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.

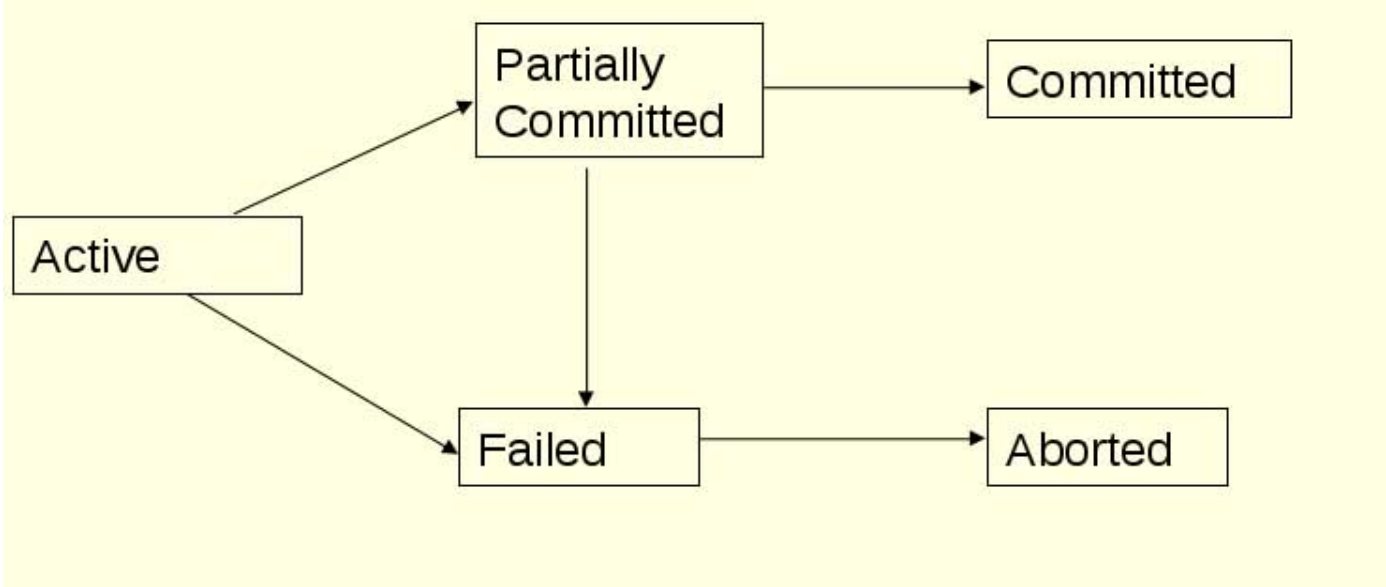
اینکه پایگاه داده ما در چه سطحی از ایزولاسیون باید عمل نماید توسط کاربر تعیین می‌شود. البته بحث در مورد ارجای موازی تراکنش‌ها و ایزولاسیون آنها بسیار مفصل است و انشاالله در مطلبی دیگر به آن خواهیم پرداخت. **Durability** :

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: **بهبود عملکرد SQL Server Locks در سیستم‌های با تعداد تراکنش بالا در Entity Framework**
نویسنده: یاسر مرادی
تاریخ: ۱۳۹۳/۰۴/۲۷ ۲۳:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, SQL Server, Performance, transaction, Entity Framework 6

بر اساس رفتار پیش فرض در دیتابیس SQL Server، در زمان انجام دادن یک دستور که منجر به ایجاد تغییرات در اطلاعات موجود در جدول می‌شود (برای مثال دستور Update)، جدول مربوطه به صورت کامل Lock می‌شود، ولو آن دستور Update، فقط با یکی از رکوردهای آن جدول کار داشته باشد.

در سیستم‌های با تعداد تراکنش بالا و دارای تعداد زیاد کلاینت، این رفتار پیش فرض موجب ایجاد صفی از تراکنش‌های در حال انتظار بر روی جداولی می‌شود که ویرایش‌های زیادی بر روی آنها رخ می‌دهد.
اگر چه که بنظر این مشکل راه حل‌های زیادی دارد ، لکن آن راه حلی که همیشه موثر عمل می‌کند استفاده از SQL Server Table Hints است.
SQL Server Table Hints به تمامی آن دستوراتی گفته می‌شود که هنگام اجرای دستور اصلی (برای مثال Select و یا Update) رفتار پیش فرض SQL Server را بر اساس Hint ارائه شده تغییر می‌دهند.
لیست کامل این Hintها را می‌توانید در اینجا مشاهده کنید .
Hint ای که در اینجا برای ما مفید است، آن است که به SQL Server بگوییم هنگام اجرای دستور Update، به جای Lock کردن کل جدول، فقط رکورد در حال ویرایش را Lock کند، و این باعث می‌شود تا باقی تراکنش ها، که ای بسا با سایر رکوردهای آن جدول کار داشته باشند متوقف نشوند، که البته این مسئله کمی به افزایش مصرف حافظه می‌انجامد، لکن مقدار افزایش بسیار ناچیز است.
این Hint که rowlock نام دارد در تراکنش‌های با Isolation Level تنظیم شده بر روی Snapshot باید با یک Table Hint دیگر با نام updlock ترکیب شود.

توضیحات مفصل‌تر این دو Hint در لینک مربوطه آمده است.
بنابر این، بجای دستور

```
update products
set Name = "Test"
Where Id = 1
```

داریم

```
update products with (nolock,updlock)
set Name = "Test"
where Id = 1
```

تا اینجا مشکل خاصی وجود ندارد، آنچه که از اینجا به بعد اهمیت دارد این است که در هنگام کار با Entity Framework، اساسا ما نویسنده دستورات Update نیستیم که به آنها Hint اضافه کنیم یا نه، بلکه دستورات SQL بوسیله Entity Framework ایجاد می‌شوند.
در Entity Framework، مکانیزمی تعبیه شده است با نام Db Command Interceptor که به شما اجازه می‌دهد دستورات SQL ساخته شده را Log کنید و یا قبل از اجرا تغییر دهید ، که برای اضافه نمودن Table Hintها ما از این روش استفاده می‌کنیم، برای انجام این کار داریم: (توضیحات در ادامه)

```
public class UpdateRowLockHintDbCommandInterceptor : IDbCommandInterceptor
    {
        public void NonQueryExecuting(DbCommand command, DbCommandInterceptionContext<Int32>
interceptionContext)
        {
            if (command.CommandType != CommandType.Text) return; // (1)
            if (!(command is SqlCommand)) return; // (2)
            SqlCommand sqlCommand = (SqlCommand)command;
            String commandText = sqlCommand.CommandText;
            String updateCommandRegularExpression = "(update) ";
```

```
            Boolean isUpdateCommand = Regex.IsMatch(commandText, updateCommandRegularExpression,
RegexOptions.IgnoreCase | RegexOptions.Multiline); // You may use better regular expression pattern
here.
            if (isUpdateCommand)
            {
                Boolean isSnapshotIsolationTransaction = sqlCommand.Transaction != null &&
sqlCommand.Transaction.IsolationLevel == IsolationLevel.Snapshot;
                String tableHintToAdd = isSnapshotIsolationTransaction ? " with (rowlock , updlock) set
" : " with (rowlock) set ";
                commandText = Regex.Replace(commandText, "^(set) ", (match) =>
                {
                    return tableHintToAdd;
                }, RegexOptions.IgnoreCase | RegexOptions.Multiline);
                command.CommandText = commandText;
            }
        }
```

این کد در قسمت (1) ابتدا تشخیص می‌دهد که آیا این یک Command دارای Command Text است یا خیر، برای مثال اگر فراخوانی یک Stored Procedure است، ما با آن کاری نداریم.

در قسمت دوم تشخیص می‌دهیم که آیا با SQL Server در حال تعامل هستیم، یا برای مثال با Oracle و ...، که ما برای Tableها فقط با SQL Server کار داریم.

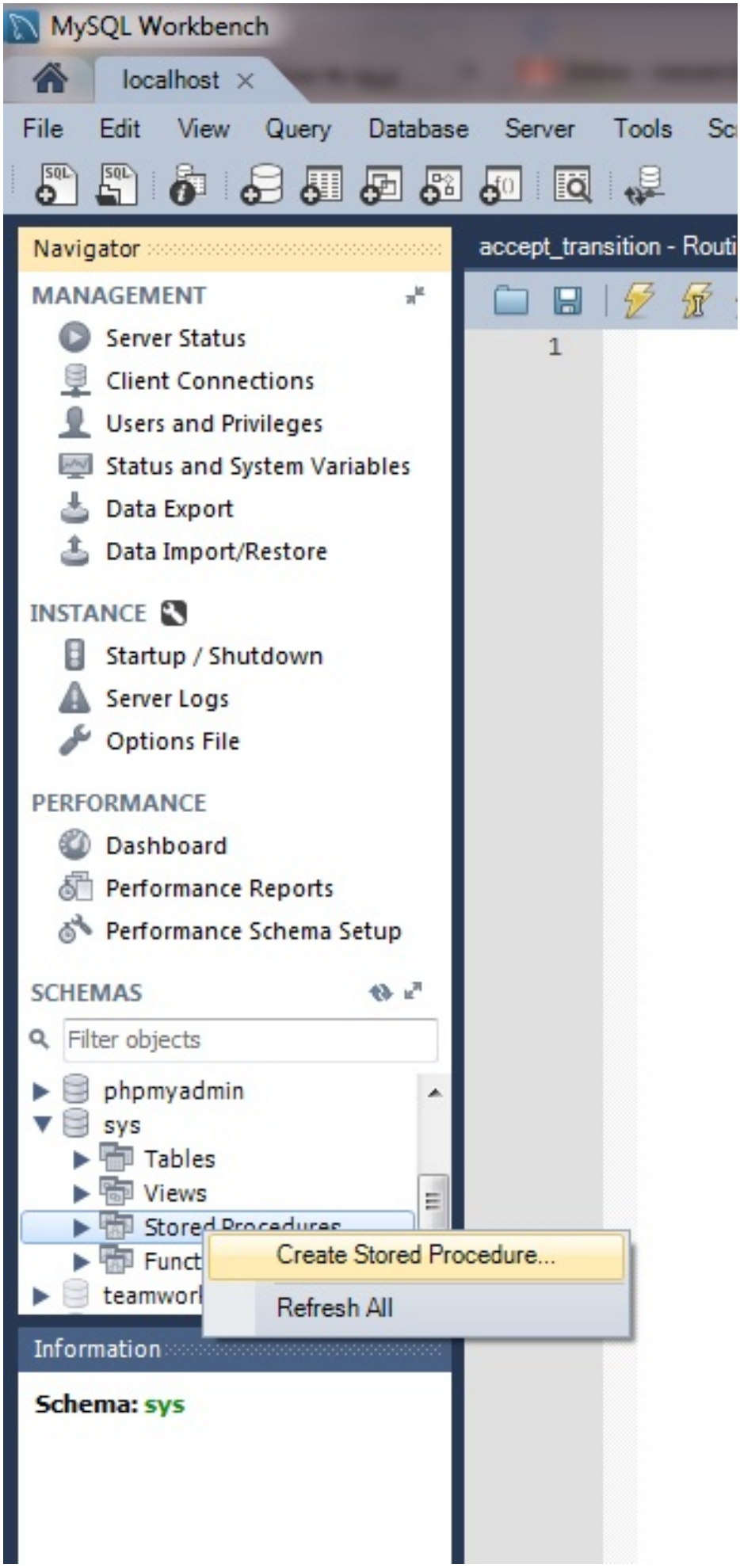
سپس باید تشخیص دهیم که آیا این یک دستور update است یا خیر ؟ برای این منظور از Regular Expressionها استفاده کرده ایم، که خیلی به بحث آموزش این پست مربوط نیست، به صورت کلی از Regular Expressionها برای یافتن و بررسی و جایگزینی عبارات با قاعده در هنگام کار با رشته‌ها استفاده می‌شود.

ممکن است Regular Expression ای که شما می‌نویسید بسیار بهتر از این نمونه باشد، که در این صورت خوشحال می‌شوم در قسمت نظرات آنرا قرار دهید.

در نهایت با بررسی Transaction Isolation Level مربوطه که Snapshot است یا خیر، به درج یک یا هر دو Table Hint مربوطه اقدام می‌نماییم.

عنوان: **مثال ساده پرداخت بانکی با استفاده از تراکنش و پروسیجر در مای اس کیو ال**
نویسنده: ناصرنیازی
تاریخ: ۱۳۹۳/۱۰/۳۰ ۱۴:۲۵
آدرس: www.dotnettips.info
گروه‌ها: MySQL, transaction, Stored procedure

برای انجام عملیاتی مثل عملیات حسابداری، نیاز به انجام پی در پی چندین دستور می‌باشد و در صورت انجام نشدن یکی از آنها، بقیه نیز نامعتبر خواهند بود که برای پیاده سازی این مکانیزم از تراکنش‌ها در بانک اطلاعاتی استفاده می‌شود. تراکنش‌ها معمولآ در بدنه‌ی توابع ذخیره شده روی بانک (stored procedure) پیاده سازی می‌شوند. برای تعریف یک پروسیجر در مای اس کیو ال من از برنامه‌ی MySQL Workbench به شکل زیر استفاده می‌کنم. البته می‌توان دستور ایجاد تابع را از روش‌های دیگر هم اجرا کرد.


MySQL Workbench
localhost
File Edit View Query Database Server Tools Sc
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
PERFORMANCE
Dashboard
Performance Reports
Performance Schema Setup
SCHEMAS
Filter objects
phpmyadmin
sys
Tables
Views
Stored Procedures
Funct
teamwork
Create Stored Procedure...
Refresh All
Information
Schema: sys
accept_transition - Routi
1

در مای اس کیو ال برای تعریف یک تابع از ساختار زیر استفاده می‌کنیم :

```
DELIMITER $$

CREATE
          DEFINER=`user_name`@`host_name`|CURRENT_USER
          PROCEDURE `transition_name`(

IN | OUT  | INOUT `parameter_name` type(bigint,int , ...)
)
    SQL SECURITY  DEFINER| INVOKER
transition_name: BEGIN

#----procedure_body

END
```

نکات مربوط به تعریف :
در قسمت

```
DEFINER=`user_name`@`host_name`|CURRENT_USER
```

کسی که تابع را تعریف کرده معرفی می‌شود. اگر شما برای انتقال دیتابیس از جایی به جای دیگر، از روش ایمپورت و اکسپورت استفاده کنید، اگر نام کاربری بانک شما متفاوت باشد، معمولاً این قسمت باعث خطا می‌شود؛ چون شما نمی‌توانید به نام فرد دیگری تابع بسازید. پیش فرض هم مقدار

```
CURRENT_USER
```

در نظر گرفته می‌شود که همان اسم کاربری و هاست شما است.
نکته بعدی : قسمت

```
SQL SECURITY  DEFINER| INVOKER
```

است که استفاده کننده از پروسیجر را مشخص می‌کند. مقدار DEFINER یعنی فقط تعریف کننده حق استفاده از این پروسیجر را دارد و مقدار INVOKER یعنی هر کسی حق استفاده از این تابع را دارد .
برای شرح تراکنش، مثال پرداخت بانکی را شرح می‌دهیم:

```
DELIMITER $$

CREATE
        DEFINER=CURRENT_USER
        PROCEDURE `transition_pay`(
                #-----------input value
               IN `pay_value` bigint,
               IN `admin_id` int,
               #-------------result code
               OUT `result` bigint
)
    SQL SECURITY INVOKER
transition_pay: BEGIN
DECLARE  admin_credit DOUBLE  DEFAULT  0;
SELECT `Credit`
INTO   admin_credit
FROM  `Admin`

WHERE `Admin_id` = admin_id
#----- transaction  body
END
```

در قسمت بالا متغیری را تعریف کرده و آخرین میزان اعتبار ادمین را داخل آن قرار دادیم تا در قسمت تراکنش، مقدار پرداختی را به آن اضافه کنیم و دو باره ادمین را آپدیت کنیم.
اگر بخواهیم به دلیلی قبل از رسیدن به تراکنش آن را کنسل کنیم، می‌توان از دستور LEAVE استفاده کرد:
مثال :

```
IF admin_id=0 THEN
set result = -1 ;
#exit procedure
LEAVE transition_pay;
END IF;
```

حال شروع تراکنش حالت ساده :

```
START TRANSACTION;
          INSERT INTO
                                `PayBalance` (`Value` , `Admin_id` )
                                 VALUES (pay_value,  admin_id);

          UPDATE `Admin`
           SET `Credit`=admin_credit + pay_value
          WHERE `admin_id`=admin_id;
COMMIT;
```

با پایان تراکنش، تمام مقادیر به درستی در بانک ذخیره می‌گردند.
حال اگر بخواهیم به دلیلی داخل تراکنش آن را لغو کنیم از دستور ROLLBACK استفاده می‌کنیم.
مثال:

```
IF pay_value=0 THEN
set result = -1 ;
#roolback procedure
ROLLBACK ;
END IF;
```

برای اطمینان از اجرا شدن دستورات در مای اس کیو ال می‌توان از

```
SET autocommit = {0 | 1}
```

نیز استفاده کرد که مقدار پیش فرض آن یک است. یعنی هر دستوری بلافاصله اجرا شود. می‌توان قبل از دستوراتی که می‌خواهیم پی در پی اجرا شوند، یک بار آن را صفر و بعد از اجرای دستورات آنرا یک کنیم.
نکته آخر اینکه با استفاده ار زبان پی اچ پی هم می‌توان تراکنشی را شروع و تمام کرد و بین این دو دستورات مورد نظر را نوشت و همیشه وجود پروسیجر الزامی نیست.