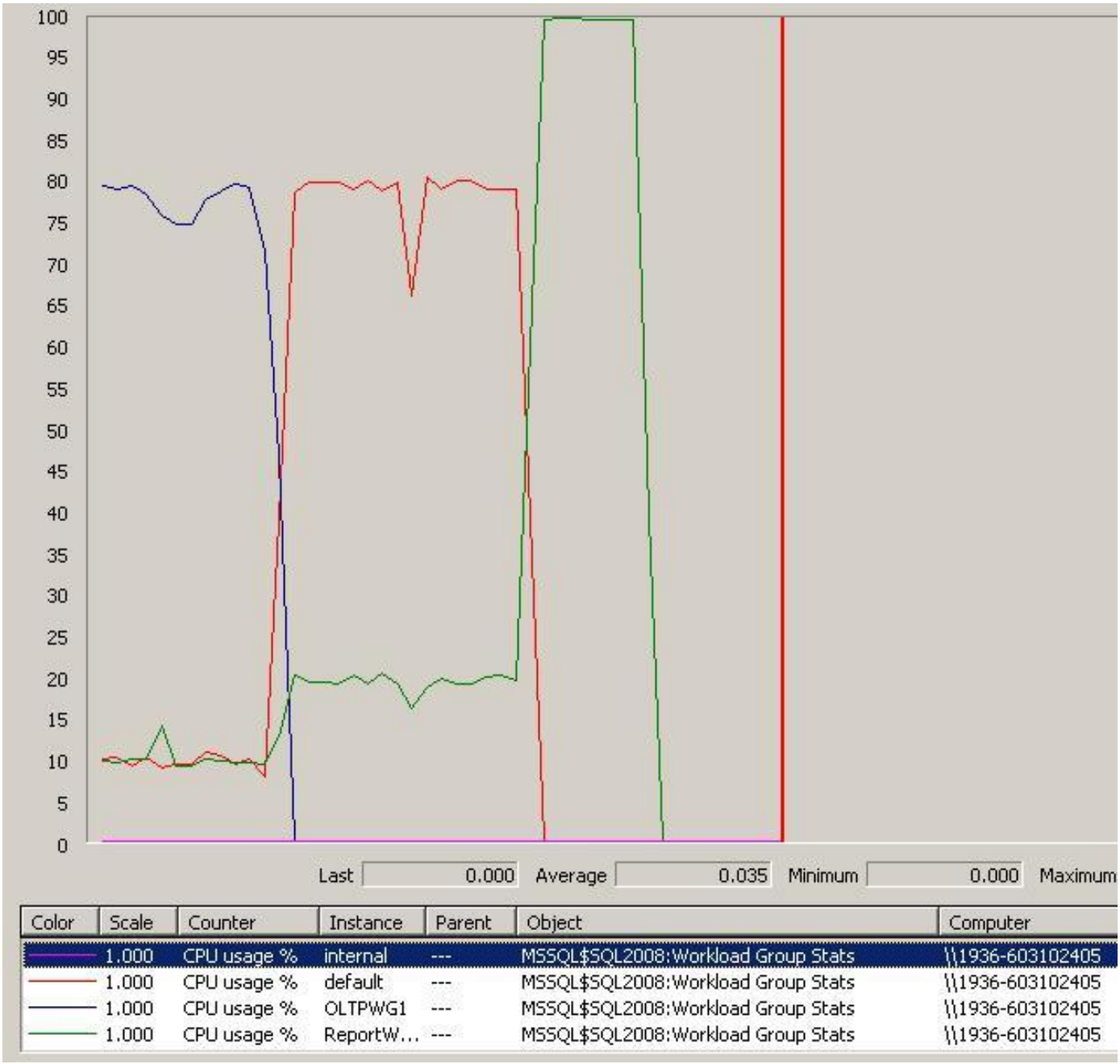
```
ALTER RESOURCE GOVERNOR
WITH (CLASSIFIER_FUNCTION = NULL);
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

تابع رده بندی می‌توان تعریف کرد که نام Workload Group را از جداول یک بانک اطلاعاتی جستجو کند به جای اینکه نام Workload Group به صورت hard-coding و مطابق با ضوابط درون تابع باشد. عملکرد، در موقع دسترسی به جدول برای جستجو کردن نام Workload Group، نباید تا حد زیادی تحت تاثیر قرار گیرد.

6- نظارت بر Resource Governor

با استفاده از Performance Monitor، events و Dynamic Management View (DMV) می‌توان Workload Group و Resource Pool را نظارت (Monitor) کرد. دو شی Performance برای این کار موجود است: SQL Server:Workload Group Stats و SQL Server:Resource Pool Stats

شکل زیر مربوط به پیکر بندی مثال مورد نظرمان می‌باشد:



7- نتیجه گیری

Resource Governor چندین مزیت بالقوه ارائه می‌دهد، در درجه اول قابلیت اولویت بندی منابع Server برای کاربران و برنامه‌های کاربردی (applications) بحرانی، جلوگیری از "runaway" یا درخواست‌های غیر منتظره ای که به شدت و بطور قابل توجهی روی کارائی Server تاثیر منفی می‌گذارند.

ضمناً Resource Governor چندین مشکل بالقوه نیز عرضه می‌کند، برای مثال پیکربندی اشتباه Resource Governor تنها به عملکرد کلی Server آسیب نمی‌رساند بلکه به طور بالقوه روی سرور قفل (Lock) می‌تواند ایجاد کند و نیاز به استفاده از اتصال اختصاصی Administrator برای متصل شدن به SQL Server به منظور اشکال یابی و رفع مشکل می‌باشد. بنابراین توصیه شده است که تنها در صورتی که DBA با تجربه ای هستید و درک خوب و آشنائی خوبی با Workload هائی که روی بانک اطلاعاتی اجرا می‌شوند دارید، Resource Governor را بکار برید. حتی در این صورت، ضروری است که پیکربندی تان را روی یک Server تستی پیش از اینکه روی محیط تولیدی بگسترانید، تست نمائید.

Resource Governor به عنوان یک ویژگی با نام تجاری جدید در SQL Server 2008، با تعدادی محدودیت همراه است که احتمالاً در نسخه‌های بعدی SQL Server حذف خواهد شد، از محدودیت های بارز :

- محدودیت منابع (Resource)، که به CPU و حافظه محدود می‌شوند. I/O Disk و منابع شبکه را در SQL Server 2008 نمی‌توان محدود کرد.
- استفاده از منابع برای Reporting Service، Analysis Service و Integeration Service را نمی‌توان محدود کرد . در این نسخه محدودیت‌های منابع تنها روی هسته موتور بانک اطلاعاتی بکار برده می‌شود.
- محدودیت‌های Resource Governor روی یک SQL Instance تعریف و بکار برده می‌شود.

نظرات خوانندگان

نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۸/۲۱ ۱۳:۳۷

در SQL Server 2012 به منظور تضمین عملکرد تعداد پشتیبانی از مخازن منابع از 20 عدد به 64 عدد افزایش یافته است. همچنین در SQL Server 2014 پشتیبانی از I/O نیز اضافه گردید.(تا پیش از ارائه نسخه 2014 محدودیت روی منابع تنها به CPU و حافظه خلاصه می‌شد)

عنوان: **بررسی Transactions و Locks در SQL Server**
نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۳ ۹:۳۰
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, T-SQL, Database, development

مقدمه SQL Server، با هر تقاضا به عنوان یک واحد مستقل رفتار می‌کند. در وضعیت‌های پیچیده ای که فعالیت‌ها توسط مجموعه ای از دستورات SQL انجام می‌شود، به طوری که یا همه باید اجرا شوند یا هیچکدام اجرا نشوند، این روش مناسب نیست. در چنین وضعیت هایی، نه تنها تقاضاهای موجود در یک دنباله به یکدیگر بستگی دارند، بلکه شکست یکی از تقاضاهای موجود در دنباله، به معنای این است که کل تقاضاهای موجود در دنباله باید لغو شوند، و تغییرات حاصل از تقاضاهای اجراشده در آن دنباله خنثی شوند تا بانک اطلاعاتی به حالت قبلی برگردد.

1- تراکنش چیست؟ تراکنش شامل مجموعه ای از یک یا چند دستور SQL است که به عنوان یک واحد عمل می‌کنند. اگر یک دستور SQL در این واحد با موفقیت اجرا نشود، کل آن واحد خنثی می‌شود و داده هایی که در اجرای آن واحد تغییر کرده اند، به حالت اول برگردانده می‌شود. بنابراین تراکنش وقتی موفق است که هر یک از دستورات آن با موفقیت اجرا شوند. برای درک مفهوم تراکنش مثال زیر را در نظر بگیرید: سهامدار A در معامله ای 400 سهم از شرکتی را به سهامدار B می‌فروشد. در این سیستم، معامله وقتی کامل می‌شود که حساب سهامدار A به اندازه 400 بدهکار و حساب سهامدار B همزمان به اندازه 400 بستانکار شود. اگر هر کدام از این مراحل با شکست مواجه شود، معامله انجام نمی‌شود.

2- خواص تراکنش هر تراکنش دارای چهار خاصیت است (معروف به ACID) که به شرح زیر می‌باشند:

2-1- خاصیت یکپارچگی (Atomicity) یکپارچگی به معنای این است که تراکنش باید به عنوان یک واحد منسجم (غیر قابل تفکیک) در نظر گرفته شود. در مثال مربوط به مبادله سهام، یکپارچگی به معنای این است که فروش سهام توسط سهامدار A و خرید آن سهام توسط سهامدار B، مستقل از هم قابل انجام نیستند و برای این که تراکنش کامل شود، هر دو عمل باید با موفقیت انجام شوند.
اجرای یکپارچه، یک عمل "همه یا هیچ" است. در عملیات یکپارچه، اگر هر کدام از دستورات موجود در تراکنش با شکست مواجه شوند، اجرای تمام دستورات قبلی خنثی می‌شود تا به جامعیت بانک اطلاعاتی آسیب نرسد.

2-2- خاصیت سازگاری (Consistency) سازگاری زمانی وجود دارد که هر تراکنش، سیستم را در یک حالت سازگار قرار دهد (چه تراکنش به طور کامل انجام شود و چه در اثر وجود خطایی خنثی گردد). در مثال مبادله سهام، سازگاری به معنای آن است که هر بدهکاری مربوط به حساب فروشنده، موجب همان میزان بستانکاری در حساب خریدار می‌شود.
در SQL Server، سازگاری با راهکار ثبت فایل سابقه انجام می‌گیرد که تمام تغییرات را در بانک اطلاعاتی ذخیره می‌کند و جزییات را برای ترمیم تراکنش ثبت می‌نماید. اگر سیستم در اثنای اجرای تراکنش خراب شود، فرآیند ترمیم SQL Server با استفاده از این اطلاعات، تعیین می‌کند که آیا تراکنش با موفقیت انجام شده است یا خیر، و در صورت عدم موفقیت آن را خنثی می‌کند. خاصیت سازگاری تضمین می‌کند که بانک اطلاعاتی هیچگاه تراکنش‌های ناقص را نشان نمی‌دهد.

2-3- خاصیت تفکیک (Isolation) تفکیک موجب می‌شود هر تراکنش در فضای خودش و جدا از سایر تراکنش‌های دیگری که در سیستم انجام می‌گیرد، اجرا شود و نتایج هر تراکنش فقط در صورت کامل شدن آن قابل مشاهده است. اگر چندین تراکنش همزمان در سیستم در حال اجرا باشند، اصل تفکیک تضمین می‌کند که اثرات یک تراکنش تا کامل شدن آن، قابل مشاهده نیست. در مثال مربوط به مبادله سهام، اصل تفکیک به معنای این است که تراکنش بین دو سهامدار، مستقل از تمام تراکنش‌های دیگری است که در سیستم به مبادله سهام می‌پردازند و اثر آن وقتی برای افراد قابل مشاهده است که آن تراکنش کامل شده باشد. این اصل در مواردی که سیستم همزمان از چندین کاربر پشتیبانی می‌کند، مفید است.

2-4- پایداری (Durability) پایداری به معنای این است که تغییرات حاصل از نهایی شدن تراکنش، حتی در صورت خرابی سیستم نیز پایدار می‌ماند. اغلب سیستم‌های مدیریت بانک اطلاعاتی رابطه ای، از طریق ثبت تمام فعالیت‌های تغییر دهنده‌ی داده‌ها در بانک اطلاعاتی، پایداری را تضمین می‌کنند. در صورت خرابی سیستم یا رسانه ذخیره سازی داده ها، سیستم قادر است آخرین

بهنگام سازی موفق را هنگام راه اندازی مجدد، بازیابی کند. در مثال مربوط به مبادله سهام، پایداری به معنای این است که وقتی انتقال سهام از سهامدار A به B با موفقیت انجام گردید، حتی اگر سیستم بعداً خراب شد، باید نتیجه‌ی آن را منعکس سازد.

3- مشکلات همزمانی(Concurrency Effects):

3-1- Dirty Read: زمانی روی می‌دهد که تراکنشی رکوردی را می‌خواند، که بخشی از تراکنشی است که هنوز تکمیل نشده است، اگر آن تراکنش Rollback شود اطلاعاتی از بانک اطلاعاتی دارید که هرگز روی نداده است.
اگر سطح جداسازی تراکنش (پیش فرض) Read Committed باشد، این مشکل بوجود نمی‌آید.

3-2- Non-Repeatable Read: زمانی ایجاد می‌شود که رکوردی را دو بار در یک تراکنش می‌خوانید و در این اثنا یک تراکنش مجزای دیگر داده‌ها را تغییر می‌دهد. برای پیشگیری از این مسئله باید سطح جداسازی تراکنش برابر با Repeatable Read یا Serializable باشد.

3-3- Phantoms: با رکوردهای مرموزی سروکار داریم که گویی تحت تاثیر عبارات Update و Delete صادر شده قرار نگرفته اند. به طور خلاصه شخصی عبارت Insert را درست در زمانی که Update مان در حال اجرا بوده انجام داده است، و با توجه به اینکه ردیف جدیدی بوده و قفلی وجود نداشته، به خوبی انجام شده است. تنها چاره این مشکل تنظیم سطح Serializable است و در این صورت بهنگام رسانی‌های جداول نباید درون بخش Where قرار گیرد، در غیر این صورت Lock خواهند شد.

3-4- Lost Update: زمانی روی می‌دهد که یک Update به طور موفقیت آمیزی در بانک اطلاعاتی نوشته می‌شود، اما به طور اتفاقی توسط تراکنش دیگری بازنویسی می‌شود. راه حل این مشکل بستگی به کد شما دارد و بایست به نحوی تشخیص دهید، بین زمانی که داده‌ها را می‌خوانید و زمانی که می‌خواهید آنرا بهنگام کنید، اتصال دیگری رکورد شما را بهنگام کرده است.

4- منابع قابل قفل شدن 6 منبع قابل قفل شدن برای SQL Server وجود دارد و آن‌ها سلسله مراتبی را تشکیل می‌دهند. هر چه سطح قفل بالاتر باشد، Granularity کمتری دارد. در ترتیب آبشاری Granularity عبارتند از:

- **Database:** کل بانک اطلاعاتی قفل شده است، معمولاً طی تغییرات Schema بانک اطلاعاتی روی می‌دهد.
- **Table:** کل جدول قفل شده است، شامل همه اشیای مرتبط با جدول.
- **Extent:** کل Extent (متشکل از هشت Page) قفل شده است.
- **Page:** همه داده‌ها یا کلیدهای Index در آن Page قفل شده اند.
- **Key:** قفلی در کلید مشخصی یا مجموعه کلید هایی Index وجود دارد. ممکن است سایر کلیدها در همان Index Page تحت تاثیر قرار نگیرند.
- **Row or Row Identifier (RID):** هر چند قفل از لحاظ فنی در Row Identifier قرار می‌گیرد ولی اساساً کل ردیف را قفل می‌کند.

5- تسریع قفل (Lock Escalation) و تاثیرات قفل روی عملکرد اگر تعداد آیتم‌های قفل شده کم باشد نگهداری سطح بهتری از Granularity (مثلاً RID به جای Page) معنی دار است. هرچند با افزایش تعداد آیتم‌های قفل شده، سربار مرتبط با نگهداری آن قفل‌ها در واقع باعث کاهش عملکرد می‌شود، و می‌تواند باعث شود قفل به مدت طولانی‌تری در محل باشد(هر چه قفل به مدت طولانی‌تری در محل باشد، احتمال این که شخصی آن رکورد خاص را بخواهد بیشتر است).
هنگامی که تعداد قفل نگهداری شده به آستانه خاصی برسد آن گاه قفل به بالاترین سطح بعدی افزایش می‌یابد و قفل‌های سطح پایین‌تر نباید به شدت مدیریت شوند (آزاد کردن منابع و کمک به سرعت در مجادله).
توجه شود که تسریع مبتنی بر تعداد قفل هاست و نه تعداد کاربران.

6- حالات قفل (Lock Modes): همانطور که دامنه وسیعی از منابع برای قفل شدن وجود دارد، دامنه ای از حالات قفل نیز وجود دارد.

6-1- Shared Locks (S): زمانی استفاده می‌شود، که فقط باید داده‌ها را بخوانید، یعنی هیچ تغییری ایجاد نخواهید کرد. Shared Lock با سایر Shared Lockهای دیگر سازگار است، البته قفل‌های دیگری هستند که با Shared Lock سازگار نیستند. یکی از کارهایی که Shared Lock انجام می‌دهد، ممانعت از انجام Dirty Read از طرف کاربران است.

6-2- Exclusive Locks (X): این قفل‌ها با هیچ قفل دیگری سازگار نیستند. اگر قفل دیگری وجود داشته باشد، نمی‌توان به Exclusive Lock دست یافت و همچنین در حالی که Exclusive Lock فعال باشد، به هر قفل جدیدی از هر شکل اجازه ایجاد شدن در منبع را نمی‌دهند.
این قفل از اینکه دو نفر همزمان به حذف کردن، بهنگام رسانی و یا هر کار دیگری مبادرت ورزند، پیشگیری می‌کند.

6-3- Update Locks (U): این قفل ها نوعی پیوند میان Shared Locks و Exclusive Locks هستند.
برای انجام Update باید بخش Where را (در صورت وجود) تایید اعتبار کنید، تا دریابید فقط چه ردیف هایی را می‌خواهید بهنگام رسانی کنید. این بدان معنی است که فقط به Shared Lock نیاز دارید، تا زمانی که واقعاً بهنگام رسانی فیزیکی را انجام دهید. در زمان بهنگام سازی فیزیکی نیاز به Exclusive Lock دارید.
Update Lock نشان دهنده این واقعیت است که دو مرحله مجزا در بهنگام رسانی وجود دارد، Shared Lock ای دارید که در حال تبدیل شدن به Exclusive Lock است. Update Lock تمامی Update Lockهای دیگر را از تولید شدن باز می‌دارند، و همچنین فقط با Shared Lock و Intent Shared Lockها سازگار هستند.

6-4- Intent Locks: با سلسله مراتب شی سر و کار دارد. بدون Intent Lock، اشیای سطح بالاتر نمی‌دانند چه قفلی را در سطح پایین‌تر داشته اید. این قفل‌ها کارایی را افزایش می‌دهند و 3 نوع هستند:

6-4-1- Intent Shared Lock (IS: Shared Lock) در نقطه پایین‌تری در سلسله مراتب، تولید شده یا در شرف تولید است. این نوع قفل تنها به Table و Page اعمال می‌شود.

6-4-2- Intent Exclusive Lock (IX): همانند Intent Shared Lock است اما در شرف قرار گرفتن در آیتم سطح پایین‌تر است.

6-4-3- Shared With Intent Exclusive (SIX: Shared Lock) در پایین سلسله مراتب شی تولید شده یا در شرف تولید است اما Intent Lock قصد اصلاح داده‌ها را دارد بنابراین در نقطه مشخصی تبدیل به Intent Exclusive Lock می‌شود.

6-5- Schema Locks: به دو شکل هستند:

6-5-1- Schema Modification Lock (Sch-M): تغییر Schema به شی اعمال شده است. هیچ پرس و جویی یا سایر عبارت‌های Create، Alter و Drop نمی‌توانند در مورد این شی در مدت قفل Sch-M اجرا شوند. با همه حالات قفل ناسازگار است.

6-5-2- Schema Stability Lock (Sch-S): بسیار شبیه به Shared Lock است، هدف اصلی این قفل پیشگیری از Sch-M است وقتی که قبلاً قفل هایی برای سایر پرس و جو-ها (یا عبارت‌های Create، Alter و Drop) در شی فعال شده اند. این قفل با تمامی انواع دیگر قفل سازگار است به جز با Sch-M.

6-6- Bulk Update Locks (BU): این قفل‌ها بارگذاری موازی داده‌ها را امکان پذیر می‌کنند، یعنی جدول در مورد هر فعالیت نرمال (عبارات T-SQL) قفل می‌شود، اما چندین عمل bcp یا Bulk Insert را می‌توان در همان زمان انجام داد. این قفل فقط با Sch-S و سایر قفل هایBU سازگار است.

7- سطوح جداسازی (Isolation Level):
7-1- Read Committed (وضعیت پیش فرض): با Read Committed همه Shared Lockهای ایجاد شده، به محض اینکه عبارت ایجاد کننده آنها تکمیل شود، به طور خودکار آزاد می‌شوند. به طور خلاصه قفل‌های مرتبط با عبارت Select به محض تکمیل عبارت Select آزاد می‌شوند و SQL Server منتظر پایان تراکنش نمی‌ماند. اگر تراکنش پرس و جویی را انجام می‌دهد که داده‌ها را اصلاح می‌کند (Insert، Delete و Update) قفل‌ها برای مدت تراکنش نگه داشته می‌شوند.
با این سطح پیش فرض، می‌توانید مطمئن شوید جامعیت کافی برای پیشگیری از Dirty Read دارید، اما همچنان Phantoms و Non-Repeatable Read می‌تواند روی دهد.

7-2- Read Uncommitted: خطرناک‌ترین گزینه از میان تمامی گزینه‌ها است، اما بالاترین عملکرد را به لحاظ سرعت دارد. در واقع با این تنظیم سطح تجربه همه مسائل متعدد هم زمانی مانند Dirty Read امکان پذیر است. در واقع با تنظیم این سطح به SQL Server اعلام می‌کنیم هیچ قفلی را تنظیم نکرده و به هیچ قفلی اعتنا نکند، بنابراین هیچ تراکنش دیگری را مسدود نمی‌کنیم.
می‌توانید همین اثر Read Uncommitted را با اضافه کردن نکته بهینه ساز **NOLOCK** در پرس و جوها بدست آورید.

7-3- Repeatable Read: سطح جداسازی را تا حدودی افزایش می‌دهد و سطح اضافی محافظت همزمانی را با پیشگیری از Dirty Read و همچنین Non-Repeatable Read فراهم می‌کند.
پیشگیری از Non-Repeatable Read بسیار مفید است اما حتی نگه داشتن Shared Lock تا زمان پایان تراکنش می‌تواند دسترسی کاربران به اشیا را مسدود کند، بنابراین به بهره وری لطمه وارد می‌کند.
نکته بهینه ساز برای این سطح **REPEATEABLEREAD** است.

7-4- Serializable: این سطح از تمام مسائل هم زمانی پیشگیری می‌کند به جز برای Lost Update.
این تنظیم سطح به واقع بالاترین سطح آنچه را که سازگاری نامیده می‌شود، برای پایگاه داده فراهم می‌کند. در واقع فرآیند بهنگام رسانی برای کاربران مختلف به طور یکسان عمل می‌کند به گونه ای که اگر همه کاربران یک تراکنش را در یک زمان اجرا می‌کردند، این گونه می‌شد « پردازش امور به طور سریالی».
با استفاده از نکته بهینه ساز **SERIALIZABLE** یا **HOLDLOCK** در پرس و جو شبیه سازی می‌شود.

7-5- Snapshot: جدترین سطح جداسازی است که در نسخه 2005 اضافه شد، که شبیه ترکیبی از Read Committed و Read Uncommitted است. به طور پیش فرض در دسترس نیست، در صورتی در دسترس است که گزینه ALLOW_SNAPSHOT_ISOLATION برای بانک اطلاعاتی فعال شده باشد.(برای هر بانک اطلاعاتی موجود در تراکنش)
Snapshot مشابه Read Uncommitted هیچ قفلی ایجاد نمی‌کند. تفاوت اصلی آن‌ها در این است که تغییرات صورت گرفته در بانک اطلاعاتی را در زمان‌های متفاوت تشخیص می‌دهند. هر تغییر در بانک اطلاعاتی بدون توجه به زمان یا Commit شدن آن، توسط پرس و جو هایی که سطح جداسازی Read Uncommitted را اجرا می‌کنند، دیده می‌شود. با Snapshot فقط تغییراتی که قبل از شروع تراکنش، Commit شده اند، مشاهده می‌شود.
از شروع تراکنش Snapshot، تمامی داده‌ها دقیقاً مشاهده می‌شوند، زیرا در شروع تراکنش Commit شده اند.
نکته: در حالی که Snapshot توجهی به قفل‌ها و تنظیمات آنها ندارد، یک حالت خاص وجود دارد. چنانچه هنگام انجام Snapshot یک عمل Rollback (بازیافت) بانک اطلاعاتی در جریان باشد، تراکنش Snapshot قفل‌های خاصی را برای عمل کردن به عنوان یک مکان نگهدار و سپس انتظار برای تکمیل Rollback تنظیم می‌کند. به محض تکمیل Rollback، قفل حذف شده و Snapshot به طور طبیعی به جلو حرکت خواهد کرد.

Isolation level	Dirty read	Non-Repeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

نظرات خوانندگان

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۲/۰۵ ۱۲:۳۷

سلام ، خسته نباشید. ممنون از زحماتی که برای این مقاله کشیدید.
من یه سوال داشتم درباره تراکنش (Transaction) : من توی پروژه ام از sqlTransaction و isolation سریالایز استفاده کردم در این بلاک، 9 جدول من مورد عملیات درج و یا ویرایش قرار می‌گیره (و البته دو وب سرویس نیز صدا زده می‌شه) و معمولا 2 یا 3 ثانیه برای ثبت این تراکنش زمان صرف میشه . تا اینجا مشکلی نیست ولی اگر چند کاربر مثلا 5 نفر این عملیات انجام بدن در بعضی از مواقع (به ندرت) به ترتیب عملیات برای هر نفر انجام میشه (همین توقع از سریالایز می‌ره). ولی در اکثر مواقع خطا deadlock می‌ده و یا خطا timeout و همچنین جاهای مختلف برنامه که به یکی از این 9 جدول select می‌شه به انها هم همین خطاها رو می‌ده . لطفا در صورت امکان راه حلی برای مشکل من بگین . ممنون

نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۵ ۱۵:۴۵

با سلام و سپاس از محبت تان، چند نکته به ذهن من میرسد، که شاید راهگشا باشد:
- تعداد دستورات درون بلاک Transaction تا جایی که میسر است، کمینه باشد.
- عملیات مربوط به وب سرویس، خارج از Transaction انجام گیرد. (به عنوان یک نکته در کار با Transactionباید عملیات Input و Output خارج از Transaction انجام گیرد.)
- طبیعتا هر چه درجه Isolation بالاتر باشد، Lock سختگیرانه‌تر برخورد می‌کند.
- برای مشکل بن بست، اگر می‌توانید ترتیب در اختیار گرفتن جداول را (پس از شناسائی محل بن بست) تغییر دهید و یا از جداول Temp استفاده کنید.
- چنانچه در قسمت هایی از برنامه همانطور که اشاره کردید فقط نیاز به خواندن مقادیر جداول دارید، از بهینه ساز پرس و جوی nolock استفاده کنید. برای مثال:

```
select * from tbl with (NOLOCK)
```

عنوان: **معرفی و استفاده از DDL Triggers در SQL Server**
نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۳ ۱۱:۵۰
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, Database

استفاده از DDL Trigger امکان ایجاد Trigger برای عملیات DDL(Data Definition Language) از SQL Server 2005 فراهم گردید. عملیاتی مانند ایجاد یک جدول جدید در بانک اطلاعاتی، اضافه شدن یک Login جدید و یا ایجاد یک بانک اطلاعاتی جدید را به وسیله این نوع Triggerها می‌توان کنترل نمود. در حقیقت DDL Trigger به شما اجازه می‌دهد که از تاثیر تعدادی از دستورات DDL جلوگیری کنید. بدین ترتیب که تقریباً هر دستور DDL به طور خودکار، تراکنشی (Transactional) اجرا می‌شود . می‌توان با دستور ROLLBACK TRANSACTION اجرای دستور DDL را لغو نمود. توجه شود همه دستورات DDL به صورت تراکنشی اجرا نمی‌شوند، به عنوان مثال دستور ALTER DATABASE ممکن است Database را تغییر دهد. در این صورت ساختار فایلی Database را تغییر می‌دهد، از آنجائی که سیستم عامل ویندوز به صورت تراکنشی عمل نمی‌کند بنابراین شما نمی‌توانید این عمل فایل سیستمی را لغو نمائید. به هر حال شما می‌توانید Trigger را با ALTER DATABASE فعال (fire) کنید برای عملیات Auditing، ولی نمی‌توان از انجام عمل ALTER DATABASE جلوگیری کرد.
برای نمونه می‌خواهیم از حذف و یا تغییر جداول یک بانک اطلاعاتی که به صورت عملیاتی در حال سرویس دهی است جلوگیری کنیم، برای اینکار از دستورهای زیر استفاده می‌کنیم:

```
create trigger Prevent_AlterDrop
on database
for drop_table, alter_table
as
  print 'table can not be dropped or altered'
  rollback transaction
```

از عبارت ON برای مشخص کردن محدوده Trigger در سطح SQL Instance (در این صورت ON All SERVER نوشته می‌شود) و یا در سطح Database (در این حالت ON DATABASE نوشته می‌شود) استفاده می‌شود و از عبارت FOR برای مشخص کردن رویداد یا گروه رویدادی که سبب فراخوانی Trigger می‌شود، استفاده خواهد شد.

1- معرفی تابع EVENTDATA() این تابع، یک تابع سیستمی مهم است که در DDL Trigger استفاده می‌شود. در حالیکه DDL Trigger در هر سطحی فعال (fire) شود تابع سیستمی EVENTDATA() فراخوانی (raise) می‌شود. خروجی تابع در قالب XML است . می‌توان اطلاعات را از تابع EVENTDATA دریافت کرد و آنها را در یک جدول با فیلدی از جنس XML و یا با استفاده از XPath Query ثبت کرد (Logging). عناصر کلیدی (Key Elements) تابع EVENTDATA به شرح زیر است:

- **EventType:** نوع رویدادی که باعث فراخوانی Trigger شده است.
- **PostTime:** زمانی که رویداد رخ می‌دهد.
- **SPID:** SPID کاربری که باعث ایجاد رویداد شده است.
- **ServerName:** نام SQL Instance که رویداد در آن رخ داده است.
- **LoginName:** نام Login که عمل مربوط به وقوع رویداد را اجرا می‌کند.
- **UserName:** نام User که عمل مربوط به وقوع رویداد را اجرا می‌کند.
- **DatabaseName:** نام Database که رویداد در آن رخ می‌دهد.
- **ObjectType:** نوع Object که اصلاح، حذف و یا ایجاد شده است.
- **ObjectName:** نام Object که اصلاح، حذف و یا ایجاد شده است.
- **TSQLCommand:** دستور T-SQL که اجرا شده و باعث اجرا شدن Trigger شده است.

2- بررسی یک سناریو نمونه
برای نمونه در دستورات زیر جدولی با نام ddl_log

```
CREATE TABLE ddl_log
  (
     EventType nvarchar(100),
 PostTime datetime,
```

```
 SPID nvarchar(100),
 ServerName nvarchar(100),
 LoginName nvarchar(100),
 UserName  nvarchar(100),
 DatabaseName  nvarchar(100),
 ObjectName  nvarchar(100),
 ObjectType  nvarchar(100),
 DefaultSchema nvarchar(100),
 [SID]  nvarchar(100),
 TSQLCommand nvarchar(2000));
```

و یک Trigger با نام log برای رویدادهایی که در سطح Database رخ می‌دهد، ایجاد می‌کنیم.

```
CREATE TRIGGER [Log] ON DATABASE
 FOR DDL_DATABASE_LEVEL_EVENTS
 AS
 DECLARE @data XML
 SET @data = EVENTDATA()
 INSERT INTO ddl_log
 VALUES (
   @data.value('(/EVENT_INSTANCE/EventType)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/PostTime)[1]',   'datetime'),
   @data.value('(/EVENT_INSTANCE/SPID)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/ServerName)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/LoginName)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/UserName)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/DatabaseName)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/ObjectName)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/ObjectType)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/DefaultSchema)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/SID)[1]',   'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/TSQLCommand)[1]',   'nvarchar(2000)'));
```

نمونه ای از مقادیر ذخیره شده در جدول ddl_log به شکل زیر خواهد بود:

EventType	PostTime	SPID	ServerName	LoginName	UserName	DatabaseName	ObjectName
CREATE_TABLE	11/7/2010 10:36:26 AM	55	1936-603102405\SQL2008	sa	dbo	Test	Tmp_TestTable
ALTER_TABLE	11/7/2010 10:36:26 AM	55	1936-603102405\SQL2008	sa	dbo	Test	Tmp_TestTable
DROP_TABLE	11/7/2010 10:36:26 AM	55	1936-603102405\SQL2008	sa	dbo	Test	TestTable
RENAME	11/7/2010 10:36:26 AM	55	1936-603102405\SQL2008	sa	dbo	Test	Tmp_TestTable
DROP_TABLE	11/7/2010 10:38:26 AM	52	1936-603102405\SQL2008	sa	dbo	Test	TestTable

DefaultSchema	SID	TSQLCommand
NULL	NULL	CREATE TABLE dbo.Tmp_TestTable (a nvarchar(50) NULL) ON [PRIMARY]
NULL	NULL	ALTER TABLE dbo.Tmp_TestTable SET (LOCK_ESCALATION = TABLE)
NULL	NULL	DROP TABLE dbo.TestTable
NULL	NULL	EXECUTE sp_rename N'dbo.Tmp_TestTable', N'TestTable', 'OBJECT'
NULL	NULL	DROP TABLE TestTable ;

3- ملاحظات

در صورت فعال شدن Trigger می‌توان برخی موارد مانند محدودیت زمانی، کاربر اجرا کننده و ... را اضافه نمود. برای مثال در دستور زیر اجازه تغییرات در این زمان (بین 7:00 A.M تا 8:00 P.M) امکان پذیر نیست و در صورت اقدام پیغام خطا دریافت می‌کنید و دستورات Create لغو خواهند شد و اگر خارج از زمان فوق دستورات DDL را اجرا کنید دستورات به طور موفقیت آمیز اجرا می‌شود و البته تغییرات نیز Log می‌شوند.

```
IF DATEPART(hh,GETDATE()) > 7 AND DATEPART(hh,GETDATE()) < 20
BEGIN
    RAISERROR ('You can only perform this change between 8PM and 7AM. Please tr
    this change again or contact Production support for an override.', 16, 1)
ROLLBACK
END
```

این Trigger تاثیرات کمی بر روی کارایی دارد به این دلیل که معمولاً رویدادهای DDL به ندرت رخ می‌دهد. می‌توانید هنگامی که قصد دارید دستورات DDL را اجرا کنید موقتاً Trigger را با دستورات زیر غیر فعال نمائید:

```
DISABLE TRIGGER ALL ON DATABASE
DISABLE TRIGGER ALL ON ALL SERVER
```

پس از Overrdie کردن می‌توانید مجدداً Trigger را فعال کنید:

```
ENABLE TRIGGER ALL ON DATABASE
ENABLE TRIGGER ALL ON ALL SERVER
```

4- معرفی DDL Event Groups:

برای مشاهده جزئیات بیشتر می‌توانید به این لینک مراجعه کنید.

Server Level

DDL_SERVER_LEVEL_EVENTS

DDL_LINKED_SERVER_EVENTS

DDL_LINKED_SERVER_LOGIN_EVENTS

DDL_REMOTE_SERVER_EVENTS

DDL_EXTENDED_PROCEDURE_EVENTS

DDL_MESSAGE_EVENTS

DDL_ENDPOINT_EVENTS

DDL_SERVER_SECURITY_EVENTS

DDL_LOGIN_EVENTS

DDL_GDR_SERVER_EVENTS

DDL_AUTHORIZATION_SERVER_EVENT Database Level

Database Level

DDL_DATABASE_LEVEL_EVENTS

DDL_TABLE_VIEW_EVENTS

DDL_TABLE-EVENTS

DDL_VIEW_EVENTS

DDL_INDEX_EVENTS

DDL_STATISTICS_EVENTS

DDL_DATABASE_SECURITY_EVENTS

DDL_CERTIFICATE_EVENTS

DDL_USER_EVENTS

DDL_ROLE_EVENTS

DDL_APPLICATION_ROLE_EVENTS

DDL_SCHEMA_EVENTS

DDL_GDR_DATABASE_EVENTS

DDL_AUTHORIZATION_DATABASE_EVENTS

DDL_FUNCTION_EVENTS

DDL_PROCEDUER_EVENTS

DDL_TRIGGER_EVENTS

DDL_PARTITION_EVENTS

DDL_PARTITION_FUNCTION_EVENTS

DDL_PARTITION_SCHEME_EVENTS

DDL_SSB_EVENTS

DDL_MESSAGE_TYPE_EVENTS

DDL_CONTRACT_EVENTS

DDL_QUEUE_EVENTS

DDL_SERVER_EVENTS

DDL_ROUTE_EVENTS

DDL_REMOTE_SERVICE_BINDING_EVENTS

DDL_XML_SCHEMA_COLEECTION_EVENTS

DDL_FULLTEXT_CATALOG_EVENTS

DDL_DEFAULT_EVENTS

DDL_EXTENDED_PROPERTY_EVENTS

DDL_PLAN_GUIDE_EVENTS

DDL_RULE_EVENTS

DDL_SYNONYM_EVENTS

DDL_EVENT_NOTIFICATION_EVENTS

DDL_ASSEMBLY_EVENTS

DDL_TYPE_EVENTS

نظرات خوانندگان

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۶/۲۵ ۲۰:۴۱

با سلام
اول از همه تشکر میکنم از شما دوست عزیز بابت زحماتی که کشیدید.
سئوال بنده اینه اگر بخواهیم خودمون یک نوع TRIGGER جدید به Database اضافه کنیم به چه صورت امکان پذیره؟
به طور مثال من می‌خوام یک تریگر فقط برای کلیه فعالیت‌ها در حوزه‌ی بکاپ و ریستور بنویسم.
مثل تریگر Alter table یا Drop_table که ماهیتشون از قبل تعریف شده است.

با تشکر

نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۷/۰۲ ۱۰:۳۴

با سلام و احترام؛ همانطور که در متن به عرض رسانده شده:
" از عبارت ON برای مشخص کردن محدوده Trigger در سطح SQL Instance (در این صورت ON All SERVER نوشته می‌شود) و یا در سطح Database (در این حالت ON DATABASE نوشته می‌شود) استفاده می‌شود و از عبارت FOR برای مشخص کردن رویداد یا گروه رویدادی که سبب فراخوانی Trigger می‌شود، استفاده خواهد شد. "
در خصوص مثالی که اشاره کردید، به نظرم می‌رسد از Trigger برای این منظور استفاده نمی‌شود (در حوزه‌ی بکاپ و ریستور)، شاید اگر قصدتان به منظور ثبت log و ... بایست از Auditing استفاده کنید. به این منظور در Auditing با توجه به جدول زیر می‌توان اقدام به ثبت موارد نمود:

Server-Level Audit Action Groups

Action group name	Event Class	Description
BACKUP_RESTORE_GROUP	Audit Backup/Restore	یک دستور Backup یا Restore صادر شود

به طور مختصر Auditing به شرح زیر است:

بررسی SQL Server Audit

بازبینی (Auditing) شامل پیگیری و ثبت رویدادهایی است که در سطح SQL Instance و یا Databaseهای روی یک سیستم اتفاق می‌افتد. چندین سطح برای Auditing در SQL Server وجود دارد که به صلاحدید و نیازمندی‌های نصب شما وابسته است. شما می‌توانید گروه اقدامات بازبینی سرویس دهنده (server audit action groups) را به ازای هر SQL Instance و گروه اقدامات بازبینی بانک اطلاعاتی (database audit action groups) را به ازای هر بانک اطلاعاتی ثبت کنید. رویداد Audit هر زمان عملی که مورد رسیدگی قرار گرفته اتفاق افتد، رخ می‌دهد.
تا پیش از SQL SERVER 2008، شما باید از خصیصه‌های متعددی برای انجام یک مجموعه کامل بازبینی (Auditing) برای نمونه DDL Trigger، DML Trigger و SQL Trace، بر روی یک SQL Instance استفاده می‌کردید.
SQL SERVER 2008، همه قابلیت‌های Auditing را روی یک audit specification ترکیب می‌کند. Audit Specification با تعریف یک شی بازبینی (audit object) در سطح سرویس دهنده برای ثبت (logging) یک دنباله بازبینی (audit trial) آغاز می‌شود.
توجه شود که بایست یک شیء بازبینی ایجاد کنید پیش از اینکه یک Server Audit Specification و یا Database Audit Specification ایجاد کنید.
Server Audit Specification، گروه اقدامات در سطح سرویس دهنده را جمع آوری می‌کند که با رویدادهای وسیعی فعال می‌شوند، این گروه اقدامات تحت عنوان Server-Level Audit Action Groups تشریح شده اند. شما می‌توانید یک Server Audit Specification را به ازای هر Audit ایجاد کنید چرا که هر دو در محدوده یک SQL Instance ایجاد می‌شوند.
Database Audit Specification، گروه اقدامات در سطح بانک اطلاعاتی را جمع آوری می‌کند که با رویدادهای وسیعی فعال می‌شود. این گروه اقدامات تحت عنوان‌های Database-Level Audit Action Groups و Database-Level Audit Actions تشریح

شده اند.می توانید یک Database Audit Specification را به ازای هر Audit در بانک اطلاعاتی SQL Server ایجاد کنید.
همچنین می‌توانید هر گروه اقدامات بازبینی(audit action groups) یا رویدادهای بازبینی(audit events) را به یک Database Audit Specification اضافه کنید. گروه اقدامات بازبینی، گروه اقدامات از پیش تعریف شده ای هستند و رویدادهای بازبینی اقدامات تجزیه ناپذیری هستند که توسط موتور بانک اطلاعاتی مورد رسیدگی قرار می‌گیرند، هر دو در محدوده بانک اطلاعاتی (Database) هستند. این اقدامات برای Audit فرستاده می‌شوند تا در Target (که می‌تواند یک فایل، Windows Security Log و یا Windows Application Log باشد) ذخیره شوند. برای نوشتن در Windows Security Log لازم است که Service Account سرویس دهنده شما به Policy، Generate security audits اضافه شده باشد، به صورت پیش فرض Local System، Local Service و Network Service بخشی از این Policy می‌باشند. پس از اینکه Audit را ایجاد و فعال کردید، Target ورودی‌ها را دریافت خواهد کرد.

Server-Level Audit Action Groups (گروه اقدامات بازبینی در سطح سرویس دهنده)

این گروه اقدامات به گروه رویداد Security Audit شبیه هستند. به طور خلاصه این گروه اقدامات، اقداماتی را که در یک SQL Instance شامل می‌شوند، در بر می‌گیرد. برای مثال اگر گروه اقدام مناسب با Server Audit Specification اضافه شده باشد هر شیء در هر Schema که مورد دستیابی قرار می‌گیرد، ثبت می‌شود. اقدامات در سطح سرویس دهنده به شما اجازه نمی‌دهد که جزئیات اقدامات در سطح بانک اطلاعاتی را فیلتر کنید. یک بازبینی در سطح بانک اطلاعاتی برای انجام به جزئیات دقیق فیلتر کردن نیاز دارد، برای مثال اجرای دستور Select روی جدول Customers برای login هایی که در گروه Employee هستند.

Database-Level Audit Action Groups (گروه اقدامات بازبینی در سطح بانک اطلاعاتی)

این گروه اعمال به کلاس‌های رویداد Security Audit شبیه هستند.

Database-Level Audit Actions

اقدامات در سطح بانک اطلاعاتی، اقدامات بازبینی خاصی را به طور مستقیم روی Database، Schema و اشیاء Schema (از قبیل جداول، View ها، رویه‌های ذخیره شده، توابع و ...) فراهم می‌کند. این اقدامات برای فیلدها (Columns) صدق نمی‌کنند.

Audit-Level Audit Action Groups

شما می‌توانید اقداماتی را که در فرآیند Auditing هستند، بازبینی کنید که می‌تواند در محدوده سرویس دهنده یا بانک اطلاعاتی باشد. در محدوده بانک اطلاعاتی تنها برای database audit specification رخ می‌دهد.
جهت بررسی بیشتر به این لینک مراجعه شود.

نویسنده: MF
تاریخ: ۱۳۹۳/۰۷/۰۲ ۱۷:۷

DDL Triggerها , حالت Befor ندارند(انجام یک سری کارها قبل از عملیات درج,حذف و ...).برای حل این مساله باید از INSTEAD OF در تریگر استفاده نمود ؟

نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۷/۰۳ ۱۲:۱

مواردی که اشاره کردید مربوط به DML Triggers می‌باشند.
برای اطلاعات بیشتر به این لینک Understanding DDL Triggers vs. DML Triggers مراجعه شود.

نویسنده: مجید فاضلی
تاریخ: ۱۳۹۳/۰۷/۰۶ ۱۲:۴۶

باید از از حالت INSTEAD OF استفاده کنیم در **DML** Trigger ای که قراره نوشته بشه.
می‌توانیم در یک جدول از دیتابیس مان بر اساس یک شرط خاص, عملیات Insert,Delete,Update را مدیریت کنیم.
بعنوان مثال در قطعه کد زیر ما قبل از عملیات Insert در جدول tblTest چک میکنیم که اگر مقدار ستون FirstName برابر با null بود عملیات Insert آن رکورد در دیتابیس لغو شود.

```
ALTER TRIGGER [dbo].[Prevent_Befor_Insert_Null]
   ON  [dbTest].[dbo].[tblTest]
   INSTEAD OF INSERT
AS
BEGIN
```

```
SET NOCOUNT ON
IF OBJECT_ID(N'dbTest.dbo.tblTest.FirstName') is null
BEGIN
DECLARE @Id int
SET @Id = (select Id from inserted)
RAISERROR ('16,1,'مقدار فیلد نام نباید خالی باشد)
ROLLBACK
END
END
```

از دو طریق می‌توان به مقادیر فیلدهای رکورد جاری دسترس داشت:
-1 استفاده از OBJECT_ID و ذکر نام فیلد مورد نظر
-2 گرفتن فیلد مورد نظر از جدول INSERTED یا DELETED
DML Triggerها دارای دو جدول خاص بنام‌های INSERTED و DELETED هستند که توسط خود SQL Server مدیریت می‌شوند.در حقیقت در پشت صحنه, ما با این دو جدول در هنگام تغییر مقادیر داده‌های جداول دیتابیس کار می‌کنیم و نمی‌توانیم بصورت مستقیم داده‌های جداول موجود در دیتا بیس مان را تغییر دهیم.
جدول INSERTED و DELETED حاوی رکورد جاری است که تحت تاثیر عمل درج, ویرایش و حذف در دیتابیس قرار گرفته است.
اطلاعات بیشتر در اینجا و اینجا

نویسنده: محمد
تاریخ: ۱۳۹۳/۰۷/۲۸ ۰:۱۵

سلام؛ تشکر از توضیحات شما. اجازه بدید من طور دیگری سئوالم رو مطرح کنم. به طور مثال ما برای کار با تاریخ شمسی در SQl چندین روش پیش رو داریم که وارد جزئیات آن نمیشوم ولی یکی از این روش‌ها که به خوبی جواب میدهد استفاده از CLR است که ما با توسط این قابلیت می‌توانیم یک نوع دیتا تایپ جدید ، با ماهیت جدید در اس کیو ال اضافه کنیم. حالا منظور بنده این است که آیا برای تریگرها هم می‌شود این کار را انجام داد یا خیر؟ مثلا توسط CLR یا هر روش دیگری که وجود دارد ، ما بیاییم و یک نوع تریگر کاملا جدید و Customize شده برای خودمان درست کنیم. به طور مثال : زمانی که کاربر از دیتابیس بخواهد بکاپ تهیه کند یا آن را ریستور کند ، یکسری فعالیتها به آن فعالیت اضافه شود یا در راستای آن انجام شود. با تشکر

عنوان: **بررسی دو نکته (ترفند) کاربردی در SQL Server**
نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۳ ۱۶:۲۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, Database

1- اندازه گیری تعداد Transactionها در واحد زمان روی یک Database خاص در SQL Server جهت بدست آوردن تعداد Transactionها در واحد زمان(Transactions Per Second) روی یک Database خاص در یک سیستم عملیاتی، جهت ارتقاء سخت افزاری ، تست فشار و ... می‌توانید از یک DMV با نام sys.dm_os_performance_counters به طریق زیر استفاده نمائید:

```
declare @cntr_value bigint

Select @cntr_value=cntr_value
from sys.dm_os_performance_counters
where instance_name='AdventureWorks' and
counter_name='Write Transactions/sec'

/* ایجاد یک تاخیر مثلاً یک ثانیه */
waitfor delay '00:00:01'

Select cntr_value -@cntr_value
from sys.dm_os_performance_counters
where instance_name='AdventureWorks' and
counter_name='Write Transactions/sec'
```

View معرفی شده تمامی شمارنده‌های عملکردی را برای یک Instance خاص شامل می‌شود، ستون instance_name برابر نام بانک اطلاعاتی مورد نظر می‌باشد.

2- sys.sp_MSforeachtable

از رویه‌های ذخیره شده UnDocumented در SQL Server می‌باشد و این قابلیت را دارا است که برای هر یک از جداول موجود در یک بانک اطلاعاتی، یک رویه‌ای را اجرا کند. برای مثال با استفاده از دستور زیر، می‌توانید تعداد سطرها، اندازه‌ی داده‌ها و ایندکس‌های یک جدول را بدست آورید

```
EXEC sys.sp_MSforeachtable 'sp_spaceused ''?''';
```

به عنوان یک مثال کاربردی، با اجرای دستور زیر می‌توان جداول بانک اطلاعاتی مورد نظرتان را از لحاظ معیارهایی که پیشتر ذکر آن رفت، مورد بررسی قرار دهید.

```
USE [AdventureWorksDW2008R2]
GO

CREATE TABLE #TableSpaceUsed(
[name] [nvarchar](120) NULL,
[rows] [nvarchar](120) NULL,
[reserved] [nvarchar](120) NULL,
[data] [nvarchar](120) NULL,
[index_size] [nvarchar](120) NULL,
[unused] [nvarchar](120) NULL
) ON [PRIMARY]

Insert Into #TableSpaceUsed
EXEC sys.sp_MSforeachtable 'sp_spaceused ''?''';

Select * from #TableSpaceUsed
Order by CAST([rows] as int) desc

Drop table #TableSpaceUsed
```

خروجی مثال فوق به شکل زیر است.

	name	rows	reserved	data	index_size	unused
1	FactInternetSalesReason	64515	2208 KB	2048 KB	48 KB	112 KB
2	FactResellerSales	60855	25736 KB	11864 KB	13008 KB	864 KB
3	FactInternetSales	60398	20464 KB	8200 KB	11312 KB	952 KB
4	FactFinance	39409	2184 KB	2152 KB	8 KB	24 KB
5	DimCustomer	18484	8832 KB	7800 KB	904 KB	128 KB
6	FactAdditionalInternationalProductDescription	15168	4216 KB	4008 KB	64 KB	144 KB
7	FactCurrencyRate	14264	584 KB	472 KB	16 KB	96 KB
8	FactSurveyResponse	2727	352 KB	192 KB	160 KB	0 KB
9	ProspectiveBuyer	2059	728 KB	632 KB	88 KB	8 KB
10	DimDate	1188	232 KB	152 KB	48 KB	32 KB
11	DimReseller	701	280 KB	192 KB	88 KB	0 KB
12	DimGeography	655	136 KB	104 KB	16 KB	16 KB
13	DimProduct	606	6120 KB	5936 KB	88 KB	96 KB
14	DimEmployee	296	176 KB	120 KB	48 KB	8 KB
15	FactSalesQuota	163	48 KB	8 KB	40 KB	0 KB
16	FactCallCenter	120	48 KB	16 KB	32 KB	0 KB
17	DatabaseLog	115	408 KB	384 KB	8 KB	16 KB
18	DimCurrency	105	32 KB	8 KB	24 KB	0 KB
19	DimAccount	99	48 KB	16 KB	32 KB	0 KB
20	DimProductSubcategory	37	32 KB	8 KB	24 KB	0 KB
21	DimPromotion	16	32 KB	8 KB	24 KB	0 KB
22	DimOrganization	14	16 KB	8 KB	8 KB	0 KB
23	DimSalesTerritory	11	32 KB	8 KB	24 KB	0 KB
24	DimSalesReason	10	16 KB	8 KB	8 KB	0 KB
25	DimDepartmentGroup	7	16 KB	8 KB	8 KB	0 KB
26	DimProductCategory	4	32 KB	8 KB	24 KB	0 KB
27	DimScenario	3	16 KB	8 KB	8 KB	0 KB
28	AdventureWorksDWBuildVersion	1	72 KB	8 KB	8 KB	56 KB

نظرات خوانندگان

نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۶/۱۸ ۱۲:۵۴

یک مورد استفاده کاربردی، از رویه **sp_MSforeachtable** به هنگام تجمیع بانک‌های اطلاعاتی است (برای مثال تجمیع دو DB کوچک در یک DB بزرگتر). برای این منظور می‌توان در ابتدا تمامی Constraintهای جداول را موقتاً غیر فعال کرد، عملیات Load داده در جداول را انجام داد و مجدداً آنها را فعال نمود.
چنانچه از SSIS Package برای اینکار استفاده شود، با فرض اینکه در مرحله قبل Schema تمامی اشیاء بانک منتقل شده است، Control Flow این Package شامل Stepهای زیر میباشد:

گام اول - غیر فعال کردن تمامی Constraintهای جداول

برای این منظور از Object موسوم به Execute SQL Task به این صورت استفاده میشود که در بخش SQL Statement دستور زیر نوشته شود:

```
exec sp_MSforeachtable 'ALTER TABLE ? NOCHECK CONSTRAINT ALL'
```

گام دوم - انتقال داده‌ها به شکل Data Only

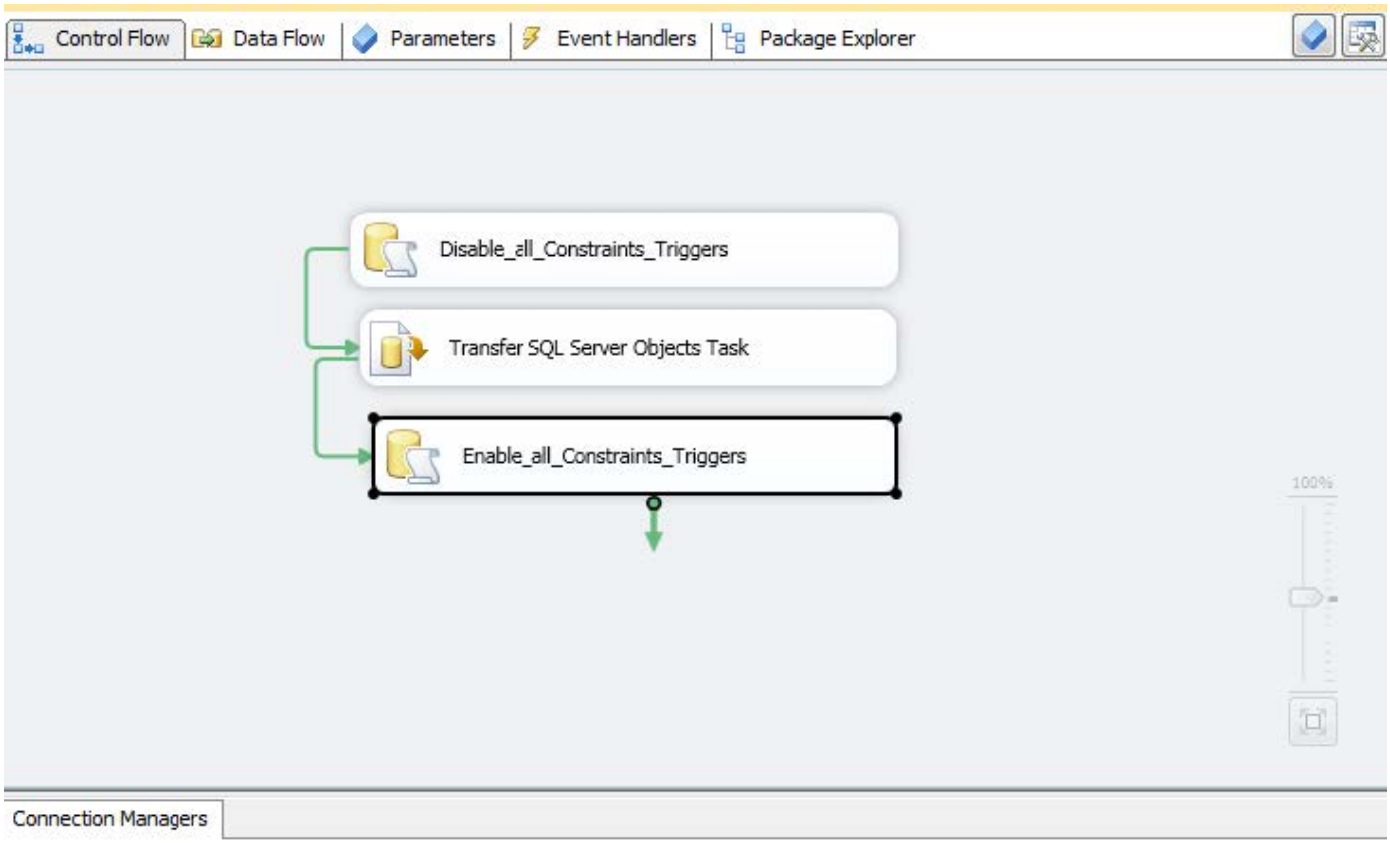
برای این منظور از Object موسوم به Transfer SQL Server Objects Task به صورت زیر استفاده میشود :
در بخش Object تغییرات زیر اعمال شود:
- در قسمت Destination ، پروپرتی CopyData با مقدار True و ExistingData با مقدار Append تنظیم شود.
- در قسمت Destination Copy & Option ، پروپرتی CopyAllTables در بخش ObjectsToCopy با مقدارTrue تنظیم شود.

گام سوم - فعال کردن مجدد Constraintهای جداول

برای این منظور از Object موسوم به Execute SQL Task به صورت زیر استفاده میشود :
در بخش SQL Statement دستور زیر نوشته شود:

```
exec sp_MSforeachtable 'ALTER TABLE ? CHECK CONSTRAINT ALL'
```

نکته: در صورت وجود Trigger برای جداول بانک اطلاعاتی نیز می‌توان به همین شکل عمل نمود. (ابتدا آنها را غیر فعال و مجدا فعال کرد)



برای مثال شکل SSIS Package فوق به صورت تصویر فوق است.

عنوان: **بررسی ابزار SQL Server Profiler**
نویسنده: محمد رجبی
تاریخ: ۱۳۹۳/۰۲/۰۶ ۱۰:۱۵
آدرس: www.dotnettips.info
گروه‌ها: SQL Server, Database, Performance

مقدمه Profiler یک ابزار گرافیکی برای ردیابی و نظارت بر کارآئی SQL Server است. امکان ردیابی اطلاعاتی در خصوص رویدادهای مختلف و ثبت این داده‌ها در یک فایل (با پسوند trc) یا جدول برای تحلیل‌های آتی نیز وجود دارد. برای اجرای این ابزار مراحل زیر را انجام دهید:

```
Start > Programs> Microsoft SQL Server > Performance Tools> SQL Server Profiler
```

و یا در محیط Management Studio از منوی Tools گزینه SQL Server Profiler را انتخاب نمائید.

1- اصطلاحات

1-1- رویداد (Event): یک رویداد، کاری است که توسط موتور بانک اطلاعاتی (Database Engine) انجام می‌شود. برای مثال هر یک از موارد زیر یک رویداد هستند.

- متصل شدن کاربران (login connections) قطع شدن ارتباط یک login
- اجرای دستورات T-SQL، شروع و پایان اجرای یک رویه، شروع و پایان یک دستور در طول اجرای یک رویه، اجرای رویه‌های دور Remote Procedure Call
- باز شدن یک Cursor
- بررسی و کنترل مجوزهای امنیتی

1-2- کلاس رویداد (Event Class): برای بکارگیری رویدادها در Profiler، از یک Event Class استفاده می‌کنیم. یک Event Class رویدادی است که قابلیت ردیابی دارد. برای مثال بررسی ورود و اتصال کاربران با استفاده از کلاس Audit Login قابل پیاده سازی است. هر یک از موارد زیر یک Event Class هستند.

- SQL:BatchCompleted
- Audit Login
- Audit Logout
- Lock: Acquired
- Lock: Released

1-3- گروه رویداد (Event Category): یک گروه رویداد شامل رویدادهایی است که به صورت مفهومی دسته بندی شده اند. برای مثال، کلیه رویدادهای مربوط به قفل‌ها از جمله Lock: Acquired (بدست آوردن قفل) و Lock: Released (رها کردن قفل) در گروه رویداد Locks قرار دارند.

1-4- ستون داده ای (Data Column): یک ستون داده ای، خصوصیت و جزئیات یک رویداد را شامل می‌شود. برای مثال در یک Trace که رویدادهای Lock: Acquired را نظارت می‌کند، ستون Binary Data شامل شناسه (ID) یک صفحه و یا یک سطر قفل شده است و یا اینکه ستون Duration مدت زمان اجرای یک رویه را نمایش می‌دهد.

1-5- الگو (Template): یک الگو، مشخص کننده تنظیمات پیش گزیده برای یک Trace است، این تنظیمات شامل رویدادهایی است که نیاز دارید بر آنها نظارت داشته باشید. هنگامیکه یک Trace براساس یک الگو اجرا شود، رویدادهای مشخص شده، نظارت می‌شوند و نتیجه به صورت یک فایل یا جدول قابل مشاهده خواهد بود.

1-6- ردیاب (Trace): یک Trace داده‌ها را براساس رویدادهای انتخاب شده، جمع آوری می‌کند. امکان اجرای بلافاصله یک Trace برای جمع آوری اطلاعات با توجه به رویدادهای انتخاب شده و ذخیره کردن آن برای اجرای آتی وجود دارد.

1-7- فیلتر (Filter): هنگامی که یک Trace یا الگو ایجاد می‌شود، امکان تعریف شرایطی برای فیلتر کردن داده‌های جمع آوری شده نیز وجود دارد. این کار باعث کاهش حجم داده‌های گزارش شده می‌شود. برای مثال اطلاعات مربوط به یک کاربر خاص جمع آوری می‌شود و یا اینکه رشد یک بانک اطلاعاتی مشخص بررسی می‌شود.

2- انتخاب الگو (Profiler Trace Templates) از آنجائیکه اصولاً انتخاب Eventهای مناسب، کار سخت و تخصصی می‌باشد برای راحتی کار تعدادی Templateهای آماده وجود دارد، برای مثال TSQL_Duration تاکیدش روی مدت انجام کار است و یا SP_Counts در مواردی که بخواهیم رویه‌های ذخیره شده را بهینه کنیم استفاده می‌شود در جدول زیر به شرح هر یک پرداخته شده است:

الگو	هدف
Blank	ایجاد یک Trace کلی
SP_Counts	ثبت اجرای هر رویه ذخیره شده برای تشخیص اینکه هر رویه چند بار اجرا شده است
Standard	ثبت آمارهای کارائی برای هر رویه ذخیره شده و Queryهای عادی SQL که اجرا می‌شوند و عملیات ورود و خروج هر Login (پیش فرض)
TSQL	ثبت یک لیست از همه رویه‌های ذخیره شده و Queryهای عادی SQL که اجرا می‌شوند ولی آمارهای کارائی را شامل نمی‌شود
TSQL_Duration	ثبت مدت زمان اجرای هر رویه ذخیره شده و هر Query عادی SQL
TSQL_Grouped	ثبت تمام loginها و logoutها در طول اجرای رویه‌های ذخیره شده و هر Query عادی SQL، شامل اطلاعاتی برای شناسائی برنامه و کاربری که درخواست را اجرا می‌کند
TSQL_Locks	ثبت اطلاعات انسداد (blocking) و بن بست (deadlock) از قبیل deadlock graphs، deadlock chains، blocked processes،... . این الگو همچنین درخواست‌های تمام رویه‌های ذخیره شده و تمامی دستورات هر رویه و هر Query عادی SQL را دریافت می‌کند
TSQL_Replay	ثبت اجرای رویه‌های ذخیره شده و Queryهای SQL در یک Instance و مهیا کردن امکان اجرای دوباره عملیات در سیستمی دیگر
TSQL_SPs	ثبت کارائی برای Queryهای SQL، رویه‌های ذخیره شده و تمامی دستورات درون یک رویه ذخیره شده و نیز عملیات ورود و خروج هر Login
Tuning	ثبت اطلاعات کارائی برای Queryهای عادی SQL و رویه‌های ذخیره شده و یا تمامی دستورات درون یک رویه ذخیره شده

3- انتخاب رویداد (SQL Trace Event Groups) رویدادها در 21 گروه رویداد دسته بندی می‌شوند که در جدول زیر لیست شده اند:

گروه رویداد	هدف
Broker	13 رویداد برای واسطه سرویس (Service Broker)
CLR	1 رویداد برای بارگذاری اسمبلی‌های CLR (Common Language Runtime)
Cursors	7 رویداد برای ایجاد، دستیابی و در اختیار گرفتن Cursor
Database	6 رویداد برای رشد/کاهش (grow/shrink) فایل های Data/Log همچنین تغییرات حالت انعکاس (Mirroring)
Deprecation	2 رویداد برای آگاه کردن وضعیت نابسامان درون یک SQL Instance
Errors and Warnings	16 رویداد برای خطاها، هشدارها و پیغام‌های اطلاعاتی که ثبت شده است
Full Text	3 رویداد برای پیگیری یک شاخص متنی کامل
Locks	9 رویداد برای بدست آوردن، رها کردن قفل و بن بست (Deadlock)
OLEDB	5 رویداد برای درخواست‌های توزیع شده و RPC (اجرای رویه‌های دور)
Objects	3 رویداد برای وقتی که یک شی ایجاد، تغییر یا حذف می‌شود
Performance	14 رویداد برای ثبت نقشه درخواست‌ها (Query Plan) برای استفاده نقشه راهنما (Plan Guide) به منظور بهینه سازی کارائی درخواست ها، همچنین این گروه رویداد در خواست‌های متنی کامل (full text) را ثبت می‌کند
Progress Report	10 رویداد برای ایجاد Online Index
Query Notifications	4 رویداد برای سرویس اطلاع رسان (Notification Service)
Scans	2 رویداد برای وقتی که یک جدول یا شاخص، پویش می‌شود
Security Audit	44 رویداد برای وقتی که مجوزی استفاده شود، جابجائی هویتی رخ دهد، تنظیمات امنیتی اشیائی تغییر کند،یک SQL Instance شروع و متوقف شود و یک Database جایگزین شود یا از آن پشتیبان گرفته شود
Server	3 رویداد برای Mount Tape، تغییر کردن حافظه سرور و بستن یک فایل Trace
Sessions	3 رویداد برای وقتی که Connectionها موجود هستند و یک Trace فعال می‌شود، همچنین یک Trigger و یک تابع دسته بندی(classification functions) مربوط به مدیریت منابع(resource governor) رخ دهد
Stored Procedures	12 رویداد برای اجرای یک رویه ذخیره شده و دستورات درون آن ، کامپایل مجدد و استفاده از حافظه نهانی (Cache)
Transactions	13 رویداد برای شروع، ذخیره ، تائید و لغو یک تراکنش
TSQL	9 رویداد برای اجرای Queryهای SQL و جستجوهای XQUERY (در داده‌های XML)

گروه رویداد	هدف
User Configurable	10 رویداد که شما می‌توانید پیکربندی کنید

به طور معمول بیشتر از گروه رویدادهای Locks، Performance، Security Audit، Stored Procedures و TSQL استفاده می‌شود.

4- انتخاب ستون‌های داده ای (Data Columns) اگرچه می‌توان همه‌ی 64 ستون داده ای ممکن را برای ردیابی انتخاب کرد ولیکن داده‌های Trace شما زمانی مفید خواهند بود که اطلاعات ضروری را ثبت کرده باشید. برای مثال شماره ترتیب تراکنش‌ها را، برای یک رویداد RPC:Completed می‌توانید برگردانید، اما همه رویه‌های ذخیره شده مقادیر را تغییر نمی‌دهند بنابراین شماره ترتیب تراکنش‌ها فضای بیهوده ای را مصرف می‌کند. بعلاوه همه ستون‌های داده ای برای تمامی رویدادهای Trace معتبر نیستند. برای مثال CPU، Write ، Read و Duration برای رویدادهای RPC:Starting و SQL:BatchStarting معتبر نیستند.
ApplicationName، NTUserName، LoginName، ClientProcessID، SPID، HostName، LoginSID، NTDomainName و SessionLoginName ، مشخص می‌کنند چه کسی و از چه منشاء دستور را اجرا کرده است.
ستون SessionLoginName معمولاً نام Login ای که از آن برای متصل شدن به SQL Instance استفاده شده است را نشان می‌دهد. در حالیکه ستون LoginName نام کاربری را که دستور را اجرا می‌کند نشان می‌دهد (EXECUTE AS). ستون ApplicationName خالی است مگر اینکه در ConnectionString برنامه کاربردیمان این خصوصیت (Property) مقداردهی شده باشد. ستون StartTime و EndTime زمان سرحدی برای هر رویداد را ثبت می‌کند این ستون‌ها بویژه در هنگامی که به عملیات Correlate نیاز دارید مفید هستند.

5- بررسی چند سناریو نمونه

• **یافتن درخواست هائی (Queries) که بدترین کارایی را دارا هستند.** برای ردیابی درخواست‌های ناکارا، از رویداد RPC:Completed از دسته Stored Procedure و رویداد SQL:BatchCompleted از دسته TSQL استفاده می‌شود.

• **نظارت بر کارایی رویه ها** برای ردیابی کارائی رویه ها، از رویدادهای SP:StmtCompleted، SP:Completed، SP:Starting و SP:StmtStaring از کلاس Stored Procedure و رویدادهای SQL:BatchCompleted ، SQL:BatchStarting از کلاس TSQL استفاده می‌شود.

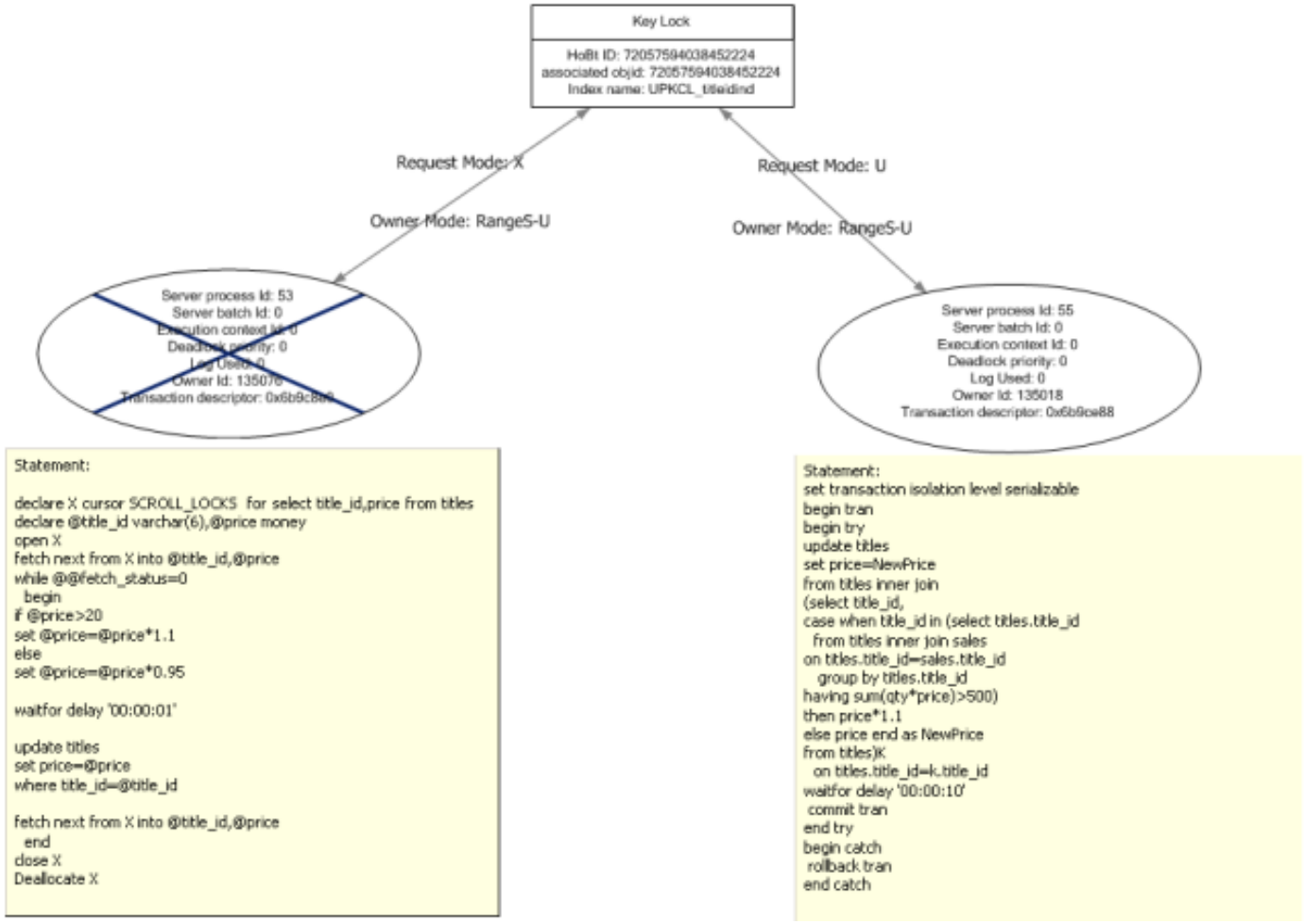
• **نظارت بر اجرای دستورات T-SQL توسط هر کاربر** برای ردیابی دستوراتی که توسط یک کاربر خاص اجرا می‌شود، نیاز به ایجاد یک Trace برای نظارت بر رویدادهای کلاس‌های Sessions، ExistingConnection و TSQL داریم همچنین لازم است نام کاربر در قسمت فیلتر و با استفاده از DBUserName مشخص شود.

• **اجرا دوباره ردیاب (Trace Replay)** این الگو معمولاً برای debugging استفاده می‌شود برای این منظور از الگوی Replay استفاده می‌شود. در ضمن امکان اجرای دوباره عملیات در سیستمی دیگر با استفاده از این الگو مهیا می‌شود.

• **ابزار Tuning Advisor (راهنمای تنظیم کارائی)** این ابزاری برای تحلیل کارائی یک یا چند بانک اطلاعاتی و تاثیر عملکرد آنها بر بار کاری (Workload) سرویس دهنده است. یک بار کاری مجموعه ای از دستورات T-SQL است که روی بانک اطلاعاتی اجرا می‌شود. بعد از تحلیل تاثیر بارکاری بر بانک اطلاعاتی، Tuning Advisor توصیه هائی برای اضافه کردن، حذف و یا تغییر طراحی فیزیکی ساختار بانک اطلاعاتی ارائه می‌دهد این تغییرات ساختاری شامل پیشنهاد برای تغییر ساختاری موارد Clustered Indexes، Nonclustered Indexes، Indexed View و Partitioning است.
برای ایجاد بارکاری می‌توان از یک ردیاب تهیه شده در SQL Profiler استفاده کرد برای این منظور از الگوی Tuning استفاده می‌شود و یا رویدادهای RPC:Completed، SQL:BatchCompleted و SP:StmtCompleted را ردیابی نمائید.

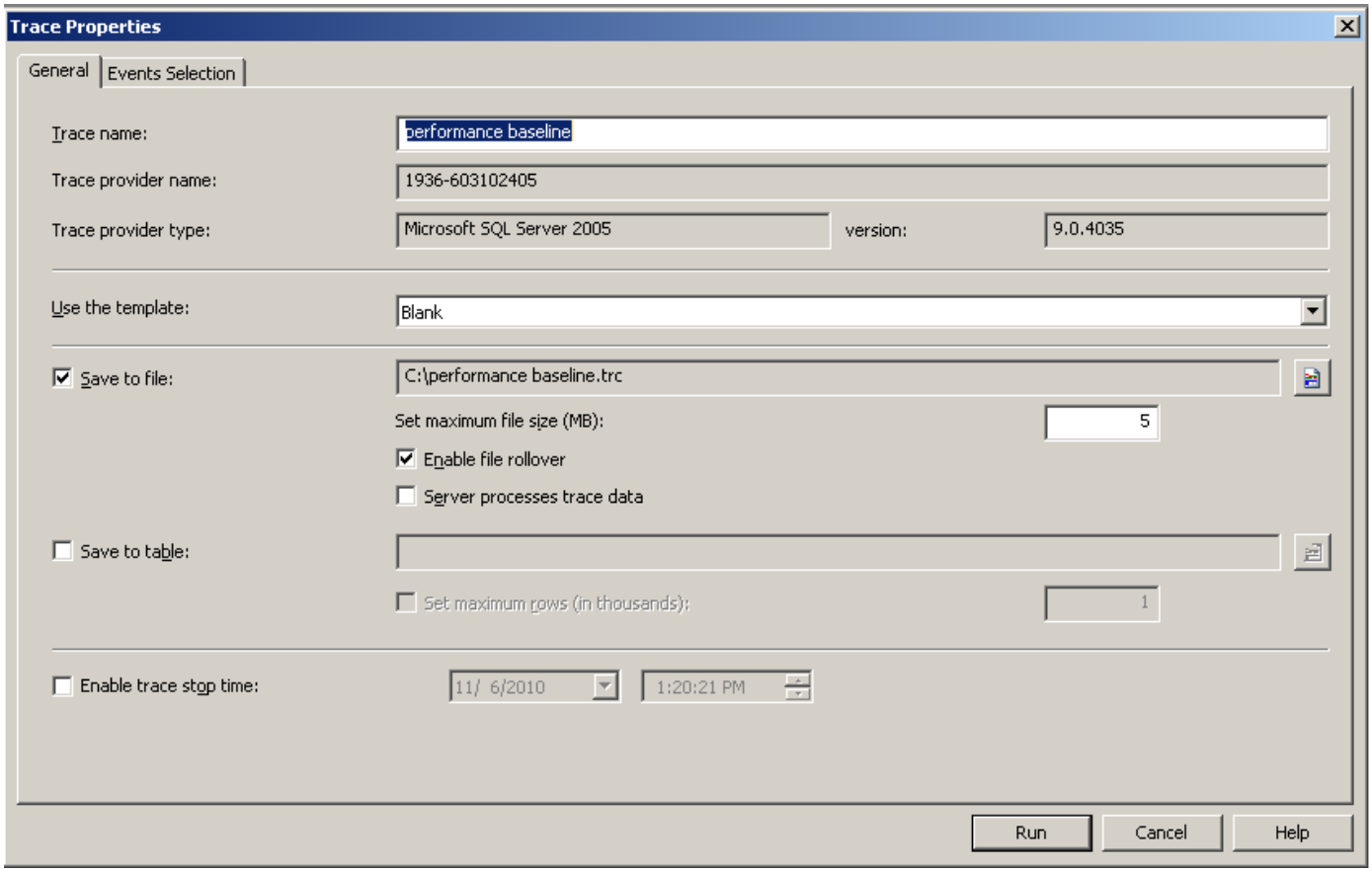
• **ترکیب ابزارهای نظارتی (Correlating Performance and Monitoring Data)** یک Trace برای ثبت اطلاعاتی که در یک SQL Instance رخ می‌دهد، استفاده می‌شود. System Monitor برای ثبت شمارنده‌های کارائی(performance counters) استفاده می‌شود و همچنین از منابع سخت افزاری و اجزای دیگر که روی سرور اجرا می‌شوند، تصاویری فراهم می‌کند. توجه شود که در مورد Correlating یک فایل ردیاب (trace file) و یک Counter Log (ابزار Performance)، ستون داده ای StartTime و EndTime باید انتخاب شود، برای این کار از منوی File گزینه Import Performance Data انتخاب می‌شود.

• **جستجوی علت رخ دادن یک بن بست** برای ردیابی علت رخ دادن یک بن بست، از رویدادهای RPC:Starting، SQLBatchStarting از دسته Stored Procedure و رویدادهای Lock:Deadlock، Deadlock graph و Lock:Deadlock Chain از دسته Locks استفاده می‌شود. (در صورتی که نیاز به یک ارائه گرافیکی دارید از Deadlock graph استفاده نمائید، خروجی مطابق تصویر زیر می‌شود).



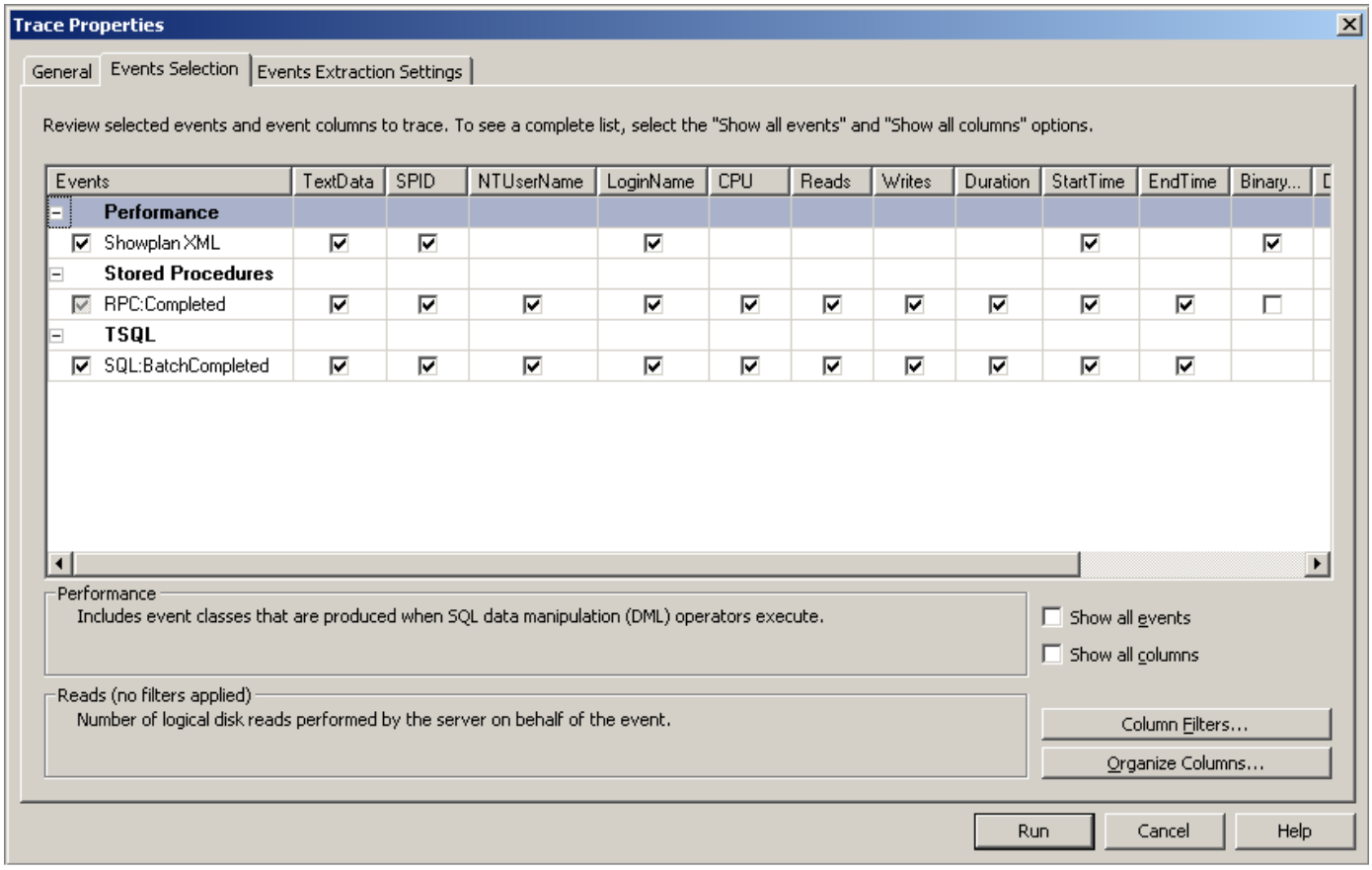
5-1- ایجاد یک Trace 1- Profiler را اجرا کنید از منوی File گزینه New Trace را انتخاب کنید و به SQL Instance مورد نظرتان متصل شوید.

2- مطابق تصویر زیر برای Trace یک نام و الگو و تنظیمات ذخیره سازی فایل را مشخص کنید.

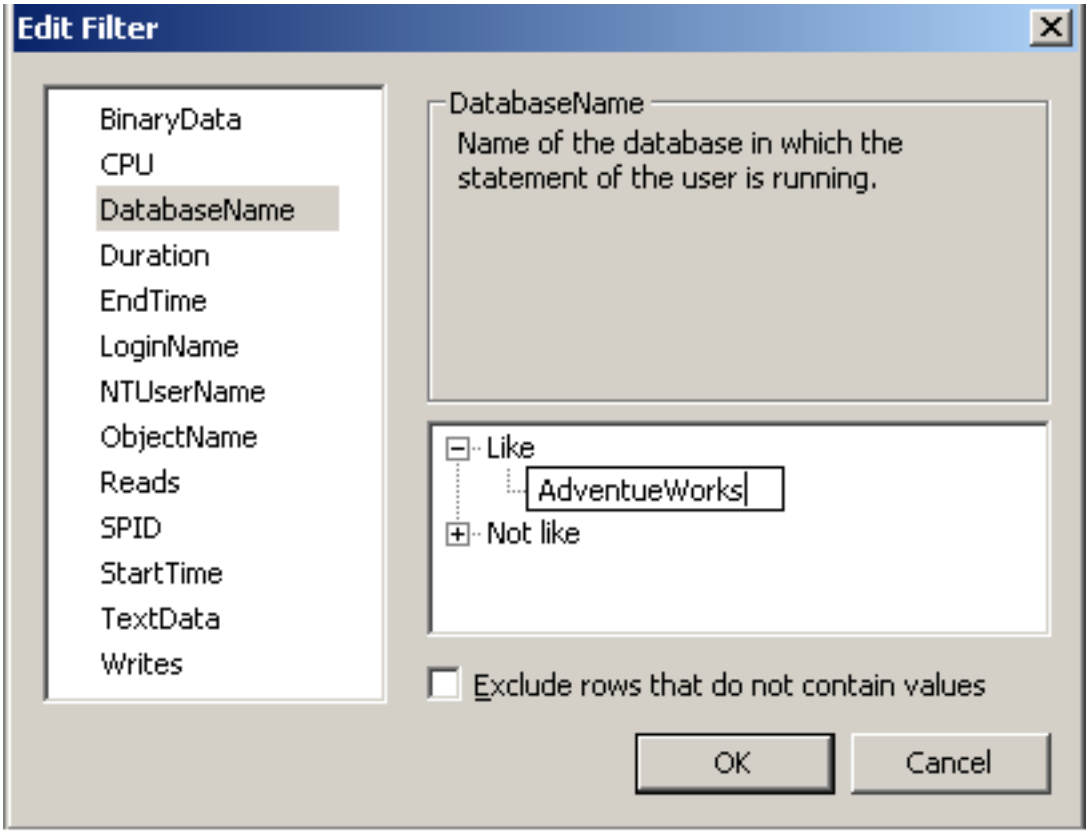


-3 بر روی قسمت Events Selection کلیک نمائید.

-4 مطابق تصویر زیر رویدادها و کلاس رویدادها را انتخاب کنید، ستون‌های TextData، NTUserName، LoginName، CPU،Reads،Writes، Duration، SPID، StartTime، EndTime، BinaryData، DataBaseName، ServerName و ObjectName را انتخاب کنید.



5- روی Column Filters کلیک کنید و مطابق تصویر زیر برای DatabaseName فیلتری تنظیم کنید.



6- روی Run کلیک کنید. تعدادی Query و رویه ذخیره شده مرتبط با پایگاه داده AdventureWorks اجرا کنید .

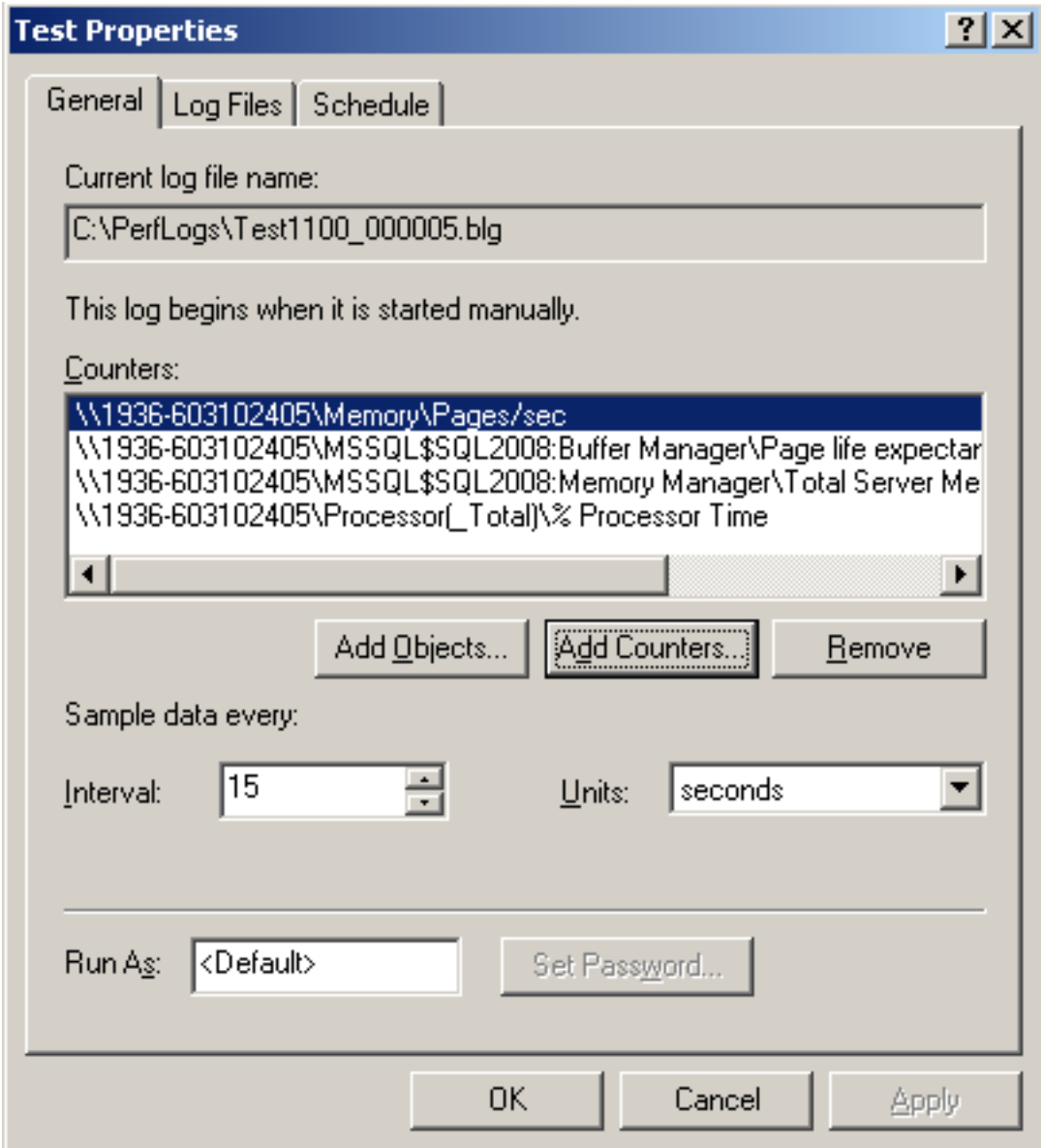
5-2- ایجاد یک Counter Log برای ایجاد یک Counter Log مراحل زیر را انجام دهید:

1- ابزار Performance را اجرا کنید (برای این کار عبارتPerfMon را در قسمت Run بنویسید).

2- در قسمت Counter Logs یک log ایجاد کنید.

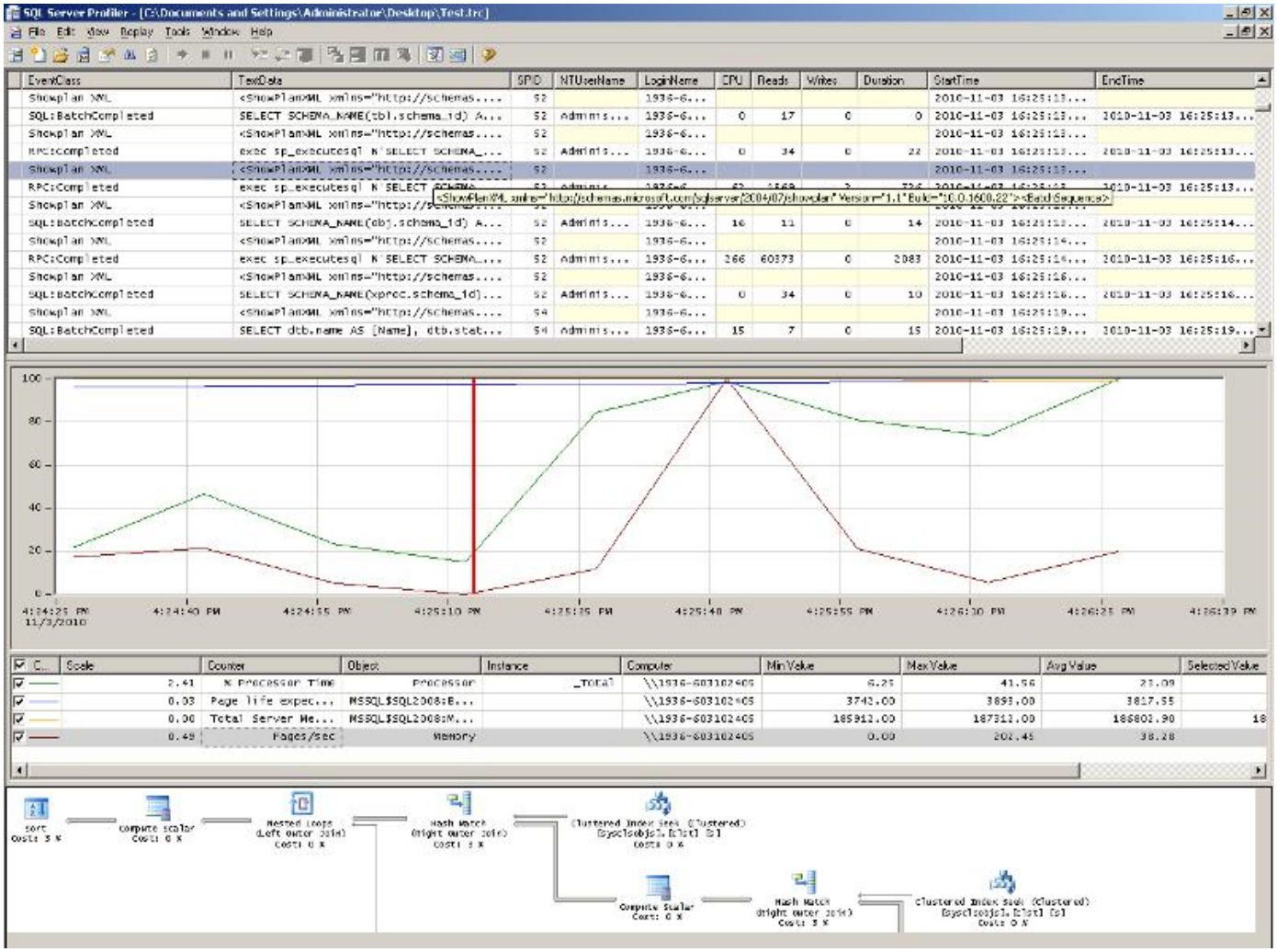
3- روی Add Counters کلیک کرده و مطابق تصویر موارد زیر را انتخاب کنید.

Performance Object	Select counters from list
Network Interface	Output Queue Length
Processor	% Processor Time
System	Processor Queue Length
SQLServer	Buffer Manager:Page life expectancy



4- روی Ok کلیک کنید تا Counter Log ذخیره شود سپس روی آن راست کلیک کرده و آنرا Start کنید.

5-3- ترکیب ابزارهای نظارتی (Correlating SQL Trace and System Monitor Data) 1- Profiler را اجرا کنید از منوی File گزینه Open و سپس Trace File را انتخاب کنید فایل trc را که در گام اول ایجاد کردید، باز نمائید.
2- از منوی File گزینه Import Performance Data را انتخاب کنید و فایل counter log را که در مرحله قبل ایجاد کردید انتخاب کنید.



نکته: اطلاعات فایل trc را می‌توان درون یک جدول وارد کرد، بدین ترتیب می‌توان آنالیز بیشتری داشت به عنوان مثال دستورات زیر این عمل را انجام می‌دهند.

```
SELECT * INTO dbo.BaselineTrace
FROM fn_trace_gettable(' c:\performance baseline.trc ', default);
```

با اجرای دستور زیر جدولی با نام BaselineTrace ایجاد و محتویات Trace مان (performance baseline.trc) در آن درج می‌گردد.