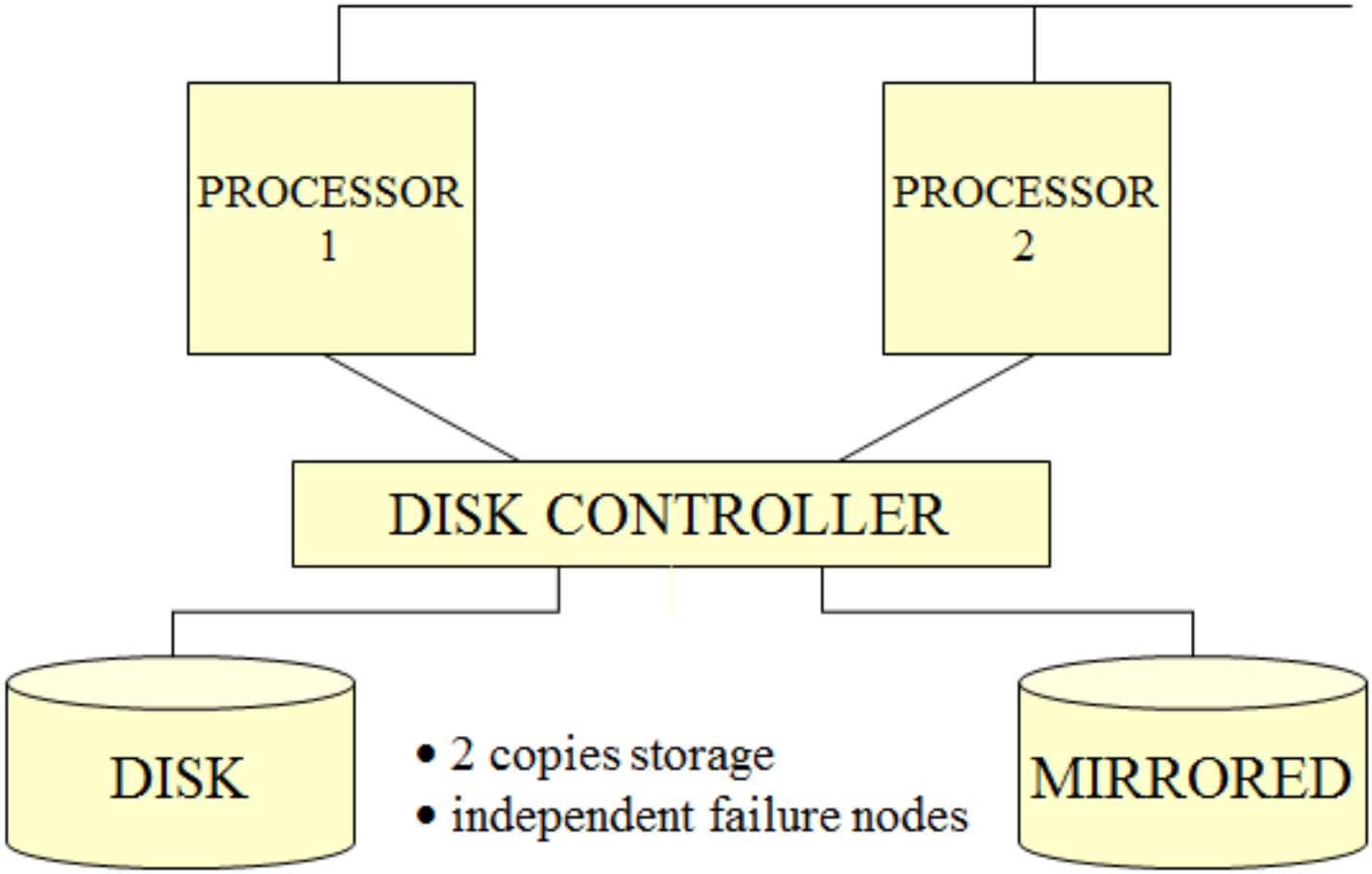
عنوان: **بازیابی پایگاه داده (database recovery)**
نویسنده: حامد خسروجردی
تاریخ: ۱۳۹۲/۱۲/۱۷ ۰:۰
آدرس: www.dotnettips.info
گروه‌ها: Database, پایگاه‌داده

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]

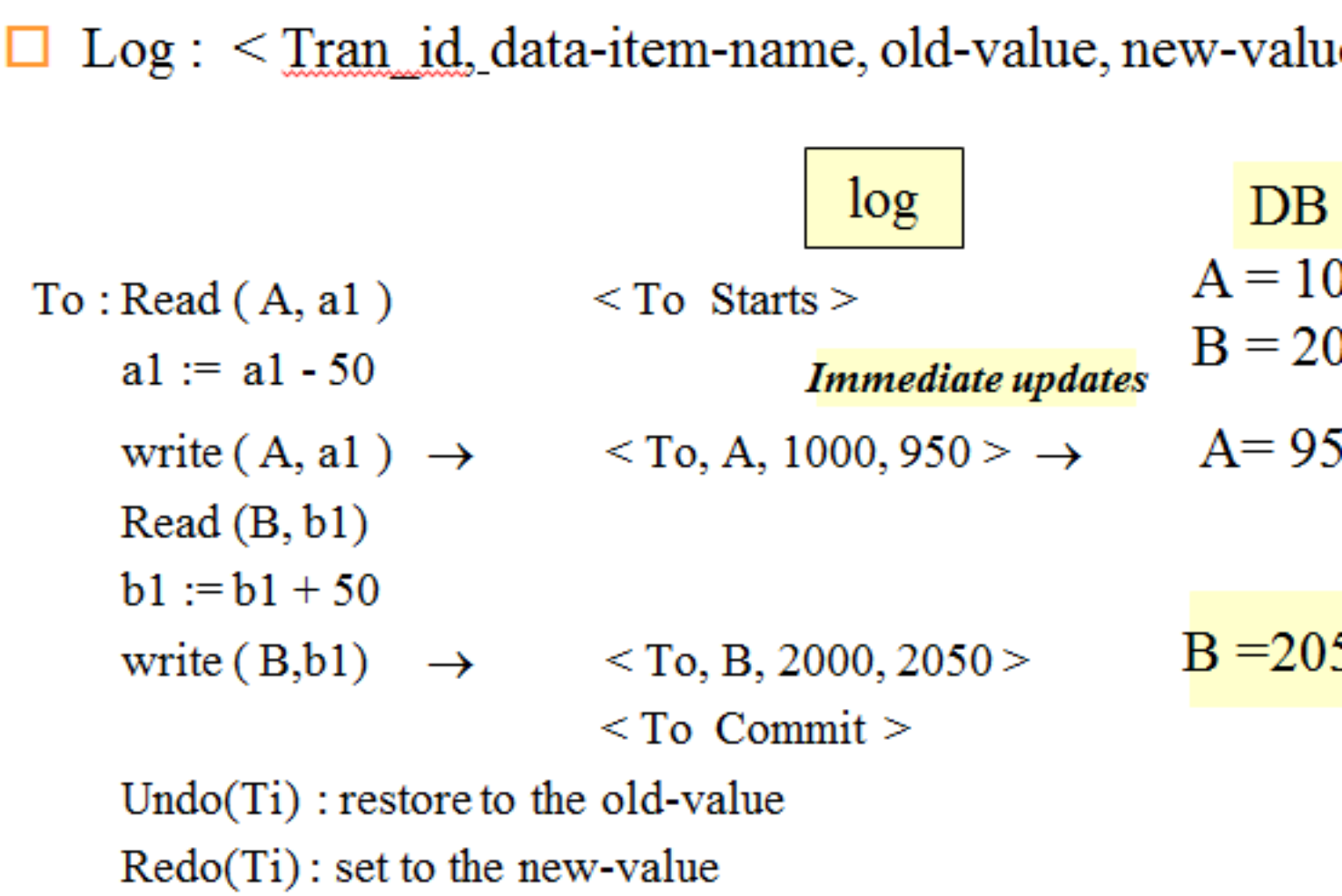
[write-item, T, X, old-value, new-value]

[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؟

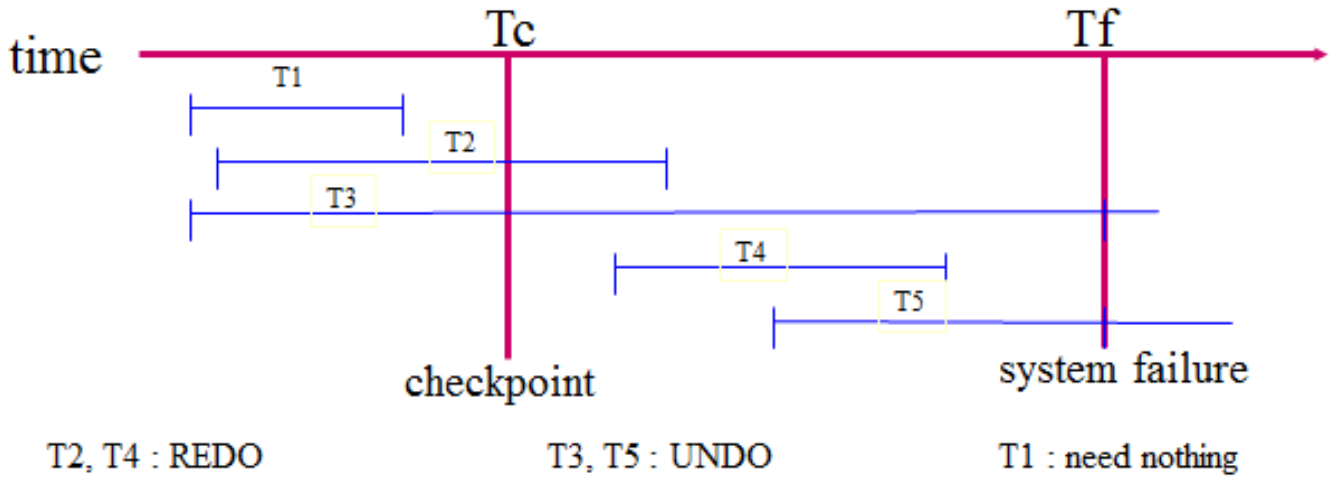
هنگام اسکن کردن (از نقطه checkpoint به سمت انتهای لاگ فایل (لحظه خطا))، هر تراکنشی که رکورد لاگ مربوط به commit آن دیده شود باید در گروه redo قرار گیرد. بعبارت دیگر تراکنش هایی که در این فاصله commit شده اند را در گروه redo قرار می‌دهیم. در مقابل هر تراکنشی که commit آن دیده نشود (commit نشده اند) باید 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