

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 نقطه ضعف سیستم‌های RDBMS می‌باشد، چرا که با افزایش شدید کاربران داده‌ها اصولاً 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 اصولاً دارای چند ویژگی زیر باشد:

۱- داده‌ها در این سیستم به صورت رابطه‌ای (جدولی) نمی‌باشند

۲- داده‌ها به صورت توزیع شده نگهداری می‌شوند.

۳- سیستم نرم‌افزاری متن باز می‌باشد.

۴- پایگاه 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 شما!، با توجه به برنامه‌های وبی فعلی ممکن است شمای نگهداری داده‌ها (ساختار کلی) مرتباً و یا گهگاهی تغییر کند. لذا در این سیستم‌ها اصولاً 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 صرف نظر 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 یا Bean 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 Basicly Available، Soft State، Eventual Consistency 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 و اگر 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 (پرس و جو 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.

نظرات خوانندگان

نویسنده: هاشمی
تاریخ: ۱۳۹۴/۰۲/۰۲ ۱۲:۴۰

سلام
من به سوال دارم
خاصیت Base که شامل Basically Available , Soft State , Eventual Consistency به چه معنی هست؟ میشه توضیح بدید؟

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۲/۰۲ ۱۳:۳۵

به مطلب [مروری بر مفاهیم مقدماتی NoSQL](#) مراجعه کنید.

عنوان:	NOSQL قسمت دوم
نویسنده:	حمید سامانی
تاریخ:	۹:۲۵ ۱۳۹۱/۱۱/۲۶
آدرس:	www.dotnettips.info
گروه‌ها:	NoSQL, Database, پایگاه داده, نوسی کوال, نواس کیوال, 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 پایگاه‌های 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 پایگاه‌های داده 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 صرفاً توسط همان کلید نگاشت شده قابل بازگردانی می‌باشد و راهی جهت دریافت مقدار بدون دانستن کلید وجود ندارد. در این ساختار داده 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 تأثیری 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 اصولاً Availability و یا در دسترس بودن داده‌ها نسبت به سایر موارد نظیر پایداری اهمیت بالاتری دارد.

از موارد استفاده این گونه سیستم‌ها به موارد زیر می‌توان اشاره کرد:

در پلتفرم‌های اشتراک گذاری داده‌ها، هدف کلی صرفاً هندل کردن آپلود محتوای (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 و بدون 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 کاملاً توسط این نوع پایگاه داده پیاده شده است نیز از [این آدرس](#) قابل مشاهده است. این برنامه به زبان‌های php , ruby و java نوشته شده است و سورس نیز در مخزن 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 صرفاً 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 پایداری می‌باشد.

همان‌گونه که در تعریف کلی عنوان شد یکی از ویژگی‌های این سیستم‌ها متن‌باز بودن آنها 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 پر استفاده 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: ۱۵:۴۷ ۱۳۹۱/۱۱/۲۸

عالی است.
متشکرم.

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

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

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

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: ۲۰:۱۷ ۱۳۹۱/۱۱/۲۸

در حالت کلی هرکدام از پایگاه داده ها بسته به نیاز استفاده می شن ، توی برنامه های اینترپرایز وبی مفهوم Polyglot Persistence مطرحه (که می شه اونو نگهداری یا ذخیره سازی چند زبانی ترجمه کرد) که می گه توی یک سیستم از چندین نوع پایگاه داده می شه (باید) استفاده کرد. به عنوان مثال برای نگهداری داده هایی جهت گزارش گیری و یا ایجاد Transaction ها بهترین گزینه همان سیستم های RDBMS هستند ، در مطالب آتی به این موضوع اشاره بیشتری خواهم کرد ، مارتین فویلر در [این مطلب](#) مفهوم Polyglot Persistence را به خوبی توضیح داده اند.

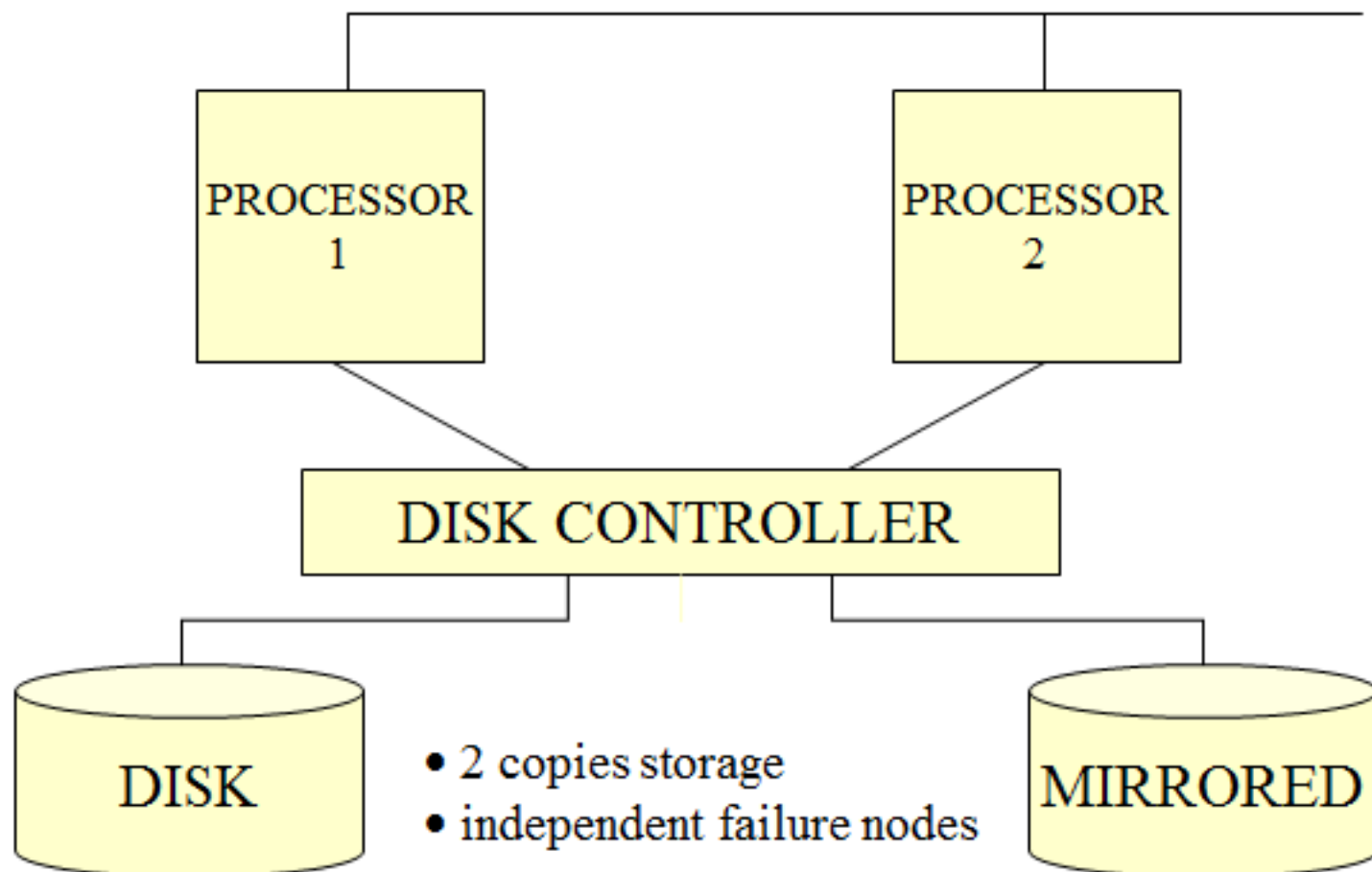
نویسنده: saremi
تاریخ: ۱۷:۲۲ ۱۳۹۲/۱۱/۲۹

سلام
می خواستم بپرسم bucket در key value store دقیقاً 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 پیچیده تر از این حرفاست؟

در این مقاله آموزشی که یکی دیگر از سری مقالات آموزشی اصول و مبانی پایگاه داده پیشرفته می باشد، قصد داریم به یکی دیگر از مقوله های مهم در طراحی سیستم های مدیریت پایگاه داده (DBMS) بپردازیم. همانطور که در مباحث قبلی بیان کردیم یکی از وظایف سیستم مدیریت پایگاه داده، حفظ سازگاری (consistency) داده ها می باشد. برای مثال یکی از راهکار هایی که برای این منظور ارائه می دهد 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، مثلاً دیسک از کار بیفتد (disk crash) یا آنکه برق قطع شود. خطاهای نرم افزاری نیز می توانند جزو موارد شکست و خرابی بحساب آیند که خطای منطق برنامه (logic) از این نمونه می باشد. در چنین شرایطی بحثی مطرح می شود 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 سیستم های مدیریت پایگاه داده را به خود اختصاص می دهند.

آنچه که در اینجا در مورد آن صحبت خواهیم کرد بازیابی بصورت نرم افزاری است که از آن تحت عنوان fail soft نام برده می شود. دقت داشته باشید در بیشتر مواقع می توان 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 نسخه های پشتیبان یک امر ضروری در سیستم های اطلاعاتی است. چرا که گاهی اوقات خرابی های فیزیکی باعث از دست رفتن تمامی اطلاعات می گردند که در این صورت نسخه های پشتیبان می توانند به کمک آیند و با کمک آنها سیستم را مجدد بازیابی کرد. در شکل زیر نمونه ای از روش های پشتیبان گیری بنام mirroring نشان داده شده است که روش رایجی در سیستم های بانک اطلاعاتی بشمار می رود. همانطور که در شکل نشان داده شده است در کنار نسخه اصلی (DISK)، نسخه (MIRROR) آن قرار داده شده است. این دو نسخه کاملاً مشابه یکدیگرند و هر عملی که در DICK انجام می شود در MIRROR آن نیز اعمال می شود تا در مواقع خرابی DISK بتوان از نسخه MIRROR استفاده نمود.

در شکل زیر نمونه بسیار ساده از نحوه لاگ کردن در حین اجرای تراکنش ها را مشاهده می کنید.



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 آن 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)** می‌باشند.

سیستم دقیقاً 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 بی‌اثر 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 مجدداً انجام گیرد. بعنوان مثال در فایل لاگ به تراکنشی برخورد می‌کنیم و سیستم تصمیم می‌گیرد که آن را مجدداً از ابتدا به اجرا در آورد. دقت داشته باشید که سیستم بر اساس قوانین و قواعدی تصمیم می‌گیرد که تراکنشی را **redo** یا **undo** نماید که در ادامه این بحث آن قوانین را باز خواهیم کرد.

در کنار لاگ فایل‌ها، 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.

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 درصد DBMS را به خود اختصاص می دهد. 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 می شود. مدتی بعد از اجرای و تمکیل تراکنش سیستم دچار مشکل می شود و مجبور به انجام بازیابی می شویم. در حین عمل بازیابی سیستم مدیریت بازیابی و ترمیم تصمیم به اجرای مجدد تراکنش مذکور می گیرد. در اینجا سیستم نباید مجدداً یک میلیون تومان دیگر به حساب آن 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 مثلاً ابتدا یک میلیون کسر و سپس یک میلیون به آن اضافه کند. این مسئله نکته بسیار مهمی است که طراحان DBMS باید حتماً آن را مد نظر قرار دهند.

لاگ کردن:

همانطور که گفته شد هر تغییری که در پایگاه داده رخ می دهد باید لاگ شود. لاگ کردن به این معنی است که هر گونه عملیاتی که در پایگاه داده انجام می شود 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 می شود. 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 می شود در فایلی مخصوص لاگ می شود. یکی از کاربردهای 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 می توان با بررسی این فایل لاگ و آنالیز فراخوانی های 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 آنها حفظ صحت، سازگاری و امنیت اطلاعات می باشد.

حال ببینیم در لاگ فایل مربوط به بازیابی اطلاعات چه چیز هایی نوشته می شود. در طول حیات پایگاه داده عملیات بسیار گوناگونی انجام می گیرد که جزئیات تمامی آنها باید لاگ شود. بعنوان مثال هنگامی که رکوردی درج می شود 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 می کنیم نام فیلد ویرایش شده، مقدار قبلی و مقدار جدید آن نیز مشخص شود. تمامی عملیات ریز لاگ می شوند 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 می تواند باشد. چراکه معمولاً فضایی که ما بر روی دیسک به دیتابیس اختصاص می دهیم فضایی محدود است. به همین دلیل به لحاظ فیزیکی نمی توان 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 فایل های حلقه ای (circular) است. بدین صورت که هنگامی که سیستم به انتهای فایل لاگ می رسد مجدداً به ابتدا آن بر می گردد و از ابتدا شروع به نوشتن می کند. البته چنین ساختار هایی بدون اشکال نیستند. چرا که پس از رسیدن به انتهای فایل و شروع مجدد از ابتدا ما برخی از تراکنش های گذشته را از دست خواهیم داد. این مسئله یکی از دلایلی است که بر اساس آن پیشنهاد می شود تا جایی که امکان دارد تراکنشها را کوچک پیاده سازی کنیم. گاهی اوقات بر روی لاگ فایل عمل فشرده سازی را نیز انجام می دهند. البته فشرده سازی بمعنای رایج آن 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. مجدداً همان رکورد را از 60 به 70 تغییر می دهیم. در این صورت برای این عملیات دو رکورد در فایل لاگ ثبت شده است که در هنگام فشرده سازی در صورت امکان می توان آن دو را به یک رکورد تبدیل نمود (تغییر از 50 به 70 را بجای آن دو لاگ کرد). بعنوان مثال دیگر فرض کنید تراکنشی در گذشته دور انجام شده است و با موفقیت کامیت شده است. می توان 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 آنها 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 آنها 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 نمونه‌هایی از رکوردهای فایل‌های لاگ هستند که در اینجا آورده شده‌اند. 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.

□ Log : $\langle \text{Tran_id_data-item-name, old-value, new-value} \rangle$

	log	DB
To : Read (A, a1)	$\langle \text{To Starts} \rangle$	A = 1000
a1 := a1 - 50	<i>Immediate updates</i>	B = 2000
write (A, a1) →	$\langle \text{To, A, 1000, 950} \rangle \rightarrow$	A = 950
Read (B, b1)		
b1 := b1 + 50		
write (B,b1) →	$\langle \text{To, B, 2000, 2050} \rangle$	B = 2050
	$\langle \text{To Commit} \rangle$	
Undo(Ti) : restore to the old-value		
Redo(Ti) : set to the new-value		

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 اصطلاحاً می‌گوییم immediate updates دارند. در مقابل این اصطلاح ما deferred را داریم. در این مدل تغییرات در انتهای کار اعمال می‌شوند (در زمان commit).

: (Write-Ahead Log (WAL

بر اساس آنچه تا بحال 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 آن 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. اصطلاحاً 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 می‌نویسیم. سپس پیش از آنکه عمل درج را انجام دهیم سیستم crash می‌کند و مجبور به بازیابی می‌شویم. در این صورت هنگامی که Recovery Manager به رکورد مربوط به عمل درج در لاگ فایل می‌رسد یا باید آن را redo کند و یا undo (بعداً می‌گوییم بر چه اساس تصمیم‌گیری 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 مجدداً 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 تراکنش‌ها، 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 آنکه 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 تراکنش‌هایی که در آن ثبت شده است، عمل ترمیم را انجام می‌دهیم. علاوه بر آن، این را هم گفتیم که لاگ فایل، معمولاً فایلی بزرگ است که از نظر منطقی با ظرفیت بینهایت پیاده‌سازی 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 زمان‌بر و پرهزینه خواهد بود. برای جلوگیری از این کار مدیر بازیابی پایگاه داده وظیفه دارد در فواصل مشخصی در لاگ فایل نقاطی را علامت‌گذاری 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 رکورد‌هایی از لاگ فایل که هنوز به دیسک منتقل نشده‌اند، 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 بگیرد 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 تراکنش‌هایی در آن زمان هنوز commit نشده و تاثیرشان به پایگاه داده اعمال نشده است. سومین کاری که در این لحظه باید انجام گیرد اینست که اگر داده‌هایی از پایگاه داده هستند که عملیات مربوط به آنها COMMIT شده‌اند اما هنوز به دیسک منتقل نشده‌اند 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 مجدداً باز گردانیم. هنگامی که عمل undo را بر روی عملیات درج (insert) می‌خواهیم 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 تقسیم کنیم، تراکنش‌هایی که باید undo شوند و تراکنش‌هایی که باید عمل redo بر روی آنها 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؟

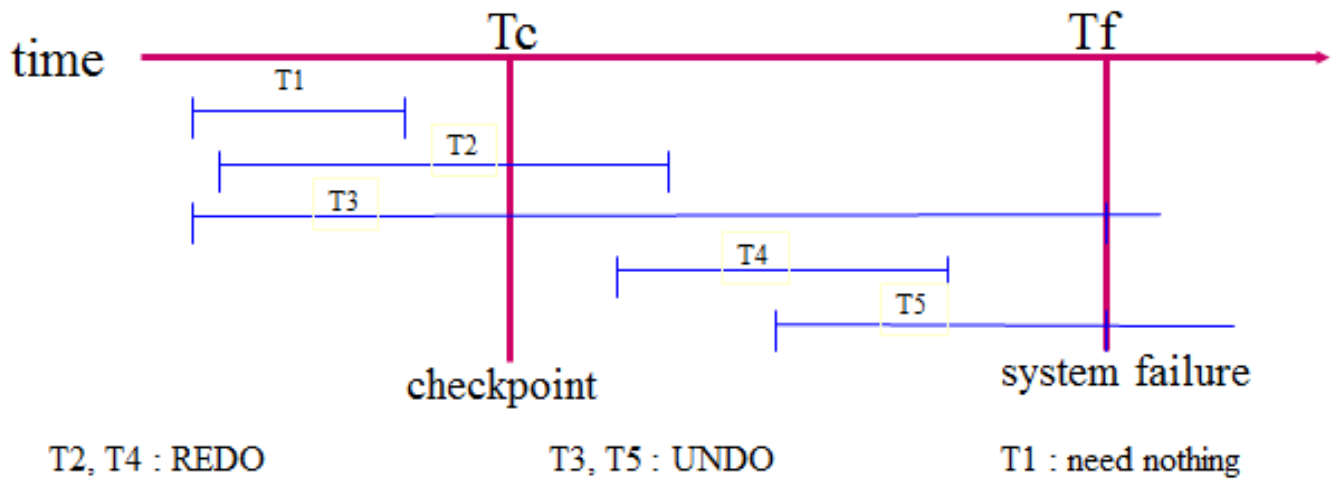
هنگام اسکن کردن (از نقطه checkpoint به سمت انتهای لاگ فایل (لحظه خطا))، هر تراکنشی که رکورد لاگ مربوط به commit آن دیده شود باید در گروه redo قرار گیرد. عبارت دیگر تراکنش‌هایی که در این فاصله commit شده اند را در گروه redo قرار می‌دهیم. در مقابل هر تراکنشی که commit آن دیده نشود (commit نشده اند) باید 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 می‌باشد که حاوی تراکنش‌هایی است که در زمان checkpoint در حال انجام بوده اند، یعنی هنوز commit نشده اند. بنابراین تمامی این تراکنش‌ها را ابتدا در گروه تراکنش‌هایی که باید undo شوند قرار می‌دهیم. بمرور که عمل اسکن را ادامه می‌دهیم اگر به تراکنشی رسیدیم که رکورد مربوط به شروع آن ثبت شده باشد، باید آن تراکنش را در لیست undo قرار دهیم. تراکنش‌هایی 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 تراکنش‌هایی 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 تراکنش‌هایی که باید undo شوند را از پایین به بالا undo کنیم. یکی از دلایل اینکه ابتدا عملیات undo را انجام می‌دهیم اینست هنگامی که تراکنش‌ها commit نشده اند، قفل‌هایی 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 می‌رسیم. 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. دلیل 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 transaction مربوط

به تراکنش T4 می‌رسیم. بهمین دلیل این تراکنش را به لیست undo ها اضافه می‌کنیم. پس از آن به commit تراکنش T2 می‌رسیم. همانطور که گفته شد باید T2 را از لیست undo ها خارج و به یست تراکنش‌هایی که باید redo شوند اضافه گردد. سپس به تراکنش T5 می‌رسیم 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 می‌رسیم. در ان لحظه لیست undo ها شامل دو تراکنش T3 و T5 و لیست Redo ها شامل تراکنش‌های T2 و T4 می‌باشند. در مورد تراکنش T1 نیز چون پیش از لحظه Tc کامیت شده است عملی صورت نمی‌گیرد.



موفق و پیروز باشید