فهرست

۲ فصل شانزدهم

هم‌روندی

۱,۱۶ مقدمه

اصطلاح هم‌روندی در واقع بدین مفهوم است که سیستم‌های مدیریت پایگاه داده (DBMSs) اجازه دهند، چندین تراکنش در یک زمان به پایگاه داده دسترسی پیدا کنند. در این چنین موارد، لازم است که سیستم از یک سری مکانیسم کنترلی استفاده کند تا مطمئن گردد که تراکنش‌ها در کار یکدیگر مداخله نداشته باشند. در این فصل ما این موضوع را به تفصیل بحث خواهیم کرد. ساختار این فصل به شرح زیر است:

- بخش ۱۶,۲ مشکلاتی را که اگر کنترل نکردن هم‌روندی، ممکن است به وجود بیایند را تشریح خواهد کرد.
- بخش ۱۶,۳ مقدمه‌ای بر قفل گذاری که شیوه متعارف برای حل این‌چنین مشکلات است را ارائه می‌دهد. قفل گذاری تنها شیوه ممکن نیست اما در عمل بسیار متداول‌تر از دیگر شیوه‌ها است.
- سپس در بخش ۱۶,۴ خواهید دید که چطور قفل گذاری مشکلات مطرح شده را حل خواهد نمود.
- متأسفانه قفل گذاری مشکلات خاص خود را خواهد داشت، یکی از معروف‌ترین آن‌ها بن بست[1] است در بخش ۱۶,۵ در این مورد بحث شده است.
- در بخش ۱۶,۶ مفهوم توالی پذیری[2] مطرح می‌گردد که در اینجا به عنوان ملاک رسمی درستی بکار برده می‌شود.

---

[1] deadlock
[2] serializability

- در بخش ١٦,٧ در مورد تأثیر هم‌روندی بر مباحث فصل گذشته، ترمیم، بحث می‌شود.
- بخش‌های ١٦,٨ و ١٦,٩ دو پالایش مهم را بر روی قفل گذاری اولیه، سطوح جدایی[1] و قفل گذاری قصدی[2] را مورد بررسی قرار می‌دهند.
- بخش ١٦,١٠ مشاهدات اندکی مشکوک و جدید در مورد خصوصیات تراکنش‌ها که اصطلاحاً ACID نامیده می‌شوند، پیشنهاد می‌گردد.
- بخش ١٦,١١ امکان مرتبط SQL را خواهد گفت.

بخش مقدمه را با دو نکته کلی به پایان می‌بریم. اول اینکه هم‌روندی نیز مانند ترمیم از اینکه سیستم مدیریت پایگاه داده‌ها رابطه است یا خیر، تا حد زیادی مستقل است؛ و دوم اینکه هم‌روندی نیز مانند ترمیم یک موضوع بسیار گسترده است و امیدواریم که بتوانیم در این فصل مقدمه‌ای از برخی مفاهیم اولیه و مهم هم‌روندی را مطرح سازیم.

٢,١٦ سه مشکل هم‌روندی

این بخش را با مشکلاتی که هر مکانیزم کنترل هم‌روندی باید مورد توجه قرار دهد، شروع می‌کنیم. اگر در هنگام اجرا تراکنش، تراکنش‌های دیگر با آن تداخل داشته باشند در برخی شرایط این تداخل منجر به این خواهد شده که پایگاه داده از سازگاری که در بخش پیش تشریح شد، خارج گردد. این بدین مفهوم نیست که هر اجرای تراکنش‌ها به صورت متداخل منجر به این می‌شود که پایگاه داده از درستی (سازگاری) خارج گردد، بلکه موارد کنترل نشده عملیات دو تراکنش که هر یک به طور جداگانه درست هستند، باعث می‌گردد که نتیجه کلی نادرست باشد. این سه مشکل عبارتند از[3]:

---

[1] Levels of isolation
[2] Intended locking
[3] البته در بعضی از منابع چهار مشکل مطرح می‌گردد.

- مشکل نتیجه از دست رفته[1]
- مشکل وابستگی تثبیت نشده[2]
- مشکل تحلیل ناسازگار[3]

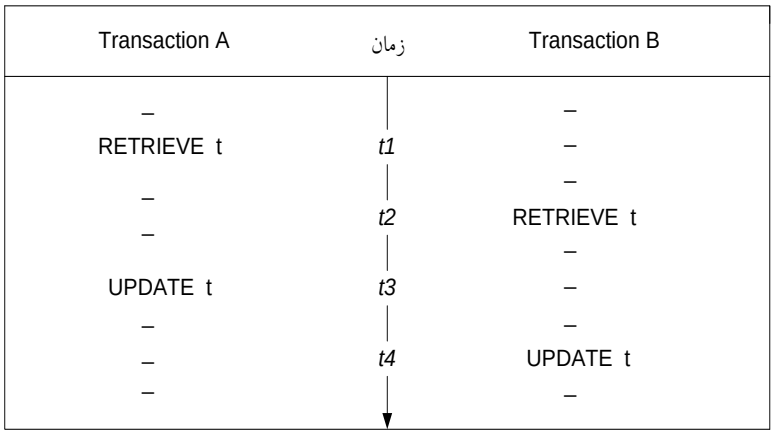
در ادامه هر یک از مشکلات را شرح داده و مثال قید می‌کنیم.

مشکل نتیجه از دست رفته

شکل ۱۶,۱ را ملاحظه فرمایید، در این شکل خط وسط زمان را نشان می‌دهد و به این ترتیب خوانده می‌شود. تراکنش A تاپل t را در زمان t_1 بازیابی می‌کند. (در واقع تراکنش A تاپل t را در زمان t_1 می‌خواند). تراکنش B همان تاپل t را در زمان t_2 بازیابی می‌کند. تراکنش A تاپل t را در زمان t_3 به هنگام می‌سازد (بر اساس همان مقداری که در زمان t_1 خوانده بود) و تراکنش B تاپل t را در زمان t_4 به هنگام می‌سازد (بر اساس همان مقداری که در زمان t_2 خوانده بود که آن همان مقداری است که در زمان t_1 خوانده شده بود). به هنگام سازی تراکنش A در زمان t_4 از بین رفته است زیرا تراکنش B بدون توجه به آن بر روی آن به هنگام سازی، نوشتن مجدد انجام داده است.

---

[1] Lost update
[2] Uncommitted dependency
[3] Inconsistent analysis

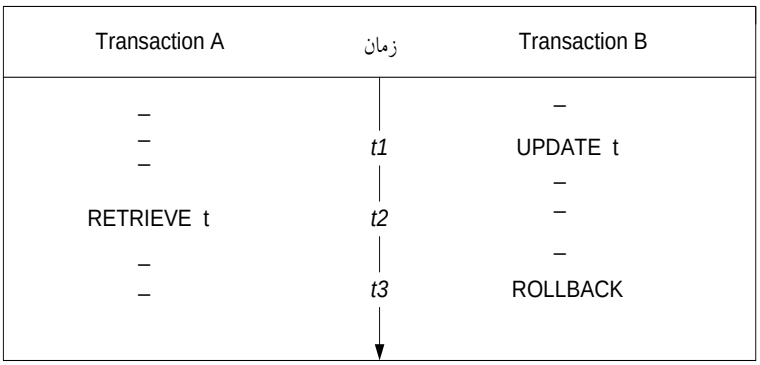


شکل ١٦,١. در زمان t_4 نتیجه یک به هنگام سازی تراکنش A از بین می‌رود.

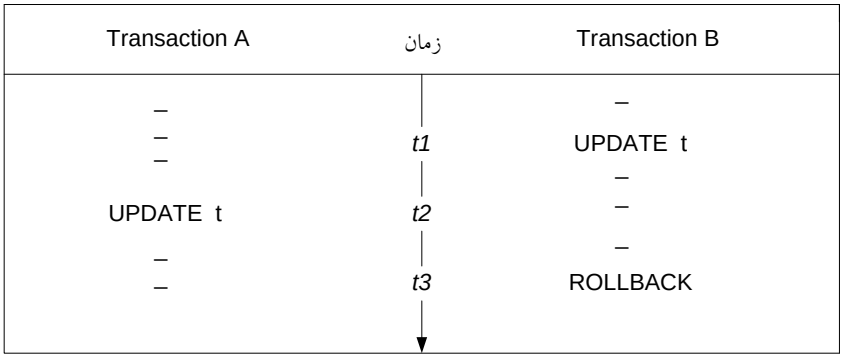
مشکل وابستگی تثبیت نشده

مشکل وابستگی تثبیت نشده زمانی بروز می‌کند که به تراکنشی اجازه داده شده باشد تاپلی را بازیابی کند که این تاپل توسط تراکنش دیگر به هنگام شده اما هنوز تثبیت نشده است. چون تراکنش دیگر هنوز تثبیت نشده است پس همیشه این امکان وجود دارد که هیچ موقع آن تراکنش تثبیت نگردد و بخواهد واکرد انجام دهد. در چنین حالتی تراکنش اول داده‌های را دیده (خوانده) که دیگر وجود ندارد (در واقع هرگز وجود نداشته است). شکل‌های ١٦,٢ و ١٦,٣ را مشاهده کنید.

در مثال اول (شکل ١٦,٢) تراکنش A در زمان t_2 یک به هنگام سازی تثبیت نشده را دیده است. آن به هنگام سازی در زمان t_3 خنثی می‌گردد. بنابراین تراکنش A بر روی یک فرض غلط عمل می‌کند. یعنی، فرضی که در آن تاپل t، مقدار دیده شده در زمان t_2 را دارد در صورتی که در حقیقت باید مقدار قبلی در زمان t_1 را داشته باشد. در نتیجه تراکنش A ممکن است یک نتیجه نادرست را تولید کند. نکته واکرد تراکنش B ممکن است ناشی از اشتباه در B نباشد بلکه نتیجه یک خرابی سیستم باشد. (و ممکن است تراکنش A در آن زمان تثبیت شده باشد و در چنین حالتی خرابی موجب واکرد تراکنش A نمی‌گردد).



شکل ۱۶,۲. تراکنش A در زمان t2 به یک تغییر تثبیت نشده، وابسته شده است.



شکل ۱۶,۳ . تراکنش Aتغییر تثبیت نشده را در زمان t1به هنگام می‌سازد و این به هنگام سازی در زمان t3 از بین می‌رود.

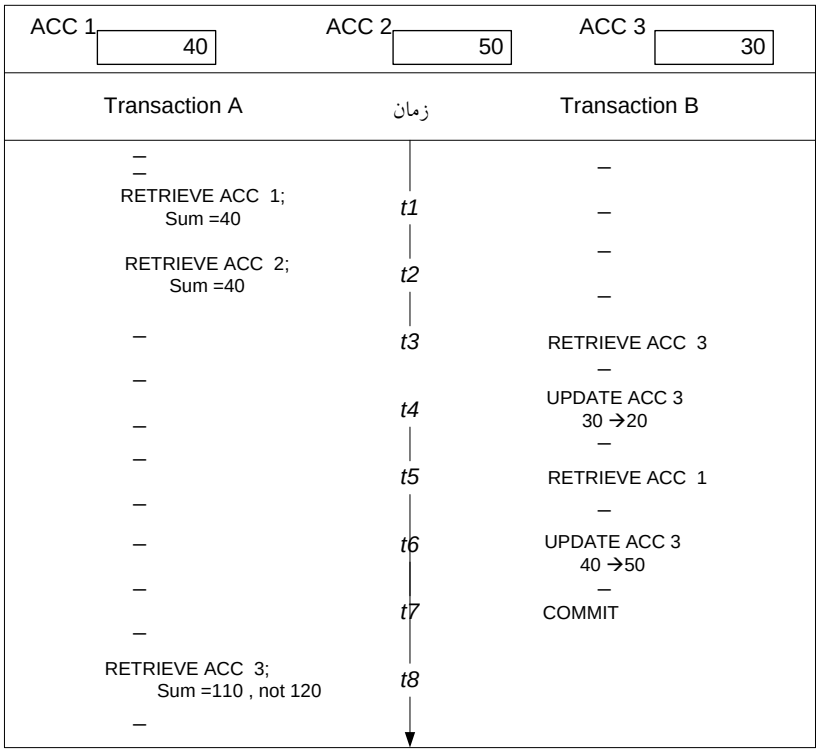
مثال دوم (شکل ۱۶,۳) حتی بدتر است. نه تنها تراکنش A به یک تغییر تثبیت نشده در زمان t_2 وابسته شده بلکه به هنگام سازی تراکنش A در زمان t_3 از دست خواهد رفت. زیرا واکرد در زمان t_3 باعث خواهد شد که تاپل t با مقدار قبل از t_1 بازگردانده شود. این نسخه دیگری از مشکل تحلیل ناخواسته است.

مشکل تحلیل ناسازگار

شکل ۱۶,۴ را ملاحظه کنید که تراکنش‌های A و B بر روی تاپل های (ACC) عمل می‌کنند. تراکنش A موجودی حساب‌ها را جمع می‌کند و تراکنش B یک مقدار$۱۰ را از حساب با شماره ۳ به حسابی با شماره ۱ منتقل می‌نماید نتیجه حاصل

از تراکنش A برابر ۱۱۰$ است که کاملاً روشن است که این مقدار اشتباه است. اگر تراکنش A نتیجه حاصله را بخواهد برگرداند باعث خواهد شد که پایگاه داده به یک وضع نادرست برود. در حقیقت تراکنش A پایگاه داده را در یک وضع نادرست، دیده و بنابراین یک تحلیل ناسازگار انجام داده است. نکته تفاوت بین این مثال و مثال قبلی در این است که هیچ تردیدی وجود ندارد که در اینجا تراکنش A به یک تغییر تثبیت نشده وابسته است چون تراکنش B همه به هنگام سازی‌ها را قبل از دیدن شماره حساب ۳ (ACC3) توسط A، تثبیت کرده است.

شکل ۱۶،٤. تراکنش A یک تحلیل ناسازگار انجام داده است.



یک نگاه دقیق‌تر

بیاید در این قسمت نگاهی دقیق‌تر به مسائل پیشین داشته باشیم. روشن است عملیات‌هایی که از دیدگاه هم‌روندی پایگاه داده مورد توجه قرار می‌گیرد، بازیابی و به هنگام سازی است: به عبارت دیگر می‌توان یک تراکنش را تنها به عنوان یک توالی از یک چنین عملیات در نظر گرفت. (البته جدا از عملیات لازم نظیر BEGIN TRANSACTION و COMMIT یا ROLLBACK). بیاید برای سادگی توافق کنیم که این دو عملیات را به ترتیب خواندن (reads) و نوشتن (writes) در نظر بگیریم. پس واضح است که اگر تراکنش‌های A و B به طور هم‌روند اجرا گردند، و اگر A و B بخواهند شئ مشخصی از پایگاه داده (مثلاً تاپل t) را بخوانند یا بنویسند، مشکلاتی می‌تواند بروز کند.

چهار امکان وجود دارد.

- RR : تراکنش‌های A و B هر دو تاپل t را می‌خوانند. خواندن نمی‌تواند در اجرای تراکنش دیگر تداخل ایجاد کند و بنابراین در این حالت هیچ مشکلی به وجود نمی‌آید.
- RW : تراکنش A تاپل t را می‌خواند و سپس تراکنش B می‌خواهد بر روی تاپل t بنویسد. اگر B اجازه یابد که این نوشتن را انجام دهد این کار باعث می‌گردد که مشکل تحلیل ناسازگار بروز کند، بنابراین می‌توان گفت که تعارض RW[1] سبب مشکل تحلیل ناسازگار می‌شود. نکته آن که اگر تراکنش B نوشتن خود را انجام دهد و تراکنش A یک‌بار دیگر تاپل t را بخواند، مقداری که اکنون تراکنش A خوانده و مقداری که قبلاً خوانده با هم فرق دارد و این حالت

---

[1] RW conflict

یک خواندن تکرار نشدنی[1] است. بنابراین، خواندن تکرار نشدنی نیز نتیجه تعارض RW است.

- WR: تراکنش A تاپل t را می‌نویسد و سپس تراکنش B قصد دارد تاپل t را بخواند. اگر تراکنش B اجازه یابد که این خواندن را انجام دهد (شکل ۱۶,۲ را نگاه کنید فقط نقش تراکنش A و B با هم جا بجا شده است)، می‌تواند سبب بروز مشکل وابستگی تثبیت نشده گردد، بنابراین می‌توان گفت که تعارض WR باعث مشکل وابستگی تثبیت نشده می‌گردد. نکته : اگر تراکنش B اجازه یابد که خواندنش را انجام دهد یک خواندن ناجور[2] صورت پذیرفته است.
- WW: تراکنش A تاپل t را می‌نویسد و سپس تراکنش B قصد دارد تاپل t را بنویسد. اگر تراکنش B اجازه یابد که این نوشتن را انجام دهد (شکل ۱۶,۱ و ۱۶,۳ را نگاه کنید)، می‌تواند سبب بروز مشکل نتیجه از دست رفته گردد، بنابراین می‌توان گفت که تعارض WW باعث بروز مشکل نتیجه از دست رفته می‌گردد. نکته : اگر تراکنش B اجازه یابد که نوشتنش را انجام دهد یک نوشتن ناجور[3] صورت پذیرفته است.

۳,۱۶ قفل گذاری[4]

همان‌طور که در بخش ۱۶,۱ گفته شد، تمام مشکلات گفته شده در بخش ۱۶,۲ را می‌توان با یک شیوه کنترل هم‌روندی به نام قفل گذاری، برطرف نمود. اساس کار ساده است، هر زمان که تراکنش A نیاز داشته باشد که اطمینان یابد برخی از اشیاء (به طور نمونه یک تاپل پایگاه داده) که مورد نیاز او است تا زمانی که آن‌ها را نیاز دارد

---

[1] Non repeatable read
[2] Dirty read
[3] Dirty write
[4] locking

در برخی حالات تغییر نکنند، یک قفل بر روی آن اشیاء بدست می‌آورد. بدست آوردن قفل بر روی یک شئ داده سبب می‌گردد که «درخواست دیگر تراکنش‌ها راکد گردد» و بنابراین از تغییر آن شی داده جلوگیری می‌گردد. بنابراین تراکنش A قادر است شئ مذکور را در حین پردازش و تا زمانی که به آن نیاز دارد، آنرا در یک وضع مناسب نگه دارد.

حال جزئیات بیشتری در مورد اینکه چطور کار خواهد کرد.

۱. اول، سیستم از دو نوع قفل، قفل انحصاری[1] (X locks) و قفل اشتراکی[2] (S locks) پشتیبانی می‌کند، که در دو پاراگراف بعدی اطلاعات بیشتر آورده شده است. توجه : قفل‌های X و S بعضی مواقع به ترتیب قفل نوشتن و قفل خواندن نامیده می‌شوند. تا زمانی که اطلاعات بیشتر داده نشده، فرض به این است که قفل‌های انحصاری و اشتراکی (S And X locks) تنها انواع قفل موجود است. بخش ۱۶,۹ امکانات دیگر را مورد بحث قرار می‌دهد. همچنین تا زمانی که گفته نشده، تاپل تنها چیزی است (واحد قفل شدنی است) که می‌تواند قفل شود، دوباره در بخش ۱۶,۹ امکانات دیگری بحث می‌گردد.

۲. اگر تراکنش A بر روی تاپل t یک قفل انحصاری (X) داشته باشد، پس یک قفل از هر نوع (چه اشتراکی و چه انحصاری) نمی‌تواند به تراکنش مجزای B بر روی تاپل t اعطا گردد.

۳. اگر تراکنش A بر روی تاپل t یک قفل اشتراکی (S) داشته باشد، پس:

---

[1] Exclusive lock
[2] Shared lock

- اگر از طرف تراکنش مجزای B یک قفل X بر روی تاپل t درخواست شود، این درخواست فوراً برآورده نمی‌گردد.
- اگر از طرف تراکنش مجزای B یک قفل S بر روی تاپل t درخواست شود، این درخواست می‌تواند فوراً برآورده گردد. (و اکنون تراکنش B نیز یک قفل اشتراکی بر روی تاپل t خواهد داشت).

	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

شکل ۱۶,۵. ماتریس همانندی برای قفل‌های نوع X و Y

این قواعد به راحتی می‌تواند توسط یک ماتریس همانندی[1] خلاصه شود (شکل ۱۶,۵). این ماتریس به این صورت تفسیر می‌شود: تاپل t را در نظر بگیرید، فرض کنید تراکنش A در حال حاضر یک قفل بر طبق مقادیر عناوین موجود در ستون جدول (خط تیره = بدون قفل) بر روی تاپل t گذاشته است و تراکنش B یک درخواست برای قفل گذاری طبق مقادیر موجود در طرف چپ جدول روی تاپل t عنوان کرده است. حال مقدار موجود در تلاقی سطر ستون ماتریس همانندی مشخص می‌نماید که آیا تراکنش B می‌تواند بر روی تاپل t قفل بگذارد و قفل‌های دو تراکنش با یکدیگر سازگاری[2] دارند (مقدار تلاقی سطر و ستون برابر "Y" است) و یا قفل‌های دو تراکنش با یکدیگر تضاد[3] خواهند داشت (مقدار تلاقی سطر و ستون برابر "N"

---

[1] Compatibility matrix
[2] compatibility
[3] conflict

است) و با درخواست تراکنش B نمی‌توان موافقت نمـود. ایـن مـاتریس کـاملاً متقـارن است.

حال، با استفاده از قفل‌های X و S یک پروتکـل دسترسـی داده[1] یـا پروتکـل قفل گذاری[2] را معرفی می‌کنیم که ضمانت می‌کند که مشکلاتی که در بخش ۱۶,۲ گفته شد اتفاق نیفتند.

۱. تراکنشی که قصد دارد یک تاپل را بازیابی کند ابتدا باید یک قفـل S بر روی آن تاپل بگذارد.
۲. تراکنشی که قصد دارد یک تاپل را به هنگام کند ابتدا باید یـک قفـل X بر روی آن تاپل بگذارد. اگر تراکنش T در حال حاضـر بـر روی تاپل، قفل S دارد و بخواهد پس از بازیابی، عمـل بـه هنگـام سـازی انجام دهـد بایـد قفـل S را بـه X تقویـت[3] نمایـد. نکتـه: در اینجـا مسئله‌ای را تشریح می‌کنیم که درخواست برای قفل گـذاری معمـولاً ضمنی انجام می‌پذیرد. یک عمل «بازیابی تاپل» به طور ضمنی یـک قفل S را بر روی تاپل مربوطه درخواست می‌کند و یـک عمـل «بـه هنگام سازی تاپل» به طـور ضـمنی یـک قفـل X را بـر روی تاپـل مربوطه درخواست می‌کند (یا به طور ضمنی درخواست تقویت قفـل را از S به X می‌کند). همچنین اصطلاح به هنگام سازی شامل عمـل درج و حذف می‌باشد.
۳. اگر درخواست قفل برای تراکنش B بدیل آنکه با قفل تراکنش A کـه در حال حاضر وجود دارد متضاد است، درخواست تـراکنش B فـوراً برآورده نمی‌گردد و این تراکنش به حالت انتظار[4] می‌رود. تراکنش B تا زمانی که درخواستش برآورده نشود در حالت انتظـار مـی‌مانـد و

---

[1] Data access protocol
[2] Locking protocol
[3] upgrade
[4] Waite state

این درخواست تا زمانی که تراکنش A قفل گشایی نکند، برآورده نخواهد شد.

٤. قفل‌های X در انتهای تراکنش (COMMIT یا ROLLBACK) آزاد می‌شوند. قفل‌های S نیز در انتهای تراکنش آزاد می‌شوند (حداقل ما تا بخش ١٦,٨ این‌چنین فرض می‌کنیم).

پروتکلی که تشریح شد پروتکل قفل گذاری دو مرحله‌ای سخت[1] نام دارد. در بخش ١٦,٦ بحث بیشتری در این مورد خواهیم نمود و همچنین خواهیم گفت چرا این پروتکل دو مرحله سخت نامیده شده است.

٤,١٦ نگاهی دیگر به سه مشکل هم‌روندی

اکنون در موقعیتی هستیم که ببینیم چطور پروتکل قفل گذاری دو مرحله‌ای سخت سه مشکلی که در بخش ١٦,٢ گفته شد را حل می‌کند. در این لحظه دوباره آن‌ها را مدنظر قرار می‌دهیم.

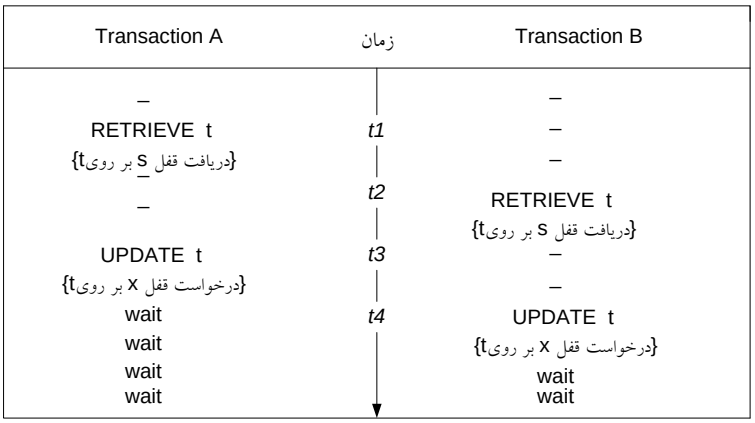
مشکل نتیجه از دست رفته

شکل ١٦,٦ یک نسخه اصلاح شده از شکل ١٦,١ است که نشان می‌دهد در میان اجرای آن شکل تحت پروتکل قفل گذاری دومرحله‌ای ای سخت (2pl سخت) چه اتفاقی می‌افتد. به هنگام سازی تراکنش A در زمان t_3 پذیرفته نمی‌شود، زیرا آن به طور ضمنی یک قفل X بر روی t درخواست کرده که این درخواست با قفل S که در حال حاضر تراکنش B بر روی t دارد در تعارض است، و بنابراین تراکنش A به حالت انتظار می‌رود. به دلیل مشابه تراکنش B در زمان t_4 به حالت انتظار می‌رود. حال هر دو تراکنش نمی‌توانند پیشرفت کنند، تردیدی نیست که مشکل نتیجه از دست رفته وجود ندارد. بنابراین توانستیم مشکل نتیجه از دست رفته با تبدیل کردن آن به مشکل دیگر

---

[1] Strict two-phase locking

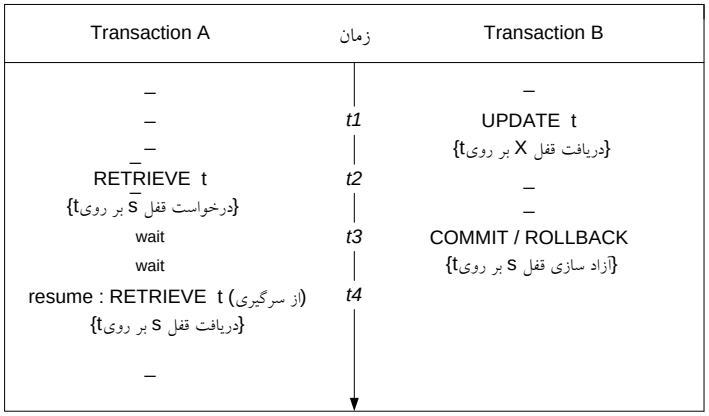
حل کنیم! ولی حداقل مشکل اصلی حل شده است. این مشکل جدید **بن بست** نام دارد که در بخش بعدی توضیح داده شده است.



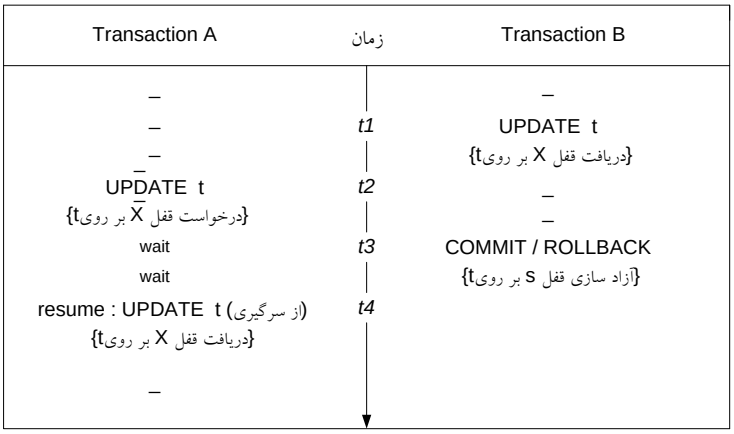
شکل ۱۶،۶. مشکل نتیجه از دست رفته رفع شده، اما در زمان t_4 بن بست رخ داده است.

مشکل وابستگی تثبیت نشده

شکل‌های ۱۶،۷ و ۱۶،۸ به ترتیب نسخه اصلاح شده از شکل‌های ۱۶،۲ و ۱۶،۳ می‌باشند که نشان می‌دهند در میان اجرای آن شکل‌ها تحت پروتکل قفل گذاری دومرحله‌ای ای سخت (2pl سخت) چه اتفاقی می‌افتد. عملیات تراکنش A در زمان t2(RETRIVE در شکل ۱۶،۷ و UPDATE در شکل ۱۶،۸) در هر دو حالت پذیرش نمی‌شود زیرا آن تراکنش به طور ضمنی یک درخواست برای یک قفل بر روی t انجام داده و این درخواست قفل با قفل X که در حال حاضر توسط تراکنش B بر روی t وجود دارد در تعارض است و بنابراین تراکنش A به حالت انتظار می‌رود و تا زمانی که تراکنش B به انتها نرسد.



شکل۱۶,۷. تراکنش A از دیدن یک تغییر تثبیت نشده در زمان t2 منع شده است.

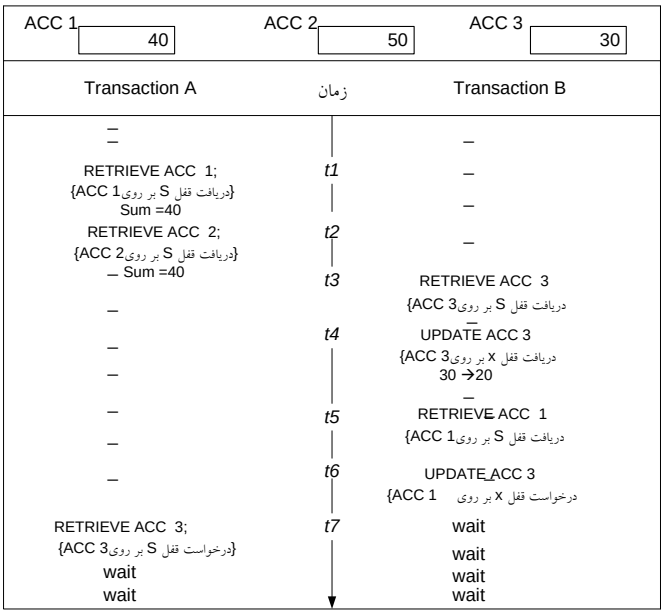


شکل ۱۶,۸. تراکنش A از دیدن یک تغییر تثبیت نشده در زمان t2 منع شده است.

COMMIT یا ROLLBACK در این حالت باقی می‌ماند، زمانی که قفلی که توسط تراکنش B گذاشته شده بود آزاد گردد تراکنش A قادر است که ادامه یابد و در آن زمان تراکنش A یک مقدار تثبیت شده را می‌بیند (در صورتی که تراکنش B تثبیت شده باشد مقدار بعد از اجرای تراکنش B و در صورتی که واکرد شده باشد مقدار قبل از اجرای تراکنش B در دسترس تراکنش A خواهد بود). به عبارت دیگر تراکنش A دیگر به یک به هنگام سازی تثبیت نشده وابسته نخواهد بود و بنابراین مشکل اصلی حل شده است.

مشکل تحلیل ناسازگار

شکل ۱٦,۹ یک نسخه اصلاح شده از شکل ۱٦,٤ است که نشان می‌دهد در میان اجرای آن شکل تحت پروتکل قفل گذاری دومرحله‌ای ای سخت (2pl سخت) چه اتفاقی می‌افتد. به هنگام سازی تراکنش B در زمان t_6 مورد قبول قرار نمی‌گیرد زیرا آن به طور ضمنی درخواست یک قفل X را روی ACC1 کرده که این درخواست با قفل S بر روی ACC1 که در حال حاضر متعلق به تراکنش A است در تعارض است و بنابراین تراکنش B به حالت انتظار می‌رود. همچنین بازیابی تراکنش A در زمان t_7 مورد قبول قرار نمی‌گیرد زیرا آن تراکنش به طور ضمنی درخواست یک قفل S بر روی ACC3 را کرده که این درخواست با قفل X که در حال حاضر تراکنش B بر روی ACC3 دارد در تعارض است و بنابراین تراکنش A به حالت انتظار می‌رود. بنابراین دوباره با ایجاد بن بست، مشکل اصلی (در اینجا مشکل تحلیل ناسازگار) حل شد.



شکل ۱٦,۹. از تحلیل ناسازگار جلوگیری می‌شود، اما در زمان t_7 بن بست رخ می‌دهد.

۵,۱۶ بن بست[1]

دیدیم که چطور با استفاده از قفل گذاری می‌توان سه مشکل هم‌روندی را حل نمود. اما متأسفانه قفل گذاری مشکلات خاص خود را دارد. که اساسی‌ترین آن‌ها مشکل بن بست است. دو مثال از بن بست در بخش قبل ارائه گردید. شکل ۱۶,۱۰ یک نسخه کلی‌تر از مسئله را نشان داده است: در این شکل منظور از r_1 و r_2، نه فقط تاپل های پایگاه داده بلکه هر منبع قابل قفل شدن می‌باشد. پس منظور از عبارت «قفل انحصاری» هر عملیاتی است که چه به طور ضمنی و چه به طور غیر ضمنی درخواست قفل X را داشته باشد.

به طور کلی بن بست موقعیتی است که در آن دو یا چند تراکنش هم‌زمان در حالت انتظار باشند و هر یک از آن‌ها منتظر است تا یکی دیگر قفل را آزاد سازد. شکل ۱۶,۱۰ یک بن بست را نشان می‌دهد که در آن دو تراکنش دخیل هستند اما بن بست با مشارکت سه، چهار یا بیشتر تراکنش نیز امکان پذیر است. به هر حال تجربه با سیستم R نشان داد که در عمل در بن بست هرگز بیش از دو تراکنش دخیل نیست.

Transaction A	زمان	Transaction B
–		–
LOCK r1 EXCLUSIVE	*t1*	–
–		–
–	*t2*	LOCK r2 EXCLUSIVE
–		–
LOCK r2 EXCLUSIVE	*t3*	–
wait		–
wait	*t4*	LOCK r1 EXCLUSIVE
wait		wait

شکل ۱۶,۱۰. یک مثال از بن بست

زمانی که بن بست رخ می‌دهد، مطلوب آن است که سیستم آن را تشخیص داده و آن را بشکند (بر طرف کند). کشف بن بست مستلزم کشف دور[2] در گراف

---

[1] Dead Lock
[2] Cycle

انتظار[1] است. لازمه شکستن بن بست این است که یکی از تراکنش‌هایی که به وجود آورنده بن بست است را انتخاب نموده و آن را قربانی کرد و تمام کارهایی که انجام داده است را خنثی سازیم بدین وسیله توانستم قفل را (حلقه در گراف انتظار) آزاد سازیم و بنابراین اجازه می‌دهیم دیگر تراکنش‌ها به اجرای خود ادامه دهند. نکته : در عمل همه سیستم‌ها کشف بن بست را انجام نمی‌دهند؛ برخی از یک شیوه مهلت زمانی استفاده می‌کنند و به این شکل عمل می‌کنند که تراکنشی که در چند دوره زمانی هیچ کاری انجام نداده در بن بست قرار دارد.

ضمناً در نظر داشته باشید که تراکنش قربانی دچار نقص شده و باید تمام کارهایی که انجام داده خنثی گردد که دیگر اثری از خطاهای آن وجود نداشته باشد. برخی سیستم‌ها به طور خودکار این قبیل تراکنش‌ها را دوباره از ابتدای تراکنش اجرا می‌کنند و فرض را بر این می‌گذارند که دیگر شرایطی که باعث بن بست شده دیگر رخ نخواهد داد. بندرت برخی از سیستم‌ها یک کد استثناء[2] «قربانی بن بست[3]» را به برنامه کاربردی باز می‌گردانند، سپس برنامه کاربردی است که با توجه به موقعیت شیوه برخورد مطلوب را اتخاذ می‌کند. از دید برنامه نویس برنامه کاربردی، اولین روش بهتر است. اما حتی اگر بعضی مواقع نیز برنامه نویس درگیر این موضوع شود همیشه بهتر است که این مشکل از دید کاربر نهایی مخفی نگه داشته شود.

اجتناب از بن بست[4]

بجای اینکه اجازه دهیم بن بست رخ داده و سپس با آن برخورد کنیم (اکثر سیستم‌ها این کار را انجام می‌دهند)، این امکان هم وجود دارد که با اصلاح پروتکل قفل گذاری از طرق مختلف، به طور کلی از بروز بن بست جلوگیری کنیم. در اینجا به طور مختصر یکی از روش‌های ممکن را بررسی می‌کنیم، این روش در دو نسخه به

---

[1] Wait-for graph
[2] Exception Code
[3] Deadlock victim
[4] Deadlock Avoidance

نام‌های *انتظار–مرگ*[1] و *برنده–انتظار*[2] تدوین شده است. این رهیافت بدین صورت کار خواهد نمود:

- هر تراکنش بر حسب زمان شروع، یک مهر زمانی[3] دارد (که باید یکتا باشد).
- زمانی که تراکنش A یک درخواست قفل بر روی چند تایی[4] را دارد که در حال حاضر توسط تراکنش B قفل است. پس
  - **انتظار– مرگ** : اگر تراکنش A پیرتر از تراکنش B باشد (مهر زمانی آن کوچک‌تر باشد) صبر می‌کند در غیر این صورت A می‌میرد (یعنی تراکنش A بازگشت می‌کند و دوباره از ابتدا شروع می‌کند).
  - **برنده– انتظار**: اگر تراکنش A از تراکنش B جوان تر باشد (مهر زمانی بیشتر باشد) منتظر می‌ماند و در غیر این صورت تراکنش A تراکنش B را می‌کشد (یعنی تراکنش B بازگشت کرده و دوباره از ابتدا شروع می‌کند).
- اگر یک تراکنش دوباره از ابتدا شروع کند مهر زمانش همان مهر زمان قبلی است.

نکته: اولین قسمت از نام روش (انتظار یا برنده) در هر حالت بر آنچه که برای تراکنش A اتفاق می‌افتد، دلالت دارد. همان‌طور که مشاهده می‌کنید اگر تراکنش A از تراکنش B بزرگ‌تر باشد، انتظار– مرگ به معنی آن است که همه تراکنش‌های قدیمی‌تر برای تراکنش‌های جوان تر صبر پیشه می‌کنند، برنده–انتظار به معنی آن است که همه

---

[1] Wait-die
[2] Wound wait
[3] Timestamped
[4] Tuple

تراکنش‌های جوان تر برای تراکنش‌های بزرگ‌تر صبر پیشه می‌کنند. هر یک از این نسخه‌ها کارا بوده و به راحتی می‌توان دید که از بن بست جلوگیری می‌کند. همچنین به راحتی می‌توان دید که هر تراکنش سرانجام درست به پایان می‌رسد (یعنی مشکل قفل زنده[1] وجود ندارد و هیچ تراکنشی برای همیشه شروع مجدد نخواهد داشت). مشکل اصلی این رهیافت (در هر دو نسخه) این است که بازگشت[2] بسیار زیادی انجام می‌دهد.

٦,١٦ توالی پذیری[3]

توالی پذیری معمولاً به عنوان «ملاکی برای درستی» اجرای تودرتو یک مجموعه از تراکنش‌ها است. یعنی این اجرا تودرتو درست همان نتیجه را ایجاد می‌کند که اجرای تراکنش‌ها به طور متوالی ایجاد می‌کند. یک طرح اجرا از یک مجموعه تراکنش، توالی پذیر است اگر و فقط اگر معادل (تضمین می‌کند که همان نتیجه حاصل می‌گردد) اجرای متوالی تراکنش‌ها باشد، جایی که:

- یک اجرای متوالی به معنی اجرای پی در پی تراکنش‌ها در یک زمان است.
- تضمین به معنی آنست که طرح اجرای مفروض و طرح اجرای متوالی همیشه فارغ از اینکه وضع اولیه در پایگاه داده چه باشد یک نتیجه را تولید می‌کنند.

این تعریف را به شرح زیر تنظیم می‌کنیم.

---

[1] Live lock
[2] RollBack
[3] serializability

۱. فرض بر این است هر یک تراکنش‌ها، درست هستند. یعنی همان‌طور که در فصل گذشته گفتیم پایگاه داده را از یک وضع درست به وضع درست دیگری منتقل می‌کنند.
۲. اجرای تراکنش‌ها در یک زمان در هر ترتیب متوالی درست است (زیرا فرض بر این است که هر ترتیب متوالی تراکنش‌ها، مستقل از یکدیگر است).
۳. یک طرح اجرای ناپیوسته (هم‌روند) درست است اگر و فقط اگر با برخی طرح اجرای سریال معادل باشد (یعنی: اگر و فقط اگر توالی پذیر باشد). نکته : «فقط اگر» به این نکته اشاره دارد که یک اجرای ناپیوسته (هم‌روند) ممکن است توالی پذیر نباشد اما بسته به وضع اولیه خاص، همان نتیجه که درست است را تولید می‌کند.

بازگردیم به مثال بخش ۱۶,۲(شکل ۱۶,۱ تا ۱۶,٤)، می‌توان دید که مشکل در هر حالت این است که اجرای ناپیوسته، توالی پذیر نیست. یعنی اینکه آن طرح اجرا هرگز معادل اجرای تراکنش A و سپس تراکنش B و یا اجرای تراکنش B سپس تراکنش A نیست. مطالعه بخش ۱٦,٤ نشان می‌دهد که تأثیر پروتکل قفل گذاری دو مرحله‌ای محض[1] این است که توالی پذیری را در هر حالت تأمین می‌نماید. در شکل‌های ۱٦,٦ و ۱٦,۹ اجرای ناپیوسته معادل اجرای تراکنش A و سپس تراکنش B است. در شکل ۱٦,٦ و ۱٦,۹ رخ دادن یک بن بست باعث می‌شود که یکی از دو تراکنش برگشت داده شود (و احتمالاً بعداً دوباره اجرا گردد). اگر تراکنش A تراکنشی باشد که برگشت (rollBAck) می‌کند، پس اجرای ناپیوسته معادل اجرای تراکنش A و سپس اجرای تراکنش B می‌گردد.

---

[1] Strict Two-Phase Locking

اصطلاحات: یک مجموعه از تراکنش‌ها مفروض است. هر اجرایی از این تراکنش‌ها چه ناپیوسته یا متوالی، یک زمان بندی[1] نامیده می‌شود. اجرای به نوبت تراکنش‌ها بدون هیچ ناپیوستگی، طرح اجرای متوالی[2] نام دارد. طرح اجرایی که متوالی نباشد ناپیوسته یا هم‌روند است. دو طرح اجرا معادل است اگر و فقط اگر، بدون اهمیت دادن به اینکه وضعیت اولیه پایگاه داده چه باشد، هر دو یک نتیجه را تولید کنند. بنابراین یک طرح اجرا، توالی پذیر و درست است اگر و فقط اگر معادل یک طرح اجرای متوالی باشد.

نکته اینکه دو طرح اجرای متوالی از یکسری تراکنش‌ها دو نتیجه مختلف را تولید می‌کنند و از این رو دو طرح هم‌روند همان تراکنش‌ها نیز نتایج مختلفی تولید می‌کنند؛ و هر دو درست هستند. برای مثال فرض کنید که تراکنش A یک واحد به x اضافه می‌کند (X+1) و تراکنش B، x را دو برابر می‌کند (2*X) (جایی که x یک آیتم در پایگاه داده است) فرض کنید که مقدار اولیه x برابر ۱۰ است. پس نتیجه طرح اجرای متوالی تراکنش‌ها به صورت اول A بعد B برابر x=22 است. در حالی که اجرای متوالی تراکنش‌ها به صورت اول B بعد A حاصل x=21 است. هر دو نتیجه به یک میزان درست است و هر طرح اجرایی که ضمانت کند معادل اجرای A سپس B و یا B سپس A باشد نیز درست است.

مفهوم توالی پذیری اولین بار توسط Eswaran و همکاران مطرح شد. همچنین در همان مقاله اثبات یک قضیه مهم که قضیه قفل گذاری دو مرحله‌ای[3] نامیده می‌شود آمده است که به شرح زیر است.

اگر همه تراکنش‌ها از پروتکل قفل گذاری دو-مرحله‌ای تبعیت نمایند، پس همه طرح اجراها هم‌روند توالی پذیر می‌باشند.

پروتکل قفل گذاری دو مرحله‌ای به شرح زیر است.

---

[1] schedule
[2] Serial schedule
[3] Two-Phase Locking

- قبل از هر عملی بر روی هر شئ (برای مثال: چند تایی پایگـاه داده)، یک تراکنش باید یک قفل بر روی آن شئ دریافت کند.
- بعد از قفل گشایی، یک تراکنش نباید هیچ قفـل دیگـری را دریافـت نماید.

بنابراین یک تراکنش که از این پروتکل تبعیت کنـد، دارای دو مرحلـه اسـت. یک مرحله دریافت قفل (قفل گذاری) مرحله بسط یا رشد[1] و مرحله قفـل گشـایی یـا مرحله کوچک شدن[2]. نکته: در عمل مرحله قفل گشایی اغلب در یـک عمـل منفـرد از تثبیت (Commit)یا بازگشت (RollBAck) در انتهای تراکنش صـورت مـی‌گیـرد. (در بخش‌های ۱٦,۷ و ۱٦,۷ باز به این موضوع خواهیم پرداخت). اگر این حالت کـه گفتـه شد انجام شود پس پروتکل نسخه محض[3] است که در بخش ۱٦,۳ تشریح شد.

l را یک طرح اجرای هم‌روند از مجموعه تراکنش‌های $T_1,T_2...T_n$ در نظـر می‌گیریم، اگر l توالی پذیر باشد پس حداقل یـک طـرح اجـرای متـوالی s از مجموعـه تراکنش‌های $T_1,T_2...T_n$ وجود دارد که l معادل s است. به S یک **سریال سـازی**[4] از l می‌گویند.

حال Ti و Tj را دو تراکنش جدا در مجموعه تـراکنش‌هـای$T_1,T_2...T_n$ در نظر می‌گیریم. Ti را جلوتر از Tj در سریال سازی S در نظر می‌گیـریم. پـس در طـرح هم‌روند l نیز در واقع Ti جلوتر از T_j اجرا می‌گردد. به عبارت دیگر در یک بیان غیر رسمی اما مفید از توالی پذیری این است که اگـر تـراکنش A و B دو تـراکنش در یـک طرح اجرای توالی پذیر باشند، در طرح اجرا، منطقی یا تراکنش A بر B مقدم است و یا تراکنش B بر A و این به معنی آن است که یا تراکنش B می‌تواند خروجـی تـراکنش A را ببینـد و یـا تـراکنش A مـی‌توانـد خروجـی تـراکنش B را ببینـد (اگـر تـراکنش A،

---

[1] Growing phase
[2] Shrinking phase
[3] Strick Two-Phase Locking
[4] serialisation

x,y,...z را به عنوان خروجی تولید کند و تراکنش B، هر یک از x,y,...z را به عنوان ورودی ببیند، پس تراکنش B یا همه آن‌ها را بعد از اینکه توسط تراکنش A تولید شده‌اند می‌بیند و یا قبل از آنکه توسط تراکنش A به عنوان خروجی تولید شوند، و نه ترکیبی از این دو حالت). اگر نتیجه اجرا به این صورت که تراکنش A قبل از تراکنش B و یا تراکنش B قبل از تراکنش A نباشد پس طرح اجرا توالی پذیر نخواهد بود.

سرانجام بر این نکته تاکید می‌کنیم که اگر تراکنش A از پروتکل قفل گذاری دو مرحله‌ای تبعیت نکند پس همیشه این امکان وجود دارد که تراکنش B به طور هم‌روند با تراکنش A اجرا گردد و یک طرح اجرا ایجاد کند که توالی پذیر نبوده و نادرست است. حال به جهت کاهش رقابت بر سر منابع و در نتیجه افزایش کارایی و بازدهی، سیستم‌های دنیای واقعی اجازه می‌دهند تراکنش‌ها از پروتکل قفل گذاری دو مرحله‌ای تبعیت نکنند این بدین معنی است که تراکنش‌ها می‌توانند زودتر (قبل از تثبیت) قفل گشایی کنند و درخواست قفل گذاری کنند. به هر حال، این شاید واضح باشد که خطر این طرح زیاد است، منت‌های مراتب اگر به تراکنش A اجازه داده شود که از پروتکل قفل گذاری دو مرحله‌ای تبعیت نکند این احتمال وجود دارد که تراکنش B که با تراکنش A در سیستم است در کار یکدیگر مداخله کنند (اگر این چنین باشد پس سیستم روی هم رفته بالقوه می‌تواند جواب اشتباه تولید کند).

۷,۱۶ بازبینی در ترمیم

بدیهی است که در یک طرح اجرای متوالی، تراکنش‌ها ترمیم پذیر اند. یک تراکنش در صورتی که لازم باشد همیشه می‌تواند با استفاده از تکنیک‌هایی که در فصل گذشته تشریح شد، دوباره اجرا و بی اثر گردد (redo و undo). اما مشخص نیست اگر به تراکنش‌ها اجازه دهیم به طور هم‌روند اجرا شوند، هنوز تراکنش‌ها ترمیم پذیر باشند.

در حقیقت مشکل وابستگی تثبیت نشده[1] که در بخش ۱۶,۲ بحث شد می‌تواند سبب مشکل عدم ترمیم پذیری گردد.

فرض کنید که مانند قبل (بخش ۱۶,۲) هیچ پروتکل قفل گذاری نداریم و از این رو در این حالت تراکنش‌ها هرگز برای دریافت قفل صبر نمی‌کنند. حال شکل ۱۶,۱۱ که یک نسخه اصلاح شده از شکل ۱۶,۲ است (با این تفاوت که اکنون تراکنش A قبل از اینکه تراکنش B برگشت داده شود، تثبیت شده است) را در نظر بگیرید. مشکل اینجا است برای اینکه تراکنش B برگشت داده شود و حالتی را ایجاد کند که اصلاً اجرا نشده، لازم است که تراکنش A نیز برگشت داده شود، زیرا تراکنش A نتیجه به هنگام سازی تراکنش B را دیده است. اما بازگشت تراکنش A غیرممکن است زیرا A هم اکنون تثبیت شده است. بنابراین طرح اجرا نشان داده شده در شکل ترمیم پذیر نیست.

یک شرط کافی برای آنکه یک طرح اجرا ترمیم پذیر باشد بدین صورت است:

اگر تراکنش A نتیجه هر به هنگام سازی تراکنش B را ببیند، پس تراکنش A باید قبل از اینکه تراکنش B خاتمه پذیرد، تثبیت نشود.

روشن است که می‌خواهیم مکانیزم کنترل هم‌روندی (یعنی از قفل گذاری استفاده می‌کنیم)، پروتکل قفل گذاری ترمیم پذیری را برای همه طرح‌های اجرا ضمانت نماید.

به هر حال آنچه که گفته شد پایان ماجرا نیست. فرض کنید اکنون در اینجا با یک پروتکل قفل گذاری کار می‌کنیم که پروتکل قفل گذاری 2pl محض نیست. بر همین اساس یک تراکنش می‌تواند قبل از خاتمه اقدام به قفل گشایی کند. حال شکل ۱۶,۱۲ را که نسخه تغییر یافته از شکل ۱۶,۱۱ است را نگاه کنید (تفاوت در اینجاست که اکنون تراکنش A قبل از خاتمه تراکنش B، نمی‌تواند تثبیت شود، اما تراکنش B

---

[1] Uncommited Dependency

می‌تواند زود) زودتر از حد معمول از زمان t قفل گشایی کند. بر طبق شکل ۱۶,۱۱ اگر بخواهیم بنا به درخواست rollback تراکنش B پایگاه داده به وضعی بازگردد که انگار تراکنش B هیچ‌گاه اجرا نشده لازم است که تراکنش A نیز برگشت داده شود (rollback) زیرا تراکنش A در حین اجرا، یک به هنگام سازی تراکنش B را دیده است. تراکنش A می‌تواند بازگشت داده شود زیرا هنوز تثبیت نشده است اما **بازگشت آبشاری**[1] تقریباً همیشه مطلوب نیست. به خصوص اگر به یک تراکنش در یک زمان بندی اجازه دهیم بازگشت داده شود، این کار سبب بازگشت دیگر تراکنش‌ها می‌شود، پس لازم است که آمادگی برخورد با «زنجیرهای آبشاری[2]» با طول دلخواه را داشته باشیم. به عبارت دیگر، مشکل موجود در طرح اجرا این شکل نشان می‌دهد که آن فاقد آبشار[3] نیست.

یک شرط کافی برای آنکه یک طرح اجرا فاقد آبشار باشد این است که:

اگر تراکنش A هر به هنگام سازی تراکنش B را ببیند پس تراکنش A نباید قبل از تراکنش B خاتمه پذیرد.

در قفل گذاری دومرحله‌ای ای محض سقوط آبشاری پیش نمی‌آید (CAscAde-free). که این نشان می‌دهد که چرا این پروتکل در اکثر سیستم‌ها مورد استفاده قرار می‌گیرد. همچنین همان‌طور که قبلاً گفتیم، به سادگی می‌توان نشان داد که هر طرح اجرا که فارغ از آبشار باشد ترمیم پذیر نیز می‌باشد.

۸,۱۶ سطوح جداسازی

سریال پذیری *جداسازی* را در همان مفهومی که در ACID(خصوصیات تراکنش) مطرح شد، تضمین می‌کند. یک نتیجه مستقیم و مطلوب این است که اگر همه طرح‌های اجرا سریال پذیر باشند پس برنامه نویس برنامه کاربردی که در حال نوشتن

---

[1] Cascading rollbacks
[2] Cascade Chains
[3] Cascade-free

کد برای تراکنش A است دیگر نیاز نیست که توجه خود را نسبت به اینکه ممکن است تراکنش دیگری مانند B در همان زمان در سیستم اجرا شود، مبذول دارد. اما او می‌توانند نگران این باشد که پروتکل مورد استفاده برای تضمین توالی پذیری می‌تواند منجر به کاهش میزان هم‌روندی یا بازدهی کلی سیستم در سطوح غیر قابل قبول گردد. بنابراین در عمل سیستم‌ها معمولاً از سطوح مختلف «جداسازی» پشتیبانی می‌کنند (در گیومه زیرا هر سطح کمتر از بیشینه به معنی آن است که تراکنش به طور کامل از دیگر تراکنش‌ها جدا نشده، همان‌گونه که بزودی خواهید دید)

سطح *جداسازی* که بر یک تراکنش اعمال می‌شود میزان تداخل را برای تراکنش مشخص می‌سازد. در واقع تراکنش مذکور چه میزان تحمل اجرای هم‌روند دیگر تراکنش‌ها را دارد. حال اگر بخواهیم توالی پذیری ضمانت گردد، تنها مقدار تداخل که می‌تواند تحمل شود **هیچ**[1] است. به عبارت دیگر در حالت کلی سطح *جداسازی* باید بیشترین مقدار ممکن باشد (در غیر این صورت درستی، ترمیم پذیری و فارغ از تسلسل طرح اجرا، نمی‌تواند ضمانت گردد). اما حقیقت این است که همان‌طور پیش‌تر گفتیم سیستم‌ها معمولاً سطوح جدایی کمتر از بیشینه را پشتیبانی می‌کنند، و در این بخش مختصراً این سطوح را مطرح می‌سازیم.

حداقل پنج سطح مختلف جدایی را می‌توان تعریف نمود، هرچند در SQL استاندارد و DB2 هر یک تنها از چهار سطح جدایی پشتیبانی می‌کند. به طور کلی بیشترین سطح جدایی کم‌ترین تداخل (و کم‌ترین هم‌روندی) را دارد و پایین‌ترین سطح جدایی بیشترین تداخل (و بالاترین هم‌روندی) را دارد. به عنوان توضیح، دو سطح جدایی، cursor stability و Repeatable که در DB2 پشتیبانی می‌شود را در نظر بگیرید. Repeatable, read (RR) بیشینه سطح است، اگر تمام تراکنش‌ها در این سطح عمل کنند، همه طرح‌های اجرا توالی پذیر هستند. در مقایسه تحت cursor stability (CS)(پایداری مکان نما) اگر تراکنش A :

---

[1] none

- قابلیت آدرس دهی[1] به چندگانه t بدست آورد، بنابراین
- یک قفل بر روی چندگانه t بدست آورده و سپس
- بدون به هنگام سازی آن از قابلیت آدرس دهی چندگانه t چشم پوشی کرده، و بنابراین
- قفل را به قفل X ارتقا نداده، پس
- قفل می‌تواند بدون صبر کردن تا انتهای تراکنش، گشوده شود.

اما نکته اینکه الآن تراکنش دیگری مانند B می‌تواند چندگانه t را به هنگام کرده و این تغیرات را تثبیت کند. اگر متعاقباً تراکنش A دوباره بازگردد و به چندگانه t نگاه کند (نکته آن که در اینجا پروتکل قفل گذاری دو مرحله‌ای نقض شده است) می‌بیند که این چندگانه تغییر کرده است، بنابراین ممکن است تحت تأثیر آن پایگاه داده را در یک وضع نادرست ببیند. در مقایسه تحت repeatable, read(RR) تمام قفل‌های چندگانه‌ها تا انتهای تراکنش نگه داشته می‌شوند و بنابراین مشکل ذکر شده رخ نمی‌دهد.

نکات حاصل:

۱. مشکل مذکور تنها مشکلی نیست که تحت CS[2] می‌تواند اتفاق بی افتد. متأسفانه آن این‌طور تلقین می‌کند که RR[3] تنها در این حالت مناسب است اما تقریباً بعید است که یک تراکنش نیاز داشته باشد که به یک چندگانه دو بار مراجعه کند (نگاه کند). در مقابل بحث‌هایی وجود دارد که می‌گوید همیشه انتخاب RR بهتر از CS است. تراکنش که تحت CS اجرا می‌شود قفل گذاری دو مرحله‌ای 2pl نیست و بنابراین (همان‌طور که در گفتار قبل گفته شد) CS نمی‌تواند

---

[1] addressability
[2] cursor stability
[3] Repeatable read

توالی پذیری را ضمانت کند. دلیل استفاده این است کـه، CS نسـبت به RR هم‌روندی بیشتری دارد (احتمالاً اما نه لزوماً).

۲. به نظر می‌رسد این حقیقت که تـوالی پـذیری نمـی‌توانـد تحـت CS ضمانت گردد در عمل درست درک نشده است. بنابراین آن کـه در ادامه می‌آید ارزش تکرار دارد. اگـر تـراکنش T در سـطحی کمتـر از بیشترین سطح جدایی عمل کند، پس نمی‌توانیم ضمانت کنیم که اگر تراکنش T به طور هـم‌رونـد بـا تراکنشـی دیگـر اجـرا گـردد آنگـاه می‌تواند پایگاه داده را از یک وضع درست بـه وضـع درسـتی دیگـر انتقال دهد.

۳. هر نوع پیاده سازی که سطوح مختلف جـدایی را پشـتیبانی مـی‌کنـد معمولاً یکسری امکانات کنترل هم‌روندی صـریح را تهیـه مـی‌بینـد- برای نمونه دستورات قفل گذاری صریح- تا به کاربران (نویسـندگان برنامه‌های کاربردی) اجازه دهد در غیاب ضمانت سیستم ایـن کـاربر باشد که سلامت اجرای تراکنش را ضمانت کند.

در ضمن همان‌گونه که گفته شد در DB2، بیشـترین سـطح جـدایی تحـت repeatable read به دست می‌آید اما متأسفانه، SQL استاندارد از همان واژه repeatable read را در سطح کمتری از جـدایی اسـتفاده کـرده اسـت (در SQL اسـتاندارد، سـطح جدایی RR کمتر از سطح جدایی بیشینه است).

شبه داده[1]

هنگامی که تراکنش‌ها در سطح جدایی کمتر از بیشینه، اجرا مـی‌شـوند ممکـن است مشکل خاصی به وجود آید که این مشکل **شبه داده** نام دارد. مثال زیر را در نظـر بگیرید:

---

[1] phantoms

- ابتدا فرض کنید که تراکنش A میانگین موجودی حساب را برای تمام حساب‌های مشتری Joe حساب می‌کند. فرض کنید که در حال حاضر برای این شخص سه حساب وجود دارد که موجودی هر یک ۱۰۰ دلار است. بنابراین تراکنش A هر سه حساب را بررسی می‌کند، چند قفل اشتراکی برای محاسبه میانگین موجودی حساب‌ها، روی این حساب‌ها گذاشته و نتیجه ۱۰۰ دلار را بدست می‌آورد.
- حال فرض کنید که تراکنش هم‌روند B در حال اجرا است، کار این تراکنش این است که حساب دیگری را برای مشتری Joe با موجودی ۲۰۰ دلار به پایگاه داده اضافه کند. فرض کنید که این حساب جدید بعد از اینکه تراکنش A میانگین حساب‌ها را ۱۰۰ دلار محاسبه کرد به پایگاه داده اضافه می‌کند. همچنین فرض کنید که تراکنش B بلافاصله پس از انجام کار تثبیت می‌شود (قفل انحصاری را که بر روی این حساب جدید داشته آزاد می‌کند).
- اکنون فرض کنید که دوباره تراکنش A تصمیم دارد حساب‌ها را برای مشتری Joe بررسی کند، این حساب‌ها را می شمارش می‌کند و مجموع موجودی حساب‌ها را جمع کرده و بر تعداد آن‌ها تقسیم می‌کند (شاید می‌خواهد ببیند واقعاً میانگین برابر است با مجموع موجودی حساب‌ها تقسیم بر تعداد آن‌ها). این دفعه، تراکنش A می‌بینید به جای سه حساب، چهار حساب وجود دارد و میانگین موجودی به جای ۱۰۰ دلار برابر ۱۲۵ دلار است!

در اینجا هر دو تراکنش از پروتکل قفل گذاری دو مرحله محض[1] پیروی کرده‌اند اما هنوز برخی چیزها اشتباه شده است. در این مثال، تراکنش A چیزی را می‌بیند که بار اول وجود نداشته است- **یک شبه داده**. در نتیجه

---

[1] Strict Two-phase Locking

توالی پذیری نقض شده است (اجرای متداخل با هیچ یک از توالی‌های A سپس B یا B سپس A برابر نیست).

توجه داشته باشید که در اینجا مشکل از قفل گذاری دو مرحله‌ای نیست بلکه مشکل این است که تراکنش باید آنچه را که به طور منطقی نیاز دارد، قفل کند. پس بجای اینکه مانند مثالی که دیدیم سه حساب مشتری Joe را حفظ کند در واقع نیاز دارد مجموعه حساب‌هایی که متعلق به Joe هستند را قفل کند به عبارت دیگر از محصول «دارنده حساب = joe> برای قفل گذاری استفاده کند. اگر تراکنش A این کار را انجام دهد، پس تراکنش B برای اضافه کردن حساب جدید باید صبر کند (زیرا تراکنش B درخواست قفلی را بر روی حساب جدید دارد که این قفل با قفل در دست تراکنش A تعارض دارد).

امروزه سیستم‌های مدیریت پایگاه داده از قفل محصول مانند آنچه که گفته شد، پشتیبانی نمی‌کنند. آن‌ها معمولاً برای جلوگیری از پدیده شبه داده، مسیری دسترسی که مورد استفاده قرار می‌گیرد تا به داده‌های مورد نظر دسترسی پیدا شود را قفل می‌کنند. برای مثال در مورد حساب‌هایی که در اختیار Joe هستند را در نظر بگیرید. اگر مسیر دسترسی با استفاده از شاخص بر روی نام دارنده حساب باشد، پس سیستم می‌تواند مدخلی را در شاخص که متعلق به مشتری Joe است را قفل نماید. این قفل از پدیده شبه داده جلوگیری می‌نماید زیرا ایجاد شبه داده نیازمند مسیر دسترسی است (در این مثال، مدخل شاخص) برای انجام به هنگام سازی است و بنابراین نیازمند بدست آوردن یک قفل X روی مسیر دسترسی است.

۹,۱۶ قفل گذاری قصدی

در تمام مواردی که تا کنون در نظر گرفته‌ایم فرض بر این است که واحد قفل شونده، یک چند تایی از رابطه است. اما به طور کلی، هیچ دلیلی وجود ندارد که قفل

گذاری روی واحدهای بزرگ‌تر یا کوچک‌تری از داده صورت نپذیرد، بـرای مثـال قفـل گذاری روی کل یک رابطه یا حتی روی کل پایگاه داده، یـا (بـر عکـس) قفـل گـذاری روی جزئی از یک چند تایی خاص، که ما آن‌را **دانه بندی قفل**[1] مـی‌نـامیم. دانـه بنـدی کوچک‌تر هم‌روندی را افزایش می‌دهد: دانه بندی درشت باعث می‌گردد که کمتـر نیـاز به قفل گذاری و آزمودن قفل داشته باشیم و در نتیجه سربار کاهش می‌یابد. برای مثـال، اگر تراکنشی یک قفل X روی کل رابطه داشته باشد، دیگر نیاز نیست که روی هر چند تایی منحصر به فرد در درون رابطـه قفـل X بگـذارد، از طرفـی دیگـر، هـیچ تـراکنش هم‌روند دیگری نمی‌تواند هیچ قفلی را بر روی آن رابطـه یـا روی چنـد تـایی‌هـای آن رابطه بدست آورد.

فرض کنید تراکنش T درخواسـت یـک قفـل X بـر روی رابطـه R را دارد، سیستم در هنگام دریافت درخواست تراکنش T، باید قادر باشد تا بگویـد آیـا در حـال حاضر تراکنش دیگری بر روی یک چند تایی از رابطه R، قفل دارد یـا خیـر؟ اگـر ایـن چنین باشد نمی‌توان با درخواست قفل گذاری موافقت نمود. سیستم چگونـه مـی‌توانـد این ناسازگاری را تشخیص دهد؟ روشن است که بررسی اینکه آیا در حال حاضر یـک چند تایی از رابطه R توسط تـراکنش دیگـری قفـل گـذاری شـده و یـا اینکـه از میـان قفل‌های موجود آیا تراکنشی یک چند تایی از رابطـه R را قفـل کـرده اسـت یـا خیـر؟ مطلوب نیست. از این رو پروتکل دیگری را معرفـی مـی‌کنـیم، پروتکـل قفـل گـذاری قصدی[2] که بر طبق آن هیچ تراکنشی نمی‌تواند قفلی را بر روی یک چنـد تـایی بدسـت آورد مگر اینکه قفلی را بر روی رابطه‌ای که حاوی چند تایی است بدسـت آورد باشـد (احتمالاً یک قفل قصدی). حال برای تراکنش‌ها تشـخیص ناسـازگاری قفـل در سـطح رابطه، نسبتاً موضوع ساده‌ای می‌شود.

---

[1] Locking granularity
[2] Intent locking protocol

قبلاً گفته‌ایم که قفل‌های X و S، برای کل رابطه‌ها و نیز برای چند تایی‌های منحصر به فرد استفاده دارند. اکنون سه نوع قفل اضافی دیگر را که قفل‌های قصدی نامیده می‌شوند، معرفی می‌کنیم، که این گونه قفل‌ها نه برای چند تایی‌های بلکه برای رابطه‌ها معنی دارد. این قفل‌ها عبارتند از: قفل‌های قصدی اشتراکی IS[1]، قفل‌های قصدی انحصاری IX[2] و قفل‌های اشتراکی قصدی انحصاری SIX[3]. این نوع قفل‌های جدید به طور غیر رسمی به شرح زیر تعریف می‌شوند (فرض می‌کنیم تراکنش T قفلی از نوع اشاره شده بر روی رابطه R درخواست کرده است) :

- قصدی اشتراکی (IS): تراکنش T قصد دارد که قفل‌های S را روی چند تایی‌های منحصر به فردی از رابطه R بگذارد، برای اینکه تا زمانی که تراکنش پردازش را بر روی چند تایی‌های رابطه شروع نکرده پایداری تضمین گردد.
- قصدی انحصاری (IX): همانند IS است، بعلاوه اینکه تراکنش T ممکن است چند تایی‌های منحصر به فردی را در رابطه T به هنگام کند به همین خاطر قفل X بر روی این چند تایی‌ها می‌گذارد.
- اشتراکی (S): تراکنش T می‌تواند در رابطه R هم‌روندی تراکنش‌های خواننده را تحمل کند، اما هم‌روندی تراکنش‌های به هنگام ساز را تحمل نمی‌کند. (خود تراکنش T نمی‌تواند هیچ چند تایی از رابطه R را به هنگام کند).
- اشتراکی قصدی انحصاری (SIX): ترکیب قفل اشتراکی (S) و قصدی انحصاری (IX) است. یعنی اینکه تراکنش T در رابطه R می‌تواند هم‌روندی تراکنش‌های خواننده را تحمل کند، اما هم‌روندی تراکنش‌های به هنگام ساز را تحمل نمی‌کند، به علاوه، تراکنش T

---

[1] Intended shared
[2] Intent exclusive
[3] Shared intent exclusive

ممکن است چند تایی‌های رابطه R را به هنگام کند و بنابراین قفل‌های X را روی این‌گونه چند تایی‌ها می‌گذارد.

- انحصاری (X): تراکنش T به هیچ وجه نمی‌تواند هیچ دسترسی هم‌روند به رابطه R را تحمل کند. (خود تراکنش T ممکن است چند تایی‌هایی از رابطه R را به هنگام سازد).

تعریفات رسمی از این پنج نوع قفل توسط نسخه گسترش یافته از ماتریس سازگاری این نوع قفل که اولین بار در بخش ۱۶,۳ بحث شد در شکل ۱۶,۱۳ نشان داده شده است.

	X	SIX	IX	S	IS	—
X	N	N	N	N	N	Y
SIX	N	N	N	N	Y	Y
IX	N	N	Y	N	Y	Y
S	N	N	N	Y	Y	Y
IS	N	Y	Y	Y	Y	Y
—	Y	Y	Y	Y	Y	Y

شکل ۱۶,۱۳ ماتریس سازگاری گسترش یافته شامل قفل‌های قصدی

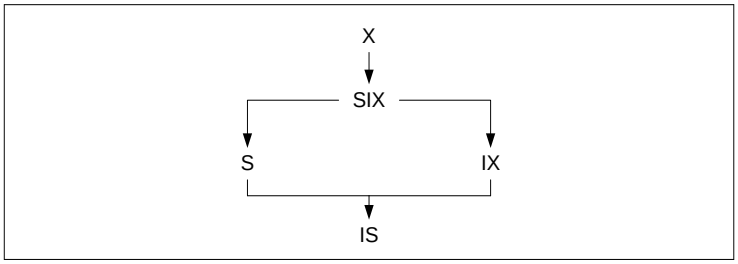
در اینجا پیرامون پروتکل قفل گذاری قصدی توضیح بیشتری ارائه می‌دهیم.

۱. قبل از اینکه تراکنشی بتواند یک قفل S بر روی یک چند تایی بدست آورد، آن تراکنش ابتدا باید بر روی رابطه‌ای که شامل آن چند تایی است یک قفل IS یا قفل قوی‌تر (در ادامه می‌بینید) بدست آورد.
۲. قبل از اینکه تراکنشی بتواند یک قفل X بر روی یک چند تایی بدست آورد، آن تراکنش ابتدا باید بر روی رابطه‌ای که شامل آن چند تایی است یک قفل IX یا قفل قوی‌تر (در ادامه می‌بینید) بدست آورد.

بعضی از این پنج نوع قفل از بعضی دیگر قوی‌تر هستند، قدرت نسبی قفل‌ها در گراف تقدم شکل ۱۶,۱۴ نشان داده شده است. می‌گوییم نوع قفل L2 از نوع قفل L1 قوی‌تر است (در گراف بالاتر است) اگر و فقط اگر، هر گاه در سطری از ستون

L1 ماتریس سازگاری، "N" وجود داشته باشد، در همان سطر از ستون L2 نیز "N" وجود داشته باشد (شکل ۱٦,۱۳ را ببینید). توجه داشته باشد که درخواست قفلی که برای یک نوع قفل رد شود مطمئناً برای یک نوع قفل قوی‌تر رد خواهد شد. همچنین نکته اینکه هیچ یک از دو قفل S و IX نسبت به یکدیگر قوی‌تر نیستند.

این نکته ارزشمند است که، در عمل، معمولاً قفل‌های رابطه نیاز دارند توسط پروتکل قفل گذاری قصدی به طور ضمنی کسب شوند. برای مثال، برای یک تراکنش فقط خواندنی، سیستم احتمالاً یک قفل IS ضمنی روی هر رابطه‌ای که تراکنش به آن دسترسی دارد، ایجاد می‌کند. برای یک تراکنش به هنگام سازی، تراکنش احتمالاً قفل‌های IX را بر روی رابطه ایجاد خواهد کرد. همچنین سیستم احتمالاً دستور قفل گذاری صریحی از برخی انواع قفل دارد، برای اینکه به تراکنش‌ها اجازه دهد در صورت که می‌خواهند، قفل‌های S، X یا SIX را در سطح رابطه بدست آورند. برای مثال، چنین عبارتی توسط DB2 پشتیبانی می‌شود (تنها برای قفل‌های S و X نه قفل‌های SIX).



شکل ۱٦,۱٤. گراف تقدم نوع قفل

این بخش را با اشاره مجدد به گسترش قفل[1] که در بسیاری از سیستم‌ها پیاده سازی شده و تلاشی برای متعادل نمودن نیازمندی‌های ناسازگار هم‌روندی بالا و کاهش سربار مدیریت قفل است، به پایان می‌رسانیم. ایده اصلی این است که زمانی که برخی آستانه‌های تعیین شده سر می‌رسند، سیستم به طور خودکار مجموعه‌ای از قفل‌ها

[1] Lock escalation

با دانه بندی کوچک را با قفلی بـا دانـه بنـدی بـزرگ جـایگزین مـی‌کنـد. بـرای مثـال مجموعه‌ای از قفل‌های S سطح چند تایی را عوض می‌کنـد و قفـل IS بـر روی متغیـر رابطه‌ای حاوی این چند تایی‌ها را به قفل S تبدیل می‌کند.

۱۰,۱۶ نگاهی دوباره به ACID

در فصل ۱۵ بنا شد تا در این فصل خصوصیات ACID تـراکنش‌هـا را مفصـل بررسی کنیم. در حقیقت بیشتر نظرات غیر متعارف در ارتباط بـا ایـن موضـوع را ارائـه می‌کنیم.

ابتدا یادآوری می‌کنیم که ACID مخفف تجزیه ناپذیری[1]، سازگاری[2]، جدایی[3]، ماندگاری[4] که به طور خلاصه در زیر تشریح شده‌اند.

- تجزیه ناپذیری: تراکنش‌های به صورت تجزیه ناپـذیر هسـتند یعنـی همه یا هیچ (یا به کلی اجرا می‌شوند و یا اصلاً اجرا نمی‌شوند).
- درستی (سازگاری) : تراکنش‌ها پایگاه داده‌ها را از یک وضع درسـت به وضع درست دیگری انتقال می‌دهنـد، بـدون آنکـه لزومـاً در اثنـاء اجرای تراکنش درستی را تضمین کند و درسـتی مـی‌توانـد در میـان اجرای تراکنش موقتاً خدشه دار شود.
- جدایی: تراکنش‌هـا از دیگـری جـدا (منفـرد) مـی‌باشـند (مسـتقل از دیگری اجرا می‌شوند). این به معنی است که در حالت کلی بسـیاری از تراکنش به طور هـم‌رونـد در حـال اجـرا مـی‌باشـند و بـه هنگـام سازی‌های هر تراکنش تا زمانی که تثبیت نشده اند از بقیه تراکنش‌ها، پنهان است. به عبارت دیگر برای مثال برای دو تـراکنش جداگانـه A و B ممکن است تراکنش A، به هنگام سازی‌های تـراکنش B را بعـد

---

[1] atomicity
[2] correctness
[3] isolation
[4] durability

از تثبیت B ببیند و یا تراکنش B به هنگام سازی‌های تراکنش A را بعد تثبیت آن ببیند اما نمی‌تواند هر دو آن‌ها را ببیند.

- ماندگاری : یک‌بار که یک تراکنش تثبیت شد، به هنگام سازی‌های آن به طور پایدار در پایگاه داده می‌ماند، حتی اگر پس از اجرا یک خرابی سیستم رخ دهد.

بنابراین ACID مخففی مطلوب است- اما آیا مفاهیمی که آن ارائه می‌دهد واقعاً بر اساس این است که آیا بازرسی می‌تواند در حین اجرای تراکنش متوقف گردد؟ در این بخش به طور کلی یکسری دلیل ارائه دهیم که پاسخ به این سوال خیر می‌باشد.

بررسی بلافاصله محدودیت

با چیزی شروع می‌کنیم که ممکن است یک انحراف به نظر برسد و آن این است که محدودیت‌های جامعیتی باید بلافاصله (یعنی در پایان دستور) بررسی شوند و تا آخر اجرای تراکنش به تعویق نیفتند. ما حداقل چهار دلیل برای قبول کردن این ادعا داریم که اکنون شرح داده می‌شوند.

۱. همان‌طور که می‌دانیم، پایگاه داده می‌تواند به عنوان مجموعه‌ای از گزاره‌ها در نظر گرفته شود. اگر این مجموعه همیشه اجازه داشته باشد که شامل هر ناسازگاری باشد پس تمام شروط بی اثر شده و هرگز نمی‌توانیم به پاسخ‌ها که از پایگاه داده ناسازگار بدست آورده‌ایم، اطمینان کنیم. در حقیقت ما نمی‌توانیم هیچ گاه پاسخ کاملی از چنین پایگاه داده‌ای بدست آوریم. خصوصیت تجزیه ناپذیری یا "I" تراکنش‌ها ممکن است این گونه معنی دهد که بیش از یک تراکنش نمی‌تواند یک ناسازگاری خاص را ببیند، پس آن یک تراکنش ناسازگاری را می‌بیند و بنابراین می‌تواند جواب‌های اشتباه تولید کند. در حقیقت، حتی اگر ناسازگاری‌ها هرگز برای بیش از

یک تراکنش در یک زمان نمایان نشوند نمی‌توانند تحمل شوند، پس لازم است که محدودیت‌ها در همان ابتدا اعمال شوند.

۲. در هر حالت، اینکه یک ناسازگاری تنها توسط یک تراکنش قابل دیدن است نمی‌تواند تضمین شود. تنها در صورتی که اجرای تراکنش‌ها از پروتکل‌های مشخصی پیروی کنند می‌توان تضمین نمود که اجرای تراکنش‌ها جدا از دیگری باشند. برای مثال اگر تراکنش A یک وضع ناسازگار پایگاه داده را ببیند و بنابراین داده‌های ناسازگاری را در فایل F بنویسد، و سپس تراکنش B این اطلاعات را از فایل F بخواند، پس در واقع A و B از یکدیگر جدا نیستند (صرف‌نظر از اینکه آیا آن‌ها به طور هم‌روند اجرا شوند و یا طور دیگر) به عبارت دیگر، خاصیت "I" تراکنش‌ها مشکوک است، حداقل در گفتار.

۳. محدودیت‌های رابطه‌ای بلافاصله بررسی می‌شوند اما محدودیت‌های پایگاه داده در آخر تراکنش بررسی می‌شوند (چیزی که نویسندگان زیادی بر آن اتفاق نظر دارند هر چند که به طرق مختلف بازگو می‌کنند). اما اصل قابلیت تبادل[1] (از رابطه‌های مبنا یا مشتق شده) بر این دلالت دارند که بسیاری از محدودیت‌های (قواعد) دنیای واقعی ممکن است محدودیت رابطه با یک طراحی برای پایگاه داده باشد و محدودیت پایگاه داده با دیگری. از این رو محدودیت‌های رابطه باید کاملاً بلافاصله بررسی شوند، همچنین به دنبال آن نیز محدودیت‌های پایگاه داده باید بلافاصله بررسی شود.

٤. قابلیت انجام «بهینه سازی معنایی» نیازمند این است که پایگاه داده نه تنها در کران‌های تراکنش، بلکه در تمام وقت سازگار و درست

---

[1] Principle of interchangeability

باشد. نکته: بهینه سازی معنایی تکنیکی است برای استفاده کردن از محدودیت‌های جامعیتی برای ساده کردن پرسش‌ها به منظور بهبود کارایی است. روشن است که اگر محدودیت‌ها رعایت نشوند پس ساده سازی نامعتبر خواهد بود.

البته، «دانش متعارف[1]» یعنی بررسی محدودیت پایگاه داده یقیناً به تعویق می‌افتد. به عنوان مثال، فرض کنید در پایگاه داده عرضه کنندگان و قطعات این محدودیت وجود دارد «عرضه کننده S_1 و قطعه P_1 در یک شهر هستند.» اگر عرضه کننده S_1 منتقل شود، مثلاً از لندن به پاریس، پس به همین نحو قطعه P_1 نیز باید از لندن به پاریس منتقل شود. راه حل معمول برای حل این مسئله این است که مانند زیر دو به هنگام سازی را در داخل یک به هنگام سازی بسته بندی کنیم:

```
BEGIN TRANSACTION;
UPDATE S WHERE S# = S#{'S1'} { CITY := 'PAris' } ;
UPDATE P WHERE P# = P#{'P1'} { CITY := 'PAris' } ;
  COMMIT;
```

در راه حل معمول، در زمان تثبیت COMMIT محدودیت بررسی می‌شود، و پایگاه داده در بین دو عمل به هنگام سازی ناسازگار است. توجه: به ویژه اینکه اگر تراکنش در حال انجام به هنگام سازی‌ها بوده و به پرسش «آیا عرضه کننده S_1 و قطعه P_1 در یک شهر هستند» در بین دو عملیات به هنگام سازی پاسخ دهد، پاسخ خیر بدست می‌آید.

به هر حال، یادآوری می‌کنیم که نیازمندیم از عمل گر **انتساب چندگانه** پشتیبانی کنیم. که به ما این امکان را می‌دهد که چندین انتساب را در یک عملیات انجام دهیم (یعنی در یک دستور) بدون آن که تا زمان اتمام انتساب‌های مورد نظر لازم باشد هیچ جامعیتی را بررسی کنیم. همچنین یادآوری می‌کنیم که INSERT، DELETE

---

[1] Conventional wisdom

و UPDATE تنها مختصری برای عملگرهای انتساب هستند. در این مثال قادر خـواهیم بود به هنگام سازی‌های مورد نظر را به عنوان یک عمل گر انجام دهیم بنابراین

```
UPDATE S WHERE S# = S#{'S1'}{CITY ;= 'PAris'},
UPDATE P WHERE P# = P#{'P1'}{CITY ;= 'PAris'};
```

حال تا زمانی که هر دو به هنگام سازی‌ها انجام نگیرد هـیچ جـامعیتی بررسـی نمی‌شود (یعنی تا زمانی که به «» نرسیدیم) همچنین توجه کنید که هـیچ راهـی بـرای دیدن پایگاه داده در وضع ناسازگار بین دو به هنگام سازی وجـود نـدارد، زیـرا اکنـون عبارت «بین دو به هنگام سازی» معنا ندارد.

بر طبق این مثال اگر انتساب چندگانـه پشـتیبانی شـود، لازم نیسـت بـر طبـق مفاهیم قبلی بررسی به تعویق افتد (یعنی بررسی تا آخر تراکنش به تعویق بیفتد).

به هر حال ما خاصیت‌های ACID بررسی می‌کنیم. مناسب تراست کـه آن‌هـا را به ترتیب C-I-D-A بحث کنیم.

درستی

قبلاً دلایل خود را برای اینکه واژه درسـتی را نسـبت بـه واژه بسـیار متـداول سازگاری ترجیح می‌دهیم، بیان کردیم. هر چند که در متون معمولاً این دو مفهوم معادل در نظر گرفته می‌شوند. برای مثال در اینجا نقل قـولی از کتـاب GAry And Reuter آورده‌ایم:

سازگار: درست.

و همین کتاب خصوصیت سازگاری تراکنش را این‌چنین تعریف کرده است:

سازگاری: تراکنش تبدیل درست یک حالت است. عملیات‌هـایی کـه در یـک گروه هستند و هیچ گونه محـدودیت جـامعیتی مـرتبط بـا وضـع پایگـاه داده را نقـض نمی‌کنند. این مستلزم این است که تراکنش یک برنامه درست باشد.

اما اگر محدودیت‌های جامعیتی همیشه بلافاصـله بررسـی شـوند، پایگـاه داده همیشه سازگار است- نه لزوماً درست! - و تـراکنش‌هـا همیشـه پایگـاه داده را از یـک وضع درست به وضع درست دیگری انتقال می‌دهند.

بنابراین اگر C در ACID قرار بر سازگاری باشد پس تا اندازه‌ای این خصوصیت بدیهی است و اگر آن قرار بر درستی باشد پس غیر قابل اجرا است. همان‌طور که قبلاً مشخص شد ما ترجیح می‌دهیم که C دال بر درستی باشد پس می‌توان گفت خصوصیت درستی واقعاً به عنوان یک خصوصیت نیست بلکه بیشتر یک خواست است.

جدایی

اکنون خصوصیت *جدایی* را مطرح می‌کنیم. همان‌طور که در ابتدای این بخش در گفتار «بررسی بلافاصله محدودیت» گفتیم این خصوصیت نیز تا حدودی مشکوک است. حداقل این درست است که اگر هر تراکنش به گونه‌ای رفتار کند که تنها یک تراکنش در سیستم است، پس یک مکانیسم کامل کنترل هم‌روندی، جدایی (و توالی پذیری) را ضمانت خواهد کرد. البته، «رفتار کردن به عنوان تنها یک تراکنش در سیستم»، بر این اشاره دارد که تراکنش مورد بحث باید:

- هیچ تلاش عمدی یا سهوی، برای ارتباط با دیگر تراکنش‌ها، هم‌روند یا غیر هم‌روند انجام ندهد.
- هیچ تلاشی برای تشخیص اینکه ممکن است تراکنش‌های دیگری در سیستم موجود باشند (با مشخص کردن سطح جدایی کمتر از بیشینه) انجام ندهد.

ماندگاری

اکنون به خصوصیت ماندگاری می‌پردازیم. این خصوصیت در سایه مکانیسم ترمیم سیستم قابل قبول است. به شرط آنکه **تراکنش تودرتو**[1] وجود نداشته باشد- در حقیقت ما در این حالت این‌چنین فرض کرده‌ایم. قبل این نکته فرض کنید که تراکنش

---

[1] Nested transaction

تودرتو پشتیبانی می‌شود. به طور مشخص‌تر (برای مثال) فرض کنید تراکنش B درون تراکنش A می‌باشد، و رویدادهای زیر پشت سر هم اتفاق می‌افتد:

```
BEGIN TRANSACTION   (transaction A);
. . .
    BEGIN TRANSACTION (transaction B);
    Transaction B updates tuple t;
    COMMIT (transaction B);
. . .
ROLLBACK (trAnsAction A);
```

اگر بازگشت (ROLLBACK) تراکنش A اتفاق بیفتد، پس تراکنش B نیز پیرو آن بازگشت می‌شود (زیرا تراکنش B در واقع قسمتی از تراکنش A است)، و بنابراین نباید اثرات تراکنش B بر روی پایگاه داده «ماندگار» باشد؛ در حقیقت، بازگشت تراکنش A سبب می‌گردد که مقدار چند تایی t با مقداری که قبل از اجرای تراکنش A داشته، جایگزین گردد. به عبارت دیگر، خصوصیت ماندگاری نمی‌تواند همیشه ضمانت گردد. حداقل برای تراکنش نظیر B که در این مثال داخل تراکنشی دیگر قرار دارد.

در حال حاضر، بسیاری از نویسندگان، همین روش پیشنهاد شده در مثال قبل را قابلیت تراکنش‌های تودرتو پیشنهاد کرده‌اند. برخی منابع ادعا کرده‌اند که این پشتیبانی حداقل به سه دلیل زیر مطلوب است. توازی درون تراکنشی[1]، کنترل ترمیم درون تراکنشی[2]، پیمانه‌ای بودن سیستم[3]. همان‌طور که مثال قبلی نشان می‌دهد که در سیستمی که این‌چنین از تراکنش‌های تو در تو پشتیبانی می‌کند، دستور تثبیت در تراکنش درونی، به هنگام سازی‌ها را تنها برای سطح خارجی بعدی تثبیت می‌کند. در حقیقت تراکنش خارجی حق وتو روی تثبیت تراکنش داخلی دارد- اگر تراکنش خارجی بازگشت انجام دهد، تراکنش داخلی نیز خنثی می‌گردد. در این مثال، تثبیت

---

[1] Intra-transaction parallelism
[2] Intra-transaction recovery control
[3] System modularity

تراکنش B، یک تثبیت برای تراکنش A است، نه تثبیتی برای دنیای خارج، و در حقیقت آن تثبیت پیرو بازگشت تراکنش A خنثی می‌گردد.

خاطر نشان می‌کنیم که تراکنش تودرتو می‌تواند تعمیمی از نقطه نگاهداشت باشد. نقطه نگاهداشت‌ها این اجازه را به تراکنش‌ها می‌دهد تا به عنوان یک توالی خطی از عملیات‌ها در یک زمان اجرا شوند (و بازگشت می‌تواند در هر زمان از شروع اولین عملیات در این توالی رخ دهد). در مقایسه، تودرتو بودن به تراکنش اجازه می‌دهد به طور بازگشتی، به عنوان سلسله مراتبی از عملیات‌ها که می‌توانند به طور هم‌روند اجرا شوند، سازمان دهی شود، به عبارت دیگر:

- BEGIN TRANSACTION برای پشتیبانی از «زیر تراکنش‌ها» اجرا می‌شود. (یعنی اگر BEGIN TRANSACTION زمانی که تراکنش از قبل در حال اجرا است، صادر شود، این دستور یک تراکنش فرزند را شروع کرده است).
- COMMIT اما تنها در ناحیه والد «تثبیت» می‌گردد (اگر این تراکنش فرزند باشد).
- ROLLBACK کار را خنثی می‌سازد، اما تنها به شروع تراکنش خاصی بازمی گرداند. (شامل، فرزند، نوه و غیره، اما تراکنش‌ها شامل تراکنش پدر نمی‌شوند، اگر باشد).

برای بازگشت به مبحث اصلی، اکنون می‌بینیم که خاصیت ماندگاری تراکنش تنها در بیرونی‌ترین سطح تراکنش اعمال می‌گردد (به عبارت دیگر، تراکنش داخل تراکنش دیگری نیست). بنابراین، در حالت کلی، می‌بینیم که خاصیت ماندگاری کمتر از ۱۰۰ درصد ضمانت می‌شود.

تجزیه ناپذیری

سرانجام، به خاصیت تجزیه ناپذیری می‌پردازیم. مانند خاصیت ماندگاری، این خاصیت نیز توسط مکانیزم ترمیم سیستم ضمانت می‌گردد. (حتی با تراکنش‌های

تودرتو). در اینجا هدفمان اندکی متفاوت است، به ویژه، ما حقیقتاً مشاهده می‌کنیم که اگر سیستم از انتساب چندگانه پشتیبانی کند، دیگر نیاز نیست که تراکنش‌ها خاصیت تجزیه ناپذیری را داشته باشند؛ زیرا انتساب‌ها خود این کار را انجام دهند.

١١,١٦ امکانات SQL

SQL استاندارد هیچ امکانات قفل گذاری صریحی را تدارک ندیده است، در حقیقت، اصلاً اشاره‌ای به قفل گذاری نکرده است. اما نیازمند آن است که امکاناتی را برای اجرای متداخل تراکنش‌ها تدارک ببیند. بسیار مهم است که به هنگام سازی‌هایی که توسط تراکنش A صورت پذیرفته تا زمانی که تراکنش A تثبیت نشده قابل مشاهده برای تراکنش مجزای B نباشند. توجه، فرض قبلی که همه تراکنش‌ها در سطح جدایی READ COMMITED، REPEATEBLE READ، یا SERIALIZABLE اجرا شوند. در هنگام اجرای تراکنش در سطح READ UNCOMMITTED (١) به تراکنش اجازه داده می‌شود «خواندن کثیف» انجام دهد، اما (٢) لازم است که تراکنش فقط خواندنی (READ ONLY) باشد (اگر READ WRITE اجازه داده شود قابلیت ترمیم برای همیشه تضمین نمی‌شود).

از فصل پیش به یاد دارید که سطوح جدایی SQL در شروع تراکنش START TRANSACTION مشخص می‌شوند. چهار امکان وجود دارد (SERIALIZABLE، REPEATABLE READ، READ COMMITED، و READ UNCOMMITED). پیش فرض SERIALIZABLE است. اگر هر یک از سه سطوح دیگر مشخص شوند، پیاده سازی از اختصاص سطوح بالاتر معاف است، سطوح بالاتر بدین ترتیب هستند.

REPEATABLE READ > READ COMMITED> READ SERIALIZABLE>UNCOMMITED .

اگر همه تراکنش‌ها در سطح جدایی SERIALIZABLE باشند (طبق پیش فرض) پس ضمانت می‌گردد اجرای متداخل هر مجموعه از تراکنش‌های هم‌روند، توالی پذیر باشد. هرچند، اگر هر تراکنشی در سطح جدایی پایین‌تری اجرا گردد، توالی

پذیری به طرق گوناگونی نقض خواهد شد. تعریف استاندارد سه نوع نقض یعنی خواندن کثیف، خواندن تکرار نشدنی، و شبه داده‌ها (که دو مفهوم اول در بخش ١٦,٢ و مفهوم سوم در بخش ١٦,٨ توضیح داده شد) و سطوح جدایی گوناگون بر حسب نقضی که آن‌ها اجازه دارند در شکل ١٦,١٥ خلاصه شده است ("y" یعنی نقض می‌تواند اتفاق بیفتد، "N" نمی‌تواند اتفاق بیفتد).

Isolation level	Dirty read	Nonrepeatable read	phantom
READ UNCOMMITED	Y	Y	Y
READ COMMITED	N	Y	Y
REPEATABLE READ	N	N	Y
SERIALIZABLE	N	N	N

شکل ١٦,١٥. سطوح جدایی SQL

ما این بخش را با یادآوری این نکته به پایان می‌رسانیم که REPEATABLE READ از SQL استاندارد و "repeatable read"(RR) از DB2 یکسان نیستند. در حقیقت، RR در DB2 مانند SERIALIZABLE در SQL استاندارد است.