پایگاه داده پیشرفته

دکتر شجاعی مهر
علیرضا سلطانی نشان

۲ دی ۱۴۰۲

فهرست مطالب

۱ تراکنش

تراکنش واحد اجرای برنامه است. عملیاتی که در هر تراکنش می‌تواند شامل شود موارد زیر می‌باشد:

- Create
- Read
- Update
- Delete

۲ قوانین ACID

۱.۲ اتمیک یا Atomicity

هر تراکنش دیتابیس به صورت اتمیک می‌باشد. این قضیه بدان معناست که این تراکنش یا باید کاملا انجام شود یا کلا لغو و صرف نظر شود. در غیر این صورت اگر تراکنش به صورت ناتمام و ناقص انجام شود عواقب مختلفی روی دیتابیس خواهد گذاشت.

۲.۲ جامعیت یا Consistency

هر تراکنش باید از قوانین جامعیت پیروی کند. نمی‌توان داده یا را وارد جدولی از دیتابیس کرد که به صورت معتبر نباشد. در برخی از مراجع این قانون را به اجرای صحیح و سازگار تراکنش می‌شناسند. مهم ترین مثال آن است که شما یک Validation روی یک مقداری از فیلد جدول تنظیم می‌کنید که هر داده‌ای بر روی آن فقط با شرایط تعریف شده بایستی وارد شود.

خالی از لطف نیست که در مورد مرجع پذیری داده‌ها در این قسمت نیز می‌توان صحبت کرد تا بتواند قوانین جامعیت را به طور صحیح کامل کرد. مرجع پذیری زمانی مطرح می‌شود که یک رکوردی از داده وقتی وارد جدولی از دیتابیس می‌شود ممکن است ارتباط مشخصی با جدولی دیگر داشته باشد. پس به همین خاطر کلیدهای اصلی و خارجی در خصوص جامعیت وجود دارند که داده‌ای معنادار را پس از پرس و جو از دیتابیس به برنامه نویس برگرداند. یادآوری، بخش جوین‌ها در دیتابیس و تعریف رفرنس در هنگام تعریف کلید جانبی.

۳.۲ انزوا یا Isolation

هر سیستم جامع پایگاه داده‌ای باید بتواند روی همروند تراکنش‌ها مدیریت و کنترل کامل داشته باشد. انزوا تراکنش‌ها قابلیت کنترل و تنظیم بر اساس DBMS است.

به طور کل همروندی یا همزمانی به حالتی گفته می‌شود که چند تراکنش بخواهند در یک زمان به صورت موازی روی یک منبع عملیات خواندن و نوشتن را انجام دهند. اما این عملیات به طور کل هزینه خاص و مشخصی برای برنامه نویس و مدیر دیتابیس دارد.

۴.۲ قابلیت اعتماد یا Duribility

قابلیت اعتماد یکی از مهم‌ترین ویژگی‌های هر سیستم دیتابیسی است. یعنی بتوان داده‌ها را در پایگاه‌داده به صورت پایدار و ثابت نگهداری و مراقبت کرد. در صورت بروز مشکل روی داده‌های یک دیتابیس می‌توان به عملیات انجام شده در این قسمت مراجعه کرد. بطور کلی این بخش قابلیت کنترل و مدیریت دارد و می‌توان مجموعه فرایندهای نگهداری و بک‌آپ را به صورت خودکار انجام داد.

۵.۲ تنظیم قابلیت انزوا

انزوا و مدیریت همروندی در دیتابیس به چهار طریق قابل انجام است:

۱. Read uncommitted

۲. Read commmitted

۳. Repeadable read

۴. Serializable

یادآوری: هر تراکنش دو حالت در پایان پیدا می‌کند:

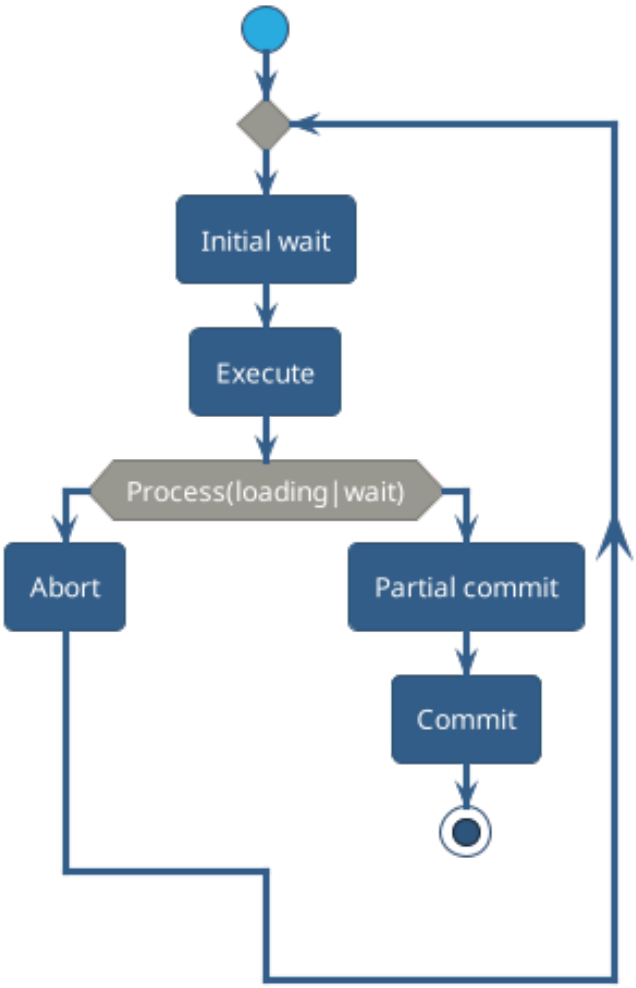
- Commit: تراکنش درنهایت تایید و انجام می‌شود
- Abort: تراکنش در نهایت سقط یا صرفه نظر می‌شود

۱.۵.۲ وضعیت تراکنش

نکته: Abort در دو شرط اتفاق می‌افتد:

۱. زمانی که اجرای تراکنش به خطای Run time دچار شود.

۲. خرابی و نقص سیستم که روی اجرای تراکنش تاثیر می‌گذارد که کامل نشود



شکل ۱: نمودار شروع فرایند تراکنش‌ها

۳ همروندی

۱.۳ مزیت همروندی

۱. افزایش سرعت گذردهی یا throughput

۲. کاهش میانگین زمان پاسخدهی به تراکنش مورد نظر

۲.۳ معایب همروندی

۱. Last update: تغییرات گمشده به دلیل همزمانی در خواندن و نوشتن قانون Write before Write

۲. Uncommitted: خواندن داده‌ای که معتبر نیست. معمولا به آن Dirty read هم گفته می‌شود. قانون Write before Read

۳. Inconsistent retrieval: بازیابی داده‌ای که ناهمگان است. Read before Write

۳.۳ زمان‌بندی

زمان‌بندی به اجرای همروند و همزمان چندین تراکنش با هم گفته می‌شود.

۱.۳.۳ نظریه پی در پی پذیری زمان‌بندی‌ها

به دو روش می‌توان به پی در پی پذیری رسید:

۱. Conflict serializability

۲. View serializability

نمادهای مورد استفاده برای تعریف تراکنش‌ها:

- $R_i|Q|$
- $W_i|Q|$
- $C_i|Q|$
- $A_i|Q|$
- $B_i|Q|$
- $E_i|Q|$

۲.۳.۳ سه شرط اصلی تصادم

اگر p_i و q_j دو تراکنش باشند:

۱. i != j

۲. هر دو به یک داده دسترسی داشته باشند

۳. حداقل یکی از دستورات عمل نوشتن یا write داشته باشد

جدول ۱: حالات تصادم

	$R_i(Q)$	$W_j(Q)$
$R_i(Q)$	ندارد	دارد
$W_j(Q)$	دارد	دارد

۳.۳.۳ زمان‌بندی سریالی

در زمان‌بندی پی در پی، زمانی که یک تراکنش commit یا abort شود به دنبال تراکنش بعدی خواهد رفت که به آن تراکنش سریالی یا Serializable schedule می‌گویند.

$$S_1 = R_1(A)W_1(A)a_1W_2(A)W_2(B)C_2$$

زمان‌بندی سریالی بالا در حقیقت به دو فرایند تقسیم می‌شود. چرا که در انتهای تراکنش اول پیام سقوط کرده و برنامه به دنبال فرایند بعدی رفته است که روی منبع دیگری در حال انجام پردازش است.

فرایند نافرجام اول:

$$S_1 = R_1(A)W_1(A)a_1$$

فرایند commit شده دوم:

$$S_1 = W_2(A)W_2(B)C_2$$

جدول ۲: تراکنش‌های سریالی پی در پی

T_1	$R_1(A)$	$W_1(A)$	a_1			
T_2				$W_2(A)$	$W_2(B)$	C_2

۴.۳.۳ زمان‌بندی‌های معادل در برخورد یا Conflict equivalent

زمانی که دستورات یک زمان‌بندی را وارد زمانبندی دیگر کنیم به گونه‌ای که باعث تصادم و برخورد نشود، این دستورات در این زمان‌بندی با هم معادل در برخورد هستند.

با توجه به تراکنش‌های t_1 و t_2 و t_3 و t_4 زیر، می‌توان دریافت که این دو تراکنش با یکدیگر معادل در برخورد هستند. به گونه‌ای که بعد از جا به جایی هیچ تصادمی رخ نداده است.

جدول ۳: تراکنش‌های معادل در برخورد اول

T_1	$R(Q)$	$W(Q)$		$R(P)$		$W(P)$	C			
T_2			$R(Q)$		$W(Q)$			$R(Q)$	$W(Q)$	C

جدول ۴: تراکنش‌های معادل در برخورد دوم

T_3	$R(Q)$	$W(Q)$		$R(P)$	$W(P)$		C			
T_4			$R(Q)$			$W(Q)$		$R(Q)$	$W(Q)$	C

اما در مثال بعد هر دو تراکنش t_1 و t_2 مستعد به برخورد در یکی از فرایندها در زمان هستند.

جدول ۵: تراکنش‌های معادل در برخورد اول

T_1	$R(Q)$	$W(Q)$		$R(P)$		$W(P)$	C			
T_2			$R(Q)$		$W(Q)$			$R(Q)$	$W(Q)$	C

جدول ۶: تراکنش‌های معادل در برخورد اول

T_1	$R(Q)$	$W(Q)$		$R(P)$		$W(P)$	C			
T_2		$R(Q)$	$R(Q)$		$W(Q)$			$R(Q)$	$W(Q)$	C

۵.۳.۳ گراف پی در پی پذیر

کامپیوتر برای تشخیص وجود برخورد در تراکنش‌ها از تئوری گراف پی در پی پذیر استفاده می‌کند. در این روش به صورت بصری ارتباطات تراکنش‌ها را نسبت به یکدیگر را نمایش می‌دهیم. در صورتی که بین دو یا چند تراکنش دور یا حلقه ایجاد شود، می‌گوییم که این تراکنش‌ها با هم برخورد دارند.

سیستم DBM از گراف زمان اجرا خبر دارد و دائما در حال بروزرسانی آن است. اگر وجود دور یا حلقه را تشخیص دهد، برخورد را بررسی کرده و اعلام می‌کند که این تراکنش‌ها پی در پی پذیر در برخورد نیستند و از اجرای این تراکنش‌ها جلوگیری می‌کند.

۶.۳.۳ کشتن فرایند تراکنش‌ها

منظور از جلوگیری می‌تواند به دو روش باشد: یا کلا از اجرای تراکنش‌ها جلوگیری می‌کند یا بررسی می‌کند که کدام تراکنش یا تراکنش‌ها باعث ایجاد برخورد در تراکنش‌های دیگر می‌شود، آن را تشخیص داده و تراکنش آن را می‌کشد [۱].

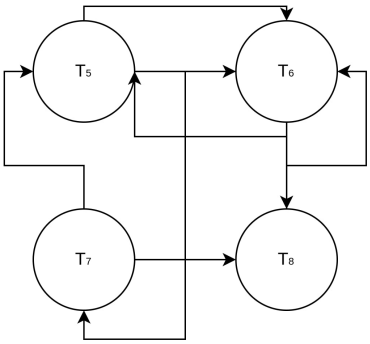
[۱] Kill transaction

برای مثال تراکنش‌های زیر را در نظر بگیرید:

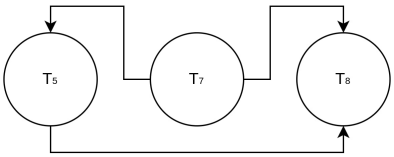
جدول ۷: تراکنش‌های بانکی

T_5			W(Q)		
T_6	R(Q)				W(Q)
T_7		W(Q)			
T_8				R(Q)	

گراف این تراکنش‌ها به شکل زیر است. توجه شود که هر تراکنش می‌تواند به صورت ترتیبی نسبت به تراکنشی بعدی خود ارتباط داشته باشد. در صورتی که حلقه ایجاد شود بایستی عامل ایجاد حلقه پیدا و سپس کشته شود.



شکل ۲: گراف تراکنش‌ها و ایجاد ارتباطات حلقه دار



شکل ۳: تراکنش حذف شده و ایجاد گرافی بدون حلقه

در این مثال برای حذف حلقه می‌تواند یکی یکی تراکنش‌های مورد نظر را بررسی کرد و در صورت حذف یکی از تراکنش‌ها حلقه حذف شد می‌توان آن را نتیجه گرفت و اعلام کرد این تراکنش‌ها باهم سازگارند و برخورد ایجاد نمی‌کنند. در نهایت سیستم DBM تصمیم به اجرای تراکنش‌ها خواهد کرد.

۷.۳.۳ پی در پی پذیری در دید یا View equivalent

زمانی می‌گوییم پی در پی پذیری در دید برقرار است که نتایج یکسانی در سیستم DBM با یک زمان‌بندی پی در پی داشته باشیم.
سه قاعده اصلی پی در پی پذیری در دید:

۱. برای هر داده Q تراکنشی که در S مقدار اولیه داده‌ای Q را می‌خواند در S' هم همان تراکنش اولیه مقدار Q را بخواند (خواندن‌های اولیه)

۲. برای هر داده Q اگر t_i در S داده Q را از t_j می‌خواند، در S' هم t_i همان داده را از t_j بخواند. (خواندن‌های میانی)

۳. برای هر داده Q آخرین تراکنشی از S که روی Q می‌نویسد در S' هم همان تراکنش نوشتن پایانی را روی Q انجام دهد. (نوشتن‌های پایانی)

نکته: یک زمانبندی پی در پی پذیر در دید است، هنگامی که معادل در دید با یک زمانبندی پی در پی پذیر باشد که نتایج درستی را منعکس کند.

۸.۳.۳ مثال اول پی در پی پذیری در دید

پی در پی پذیر در دید است چرا که فرایند خواندن اولیه و عملیات میانی و در نهایت نوشتن پایانی را دارا می‌باشد.

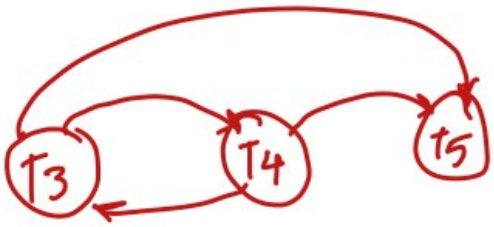
$T_6 < ..... < T_7$

$T_6 < T_5 < T_8 < T_7$

اما پی در پی پذیر در برخورد نیست چرا که بین تراکنش T_7 و T_8 یک حلقه ایجاد می‌شود و می‌تواند عاملی در برخورد باشد.

جدول ۸: پی در پی پذیری در دید

T_5			W(Q)		
T_6	R(Q)				
T_7		W(Q)			W(Q)
T_8				R(Q)	



۹.۳.۳ مثال دوم پی در پی پذیری در دید

جدول ۹: پی در پی پذیری در دید

T_3	R(Q)		W(Q) C
T_4		W(Q) C	
T_5			W(Q) C

جواب: این مثال پی در پی پذیر در دید است:

$T_3 < ..... < T_5$

چرا که در T_3 خواندن‌های اولیه صورت گرفته، در T_4 و زمان میانی T_3 عملیات میانی نوشتن رخ داده است. در انتها در تراکنش T_5 مطابق با قانون پی در پی پذیری در دید نوشتن پایانی انجام شده است.

اما پی در پی پذیر در برخورد نیست چرا که در میان تراکنش‌ها حلقه رخ داده است.

۱۰.۳.۳ نمادگزاری

کامپیوتر چگونه پی در پی پذیری در دید را متوجه می‌شود؟ با استفاده از نمادگزاری (خواندن از). برای یک زمانبندی، مجموعه‌ای از (خواندن از)ها را تشکیل می‌دهیم. این مجموعه باید با مجموعه خواندن ازها در یک زمانبندی پی در پی دیگر یکسان باشد تا در دید هم پی در پی پذیر باشد. در این روش مدت زمان اجرا [۲] برای کامپیوتر طولانی است و اجرای آن برای کامپیوتر بهینه نیست.

مثال:

$S = r_2(x), w_2(x), r_1(x), r_1(y), r_2(y), w_2(y), c_1, c_2$

بدست آوردن مرجع اصلی

$RF(S) = (T_0, x, T_2), (T_2, x, T_1), (T_0, y, T_1), (T_0, y, T_2)$

بدست آوردن $T_1 < T_2$

در این مرحله ابتدا تراکنش‌های زمانبندی اول انجام می‌شود و سپس تراکنش‌های زمانبندی دوم:

$T_1 < T_2$ = $r_1(x), r_1(y), c_1, r_2(x), w_2(x), r_2(y), w_2(y), c_2$

بدست آوردن RF به وسیله ترتیب زمانبندی بالا:

$RF(T_1 < T_2) = (T_0, x, T_1), (T_0, y, T_1), (T_0, x, T_2), (T_0, y, T_2)$

بدست آوردن $T_2 < T_1$

در این مرحله زمانبندی دوم در ابتدا و سپس زمانبندی اول بعد از آن اجرا می‌شود:

$T_2 < T_1$ = $r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(x), r_1(y), c_1$

بدست آوردن RF به وسیله ترتیب زمانبندی جدید بالا:

$RF(T_2 < T_1) = (T_0, x, T_2), (T_0, y, T_2), (T_0, x, T_1), (T_0, y, T_1)$

بعد از نوشتن عملیات بالا متوجه خواهید شد که هیچ کدام از $RF(T_1 < T_2)$ و $RF(T_2 < T_1)$ با مرجع اصلی $RF(S)$ که در ابتدا نوشتیم برابر نیست.

یک زمانبندی ۲ شرط دارد که درست باشد:

- پی در پی پذیر باشد (قانون جامعیت در برخورد و دید برقرار باشد)
- ترمیم پذیر باشد

نکته: اگر یک زمانبندی پی در پی پذیر در برخورد باشد در دید هم پی در پی پذیر خواهد بود.

۴ ترمیم پذیری

۱.۴ مفهوم Rollback شدن

اگر یک زمانبندی در میان اجرا Abort شود چون تراکنش‌های دیگر به آن وابسته هستند، این تراکنش برای درست انجام شدن بایستی از اول انجام شود یا اصطلاحا Rollback صورت گیرد.

[۲] Runtime

۲.۴ زمانبندی ترمیم پذیر یا Recoverable scheduling

زمانبندی را ترمیم پذیر می‌گوییم اگر T_j از T_i روی منبع اطلاعاتی خواندنی را انجام می‌دهد که حتما به طور صحیح و کامل انجام شود. منظور از صحیح بودن آن است که حتما تراکنش‌ها در زمانبندی Commit شده باشند. اما توجه شود که تراکنش قبلی بایستی زودتر از تراکنش بعد خود Commit شده باشد.

مثال ۱: آیا زمانبندی زیر ترمیم پذیر است؟

جدول ۱۰: مثال ۱: بررسی ترمیم پذیری

T_1	R(A)	W(A)		R(B)	A
T_2			R(A)		C

این زمانبندی ترمیم پذیر نیست چرا که درست نیست. زیرا در زمانبندی T_1 بعد از انجام تراکنش عمل سقوط یا Abort اتفاق افتاده است و T_2 در حال خواندن مقدار از منبعی از زمانبندی بالاتر خود است که تراکنش‌اش به دلیل $\mathrm{Dirty\ Read}$ RollBack خواهد شد و به صورت صحیح کامل نشده است.

مثال ۲: ترمیم پذیری زمانبندی زیر را بررسی کنید

جدول ۱۱: مثال ۲: بررسی ترمیم پذیری

T_1	R(A)	W(A)	W(B)	C		
T_2			R(A)	W(A)	R(B)	C

این زمانبندی RC می‌باشد چرا که تراکنش‌ها به صورت صحیح انجام شداند (عمل Commit شدن در تراکنش‌ها وجود دارد). نکته مهم در این زمانبندی آن است که به دلیل وابسته بودن عملیات تراکنش‌ها به یکدیگر ممکن است دائما در حال بررسی وجود Commit در تراکنش‌ها باشیم تا زمانی عمل Abort رخ ندهد (اشاره به تراکنش دوم زمانی خواندن روی منبع A صورت گرفته است). به همین دلیل زمانبندی ACA در اینجا تعریف خواهد شد. زمانی تراکنش بالا می‌تواند ACA باشد که اولین خواندن دقیقا بعد از کامیت تراکنش اول صورت گیرد.

۳.۴ سقوط‌های آبشاری یا Cascading Aborts

در جدول ۱۲، دقیقا مانند مثال ۲، تمام تراکنش‌ها به همان شکل است. اما به جای کامیت شدن در این جا تراکنش اول در نهایت سقوط می‌کند، هبا شکل دیگر ترمیم پذیر نخواهد بود و با سقوط‌های آبشاری رو به رو است (اشاره به عملیات R(A) و R(B) که نوبتی سقط می‌شوند).

جدول ۱۲: بررسی سقوط‌های آبشاری در مثال ۲

T_1	R(A)	W(A)	W(B)	A		
T_2			R(A)	W(A)	R(B)	C

۴.۴ Avoiding Cascading Aborts

در حقیقت فرایند زمانی فاقد سقوط آبشاری است؛ اگر T_j از T_i بخواند آنگاه T_i قبل از خواندن T_j کامیت شده باشد. بطور کل به آن ACA می‌گویند که جز تراکنش‌های ترمیم پذیر می‌باشد. به بیانی دیگر اگر قبل از اولین Read در تراکنش دوم، در تراکنش اول کامیت صورت گرفته باشد آن زمانبندی ACA می‌باشد.

جدول ۱۳: نمونه‌ای از فرایند ACA

T_1	R(A)	R(B)	W(A)	C					
T_2					R(A)	W(A)	C		
T_3								R(A)	C

نکات

- در پی در پی پذیری تنها در مورد مشکلات همروندی صحبت می‌شد
- در زمانبندی‌های ACA هدف آن است که اول کامیت انجام شود و سپس خواندن منبع صورت گیرد در غیر این صورت زمان برای خواندن مقداری که تثبیط نشده است صرف می‌شود و زمان اصلی برای انجام فرایندهای دیگر را از دست خواهیم داد.
- یکی از قوانین ترمیم پذیری عدم وجود سقوط‌های آبشاری است، پس اگر یک زمانبندی ACA باشد پس ترمیم پذیر می‌باشد.

سوال، زمانبندی زیر را از نظر ACA و RC بررسی کنید

جدول ۱۴: بررسی زمانبندی مثال ۴

T_1	R(A)	W(A)	W(B)	C			
T_2		W(B)	W(C)	W(D)	R(A)	R(B)	C

این زمانبندی ACA می‌باشد چرا که اولین Read در تراکنش T_j دقیقا بعد از کامیت تراکنش T_i صورت گرفته است.

۵.۴ زمانبندی‌های محض (سختگیرانه) یا Strict

در دو تراکنش T_i و T_j، اگر T_j داده‌ای را پس از نوشتن T_i بخواند یا بنویسد بایستی قبل از آن Commit صورت گرفته باشد.

مثال ۵: زمانبندی زیر را از نظر محض بودن، ترمیم پذیری و ACA بررسی کنید

جدول ۱۵: مثال ۵: بررسی تمام لایه‌های ترمیم پذیری

T_1	R(A)	R(B)	W(A)		C	
T_2				W(A)	W(B)	C

- زمانبندی بالا محض نیست، چرا که بعد از نوشتن در تراکنش T_i بایستی کامیت گذاشته شود و سپس تراکنش T_j می‌تواند خواندن و نوشتن خود را انجام دهد. در این مثال تراکنش دوم خواندن یا نوشتن خود را بعد از کامیت نوشتن تراکنش اول انجام نداده است.
- در این مثال به دلیل آنکه خواندنی بعد از کامیت صورت نگرفته (اشاره به قانون ACA می‌باشد) و تراکنش‌ها هر دو کامیت شده‌اند و یک زمانبندی صحیح می‌باشد، پس ترمیم پذیر می‌باشد.

نکته: سیستم DBM از یکسری پروتکل‌هایی برای پی در پی پذیری و ترمیم پذیری استفاده می‌کند تا دیتابیس به شکل صحیح کار کند. (پیروی از دو شرط اصلی)

۵ پروتکل‌های کنترل همروندی

بعد از دیدن دستور، ۳ کار انجام می‌شود:

۱. اجرای دستور

۲. به تاخیر انداختن دستور (ممکن است به دلایلی وارد صف شود برای بدست آوردن قفل)

۳. نپذیرفتن دستور یا سقوط آن

۱.۵ پروتکل‌های مبتنی بر قفل

در این نوع پروتکل واحدی به نام Lock Manager تراکنش‌ها را بررسی می‌کند، اگر ناسازگاری ww wr یا rw وجود نداشته باشد اجازه خواندن را به تراکنش می‌دهد و سپس بعد از آن که تراکنش کارش تمام شد می‌تواند قفل را تحویل دهد تا تراکنش بعدی بتواند عملیات قفل گذاری را انجام دهد.

قفل‌ها دو نوع هستند

۱. قفل‌های دو حالته (دودویی): هیچ تفاوتی ندارد که تراکنش می‌خواهد بخواند یا بنویسد، به هر صورت قفل را اختصاص می‌دهد و در این فرایند هم تنها یک قفل برای هر دو عمل خواندن و نوشتن وجود دارد

۲. قفل‌های اشتراکی_انحصاری یا Shared Exclusive Lock: از یک قفل برای خواندن (S) استفاده می‌کند و از قفل دیگر برای نوشتن (X)

نکات

- مزیت قفل‌های اشتراکی_انحصاری در انجام تراکنش‌ها به صورت موازی است
- اگه قفل به حالت ناسازگار برسد آن تراکنش را به تاخیر می‌اندازد
- قفل گذاری روی داده‌های زیاد با Seed بالا همروندی را کاهش می‌دهد
- وقتی Seed کم باشد Overhead زمانی خواهیم داشت و پردازش گران است
- منظور از Seed در حقیقت منبعی است که می‌خواهیم روی آن قفل گذاری کنیم
- منابع مورد قفل گذاری می‌تواند یک ویژگی از جدول، یک جدول با رکوردهای متفاوت و یا حتی یک OS Page Table باشد
- قفل گذاری درست باعث می‌شود تا زمانبندی درست داشته باشیم
- زمانی که بر روی یک Table قفل می‌گذاریم، روی داده‌های بیشتری قفل گذاشته می‌شود و داده‌های بیشتری از دسترس خارج می‌شود که در نهایت همراه با همروندی کمتر است
- در قفل گذاری اشتراکی_انحصاری چندین تراکنش می‌توانند به طور همزمان قفل S را بدست آورند. زیرا حالت Read-Read پدید می‌آید و حالت سازگاری است و مشکلی ایجاد نمی‌کند.
- حالت ناسازگار زمانی است که یک تراکنش بخواهد قفل S را بدست آورد و دیگری می‌خواهد قفل X را بدست آورد.

نوشتار

- $S_i(Q)$: دریافت قفل اشتراکی برای عملیات خواندن
- $X_i(Q)$: دریافت قفل انحصاری برای عملیات نوشتن
- $U_i(Q)$: آزادسازی قفل روی منبع Q

مثال ۱

$$S_1 = R_1(A)W_1(A)A_1W_2(A)W_2(B)C \quad (۱)$$

پاسخ مثال ۱

S_1 = $S_1(A)\ R_1(A)\ X_1(A)\ W_1(A)\ U_1(A)\ A_1\ X_2(A)\ W_2(A)\ X_2(B)\ W_2(B)\ U_2(A)\ U_2(B)\ C$

مثال کلید اشتراکی-انحصاری

T_3			W(Q)		
T_4	R(Q)			W(Q)	
T_5		W(Q)			
T_6					R(Q)

تبدیل جدول به سریال

$R_4(Q)\ W_5(Q)\ W_3(Q)\ W_4(Q)\ R_5(Q)$

حل

$\rightarrow S_4(Q)\ R_4(Q)\ X_5(Q)\ X_3(Q)\ X_4(Q)\ W_4(Q)\ U_4(Q)\ W_5(Q)\ U_5(Q)\ W_3(Q)\ U_3(Q)\ S_6(Q)\ R_6(Q)\ U_6(Q)$

در این مسئله به دلیل وجود دو درخواست [۳] در تراکنش T_4 ابتدا قفل به خواندن منبع Q اختصاص داده می‌شود ولی بعد از آن قفل آزاد نمی‌شود، تا زمانی که این تراکنش به طور کامل کارش را انجام دهد و تمام شود. بعد از آن یکی یکی تراکنش‌ها می‌توانند به درخواست‌هایشان برسند و عمل خواندن را از صف خارج کرده و بعد از انجام موفقیت آمیز عملیات خواندن قفل را آزاد کنند.

[۳] Request

۲.۵ بن بست و قحطی

سوال: چه زمانی بن‌بست یا DeadLock رخ می‌دهد؟ زمانی که یک پردازه (تراکنش) منتظر بدست آوردن قفل باشد. مهم‌ترین راهکار برای کم کردن بن‌بست حذف یا Abort تراکنش باعث بن‌بست است.

جدول ۱۶: نمونه‌ای از تراکنش‌هایی که به بن‌بست بر خورده‌اند

T_3	x(B)	w(B)			x(A)
T_4		s(A)	r(A)	s(B)	

جدول بالا به دلیل ناسازگاری WR و RW به بن‌بست بر می‌خورد. چرا که در تراکنش T_3 برای نوشتن روی منبع B قفل نوشتن گذاشته شده است ولی Unlock نشده است و تراکنش T_4 نمی‌تواند قفل خواندن را روی منبع A بگذارد چرا که تراکنش T_3 هنوز قفل را آزاد نکرده است. در این حالت یک انتظار چرخشی یا Unlimited wating بین تراکنش‌ها رخ داده است که دائما منتظر آزاد سازی قفل یکدگیر هستند تا بتوانند بقیه عملیات را انجام دهند. در T_4 قفل خواندن روی منبع A گذاشته می‌شود و بعد از آن در خواست قفل گذاری را روی منبع B را دارد در حالی T_3 دقیقا روی منبع B عمل نوشتن را انجام می‌دهد و قفل را رها نکرده است و در مقابل در ادامه همین تراکنش درخواست نوشتن روی منبع A را دارد که در تراکنش T_4 درخواست آن داده شده ولی هیچ قفلی آزاد نشده است. دلیل اصلی بن‌بست همین است.

بایستی در نظر داشت که با ساقط کردن یک تراکنش نمی‌توان به تنهایی مشکل بن‌بست را حل کرد بلکه باعث ایجاد مشکل جدیدی به نام قحطی خواهد شد. برای مثال یک تراکنش که قصد زدن قفل x روی داده‌ای است منتظر دنباله‌ای از تراکنش‌ها بماند که همگی می‌خواهند قفل s را روی همان منبع (داده) بزنند و این انتظار به پایان نرسد می‌گویم در این حالت تراکنش تعریف قفل x روی منبع دچار قحطی شده است.

جدول ۱۷: قحطی

T_1	S(Q)			U(Q)
T_2		X(Q)		
T_3			S(Q)	
T_4				S(Q)
...				

در تراکنش‌های بالا به دلیل انتظار نامحدود ممکن است قحطی بین تراکنش‌های دیگر پیش آید، به دلیل آنکه همه می‌خواهند روی یک منبع عملیاتی را انجام دهند که در تراکنش اول قفل خواندن در دست است و تراکنش‌های دیگر باید منتظر آزاد سازی آن باشند.

۳.۵ پروتکل‌های قفل دو مرحله‌ای 2PL

برای توضیح این پروتکل‌ها تراکنش‌های زیر را در نظر بگیرید:

جدول ۱۸: زمانبندی S_5

T_1	x(A)	Dec(A، amount)	w(A)	u(A)								x(B)	Inc(B، amount)	w(B)	u(B)
T_2					s(A)	r(A)	s(B)	r(B)	Dis(A+B)	u(A)	u(B)				

در جدول ۱۶، شما تراکنش‌هایی را می‌بینید که در حال کم کردن از یک منبع و اضافه کردن آن مقدار به منبع دیگری هستند. ولی این تراکنش‌ها صحیح نیستند و دیتابیس نمی‌تواند به درستی کار کند چرا که با بازیابی ناسازگار رو به رو است. با توجه به تراکنش T_2 می‌توان دریافت که بعد از قفل گذاری روی منبع A برای خواندن، سعی در قفل گذاری رو منبع B دارد که اصلا معتبر نیست. زیرا در تراکنش T_1 هیچ عملیات یا حتی قفل گذاری روی منبع B انجام نشده است که الان سعی در خواندن آن دارد. پس با بازیابی ناهمگام یا Inconsistent retrieval رو به رو خواهد بود و باید از یک پروتکل قفل گذاری مناسب جهت این کار استفاده کند.

نکته

اگر زمابندی پی در پی پذیر در برخورد باشد آنگاه تمام مشکلات مربوط به همروندی تراکنش‌ها برطرف خواهد شد.

۴.۵ مراحلی که در فرایند پروتکل 2PL برای قفل گذاری صورت می‌گیرد

مرحله اول - مرحله رشد یا Growing

در این مرحله تراکنش می‌تواند قفل گذاری کند (احتمال انجام کار را دارد)، اما نمی‌تواند قفل را آزاد کند.

مرحله دوم - مرحله عقب نشینی یا Shrinking

در این مرحله تراکنش می‌تواند قفل را آزاد کند (احتمال انجام کار دارد)، اما نمی‌تواند روی منبعی قفل گذاری جدیدی را انجام دهد.

۵.۵ پروتکل B۲PL یا Basic Two Phase Locking

در این مرحله، تراکنش‌ها شروع به قفل گذاری منابع برای انجام عملیات خود می‌کنند به محض اینکه یکی از تراکنش‌ها قفلی را آزاد کند وارد مرحله دوم یا Shrinking خواهد شد و از این بعد نمی‌تواند هیچ قفل گذاری را انجام دهد.

جدول ۱۹: زمانبندی S_6

T_1	X(A)	Dec(A، amount)	W(A)	X(B)	Inc(B، amount)	U(a)					W(B)	U(B)						
T_2							S(A)	R(A)					S(B)	R(B)	Dis(A+B)	U(A)	U(B)	

این پروتکل قفل گذاری ترمیم پذیر نخواهد بود چرا که مشکل بن‌بست و سقوط‌های آبشاری را دارد. برای رفع این مشکلات پروتکل دیگری به نام C۲PL یا قفل گذاری محافظه کارانه را معرفی کردند.

۶.۵ قفل گذاری C۲PL یا Conservative Two Phase Locking

در این پروتکل قبل از اجرای هر دستور و عملیاتی، تراکنش‌ها بایستی قفل‌های مورد نیاز را از قبل گرفته باشند اگر موفق نشد دوباره در صف قرار می‌گیرد (تا اینکه قفل‌های قبلی باز شوند و بتواند قفل جدیدی را تعریف کند).

جدول ۲۰: زمانبندی S_7

T_1	X(A)	X(B)	Dec(A، amount)	W(A)	Inc(B، amount)	W(B)	U(A)				U(B)	C						
T_2								S(A)	R(A)				S(B)	R(B)	U(A)	Disp(A+B)	U(B)	C

مهم‌ترین مشکلات این روش پایین آمدن سطح سرعت همروندی و نیاز به دانستن مجموعه قفل‌های مورد نیاز هر تراکنش قبل از شروع اجرای دستورات می‌باشد. امکان بن‌بست در این روش از بین می‌رود اما باز هم ترمیم پذیر نخواهد بود فلذا می‌تواند باعث رخ دادن سقوط آبشاری شود. استفاده از این پروتکل گران است چرا که برای تضمین عدم وقوع بن‌بست، سرعت و کارایی همروندی را تا حد چشمگیری کاهش می‌دهد در حالی که در دنیای واقعی احتمال بروز بن‌بست آنقدر زیاد نمی‌باشد.

۷.۵ پروتکل S۲PL یا Strict Two Phase Locking

در این پروتکل علاوه‌بر بن‌بست، امکان سقوط آبشاری نیز وجود دارد اما به طور کلی در این پروتکل بعد از قفل گذاری‌ها، ابتدا تراکنش بایستی کامیت یا Abort شود و سپس قفل‌هایی که در اختیار دارد را آزاد کند. قفل‌های خواندن می‌تواند کمی زودتر بعد از آخرین دستور تراکنش یا قبل از کامیت یا Abort باز شوند وگرنه در بقیه عملیات شبیه B۲PL عمل می‌کند. اگرچه این پروتکل کمی سختگیرانه عمل می‌کند و شاید بسیاری از زمانبندی‌ها که در واقع درست هستند را به دلیل احتمال بروز مشکل نپذیرد، اما به عنوان یکی از بهترین گزینه‌ها در اکثر سیستم‌های دیتابیسی مورد استفاده قرار گرفته است. مزیت اصلی این پروتکل که آنرا به پرکاربردترین و بهترین گزینه تبدیل کرده است، تمضین پی در پی پذیری و ترمیم پذیری است.

از مزیت دیگر این پروتکل می‌توان به کم کردن پیام‌ها در بانک‌های اطلاعاتی نامتمرکز اشاره کرد زیرا نیازی به پیام‌های باز کردن قفل ندارد.

جدول ۲۱: زمانبندی S_8

T_1	X(A)	Dec(A، amount)	W(A)	X(B)	Inc(B، amount)	W(B)	**C**	U(A)		U(B)								
T_2									S(A)		R(A)	S(B)	R(B)	Disp(A+B)	**C**	U(A)	U(B)	

نکات و بررسی SS۲PL یا R۲PL

۱. این پروتکل همانطور که از نامش پیداست (Strong Strict) سختگیرانه‌تر از S۲PL می‌باشد و معمولا استفاده نمی‌شود.

۲. در این پروتکل قفل‌های خواندن و نوشتن یا S و X باید بعد از Abort یا Commit آزاد شوند.

۸.۵ پروتکل SC۲PL

این پروتکل ترکیبی از دو پروتکل S۲PL و C۲PL برای بهروری و کارایی بیشتر است. در این پروتکل بن‌بست و گرسنگی و سقوط آبشاری وجود ندارد! عملکرد این پروتکل با خواندن دو پروتکل ترکیبی آن حاصل می‌شود. اما کمترین میزان همروندی را خواهد داشت.

جدول ۲۲: زمانبندی S_9

T_1	X(A)	X(B)	Dec(A، amount)	W(A)	Inc(B، amount)	W(B)	**C**	U(A)		U(B)										
T_2									S(A)		S(B)	R(A)	R(B)	Disp(A+B)	**C**	U(A)	U(B)			

مثال ۱

معادل زمانبندی زیر را یکبار با قفل باینری و یکبار با قفل S/X و رعایت پروتکل B۲PL بنویسید:

مثال ۲

همین تمرین را با پروتکل‌های C۲PL S۲PL و SC۲PL آنجام دهید.