

پایگاه داده پیشرفته  
دکتر شجاعی مهر  
علیرضا سلطانی نشان  
۳ دی ۱۴۰۲

## فهرست مطالب

۳	۱ تراکنش
۳	۲ قوانین ACID
۳	۱.۲ اتمیک یا Atomicity
۳	۲.۲ جامعیت یا Consistency
۳	۳.۲ انزوا یا Isolation
۳	۴.۲ قابلیت اعتماد یا Durability
۴	۵.۲ تنظیم قابلیت انزوا
۴	۱.۵.۲ وضعیت تراکنش
۶	۳ همروندی
۶	۱.۳ مزیت همروندی
۶	۲.۳ معایب همروندی
۶	۳.۳ زمان‌بندی
۶	۱.۳.۳ نظریه پی در پی پذیری زمان‌بندی‌ها
۶	۲.۳.۳ سه شرط اصلی تصادم
۷	۳.۳.۳ زمان‌بندی سریالی
۷	۴.۳.۳ زمان‌بندی‌های معادل در برخورد یا Conflict equivalent
۸	۵.۳.۳ گراف پی در پی پذیر
۸	۶.۳.۳ کشتن فرایند تراکنش‌ها
۱۰	۷.۳.۳ پی در پی پذیری در دید یا View equivalent
۱۰	۸.۳.۳ مثال اول پی در پی پذیری در دید
۱۱	۹.۳.۳ مثال دوم پی در پی پذیری در دید
۱۲	۱۰.۳.۳ نمادگذاری
۱۲	۴ ترمیم پذیری
۱۲	۱.۴ مفهوم Rollback شدن
۱۳	۲.۴ زمان‌بندی ترمیم پذیر یا Recoverable scheduling
۱۳	۳.۴ سقوط‌های آبشاری یا Cascading Aborts
۱۴	۴.۴ Avoiding Cascading Aborts
۱۴	۵.۴ زمان‌بندی‌های محض (سختگیرانه) یا Strict
۱۵	۵ پروتکل‌های کنترل همروندی
۱۵	۱.۵ پروتکل‌های مبتنی بر قفل
۱۷	۲.۵ بن بست و قحطی

۳.۵	پروتکل‌های قفل دو مرحله‌ای ۲PL	۱۸
۴.۵	مراحل‌ی که در فرایند پروتکل ۲PL برای قفل گذاری صورت می‌گیرد	۱۸
۵.۵	پروتکل B۲PL یا Basic Two Phase Locking	۱۹
۶.۵	قفل گذاری C۲PL یا Conservative Two Phase Locking	۱۹
۷.۵	پروتکل S۲PL یا Strict Two Phase Locking	۱۹
۸.۵	پروتکل SC۲PL	۲۰
<b>۶</b>	<b>پروتکل‌های مبتنی بر مهر زمانی Timestamp</b>	<b>۲۱</b>
۱.۶	قواعد	۲۱
۱.۱.۶	قاعده خواندن	۲۱
۲.۱.۶	قاعده نوشتن	۲۲
<b>۷</b>	<b>روش‌های مدیریت بن‌بست</b>	<b>۲۳</b>
۱.۷	چشم پوشی یا Ignore	۲۳
۲.۷	فرصت یا Timeout	۲۳
۳.۷	پیشگیری یا Prevention	۲۳
۴.۷	اجتناب یا Avoidance	۲۳
۵.۷	تشخیص و رفع بن‌بست یا Detection and resolve	۲۴
<b>۸</b>	<b>واحد مدیریت ترمیم و الگوریتم‌ها</b>	<b>۲۵</b>
۱.۸	انواع خطاهای مربوط به سیستم دیتابیس	۲۵
۱.۱.۸	خطاهای تراکشنی	۲۵
۲.۱.۸	خطاهای سیستمی	۲۵
۳.۱.۸	خطای رسانه‌ای	۲۵
۲.۸	روال دسترسی به داده‌ها برای انجام تراکشن	۲۶
۳.۸	الگوریتم‌های ترمیم	۲۶
۱.۳.۸	مرحله اول	۲۶
۲.۳.۸	مرحله دوم	۲۶
۴.۸	عملیات Redo	۲۷
۵.۸	عملیات Undo	۲۷
۶.۸	رویکردهای الگوریتم ترمیم	۲۷

## 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 کاملاً انجام شود یا کلاً 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. مهم‌ترین 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 جانی.

### ۳.۲ انزوا یا Isolation

هر سیستم جامع پایگاه داده‌ای باید بتواند روی هم‌رود 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 Durability

قابلیت اعتماد یکی از مهم‌ترین ویژگی‌های هر سیستم دیتابیزی است. یعنی بتوان داده‌ها را در پایگاه داده 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 بک آپ 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. Repeatable read

۴. Serializable

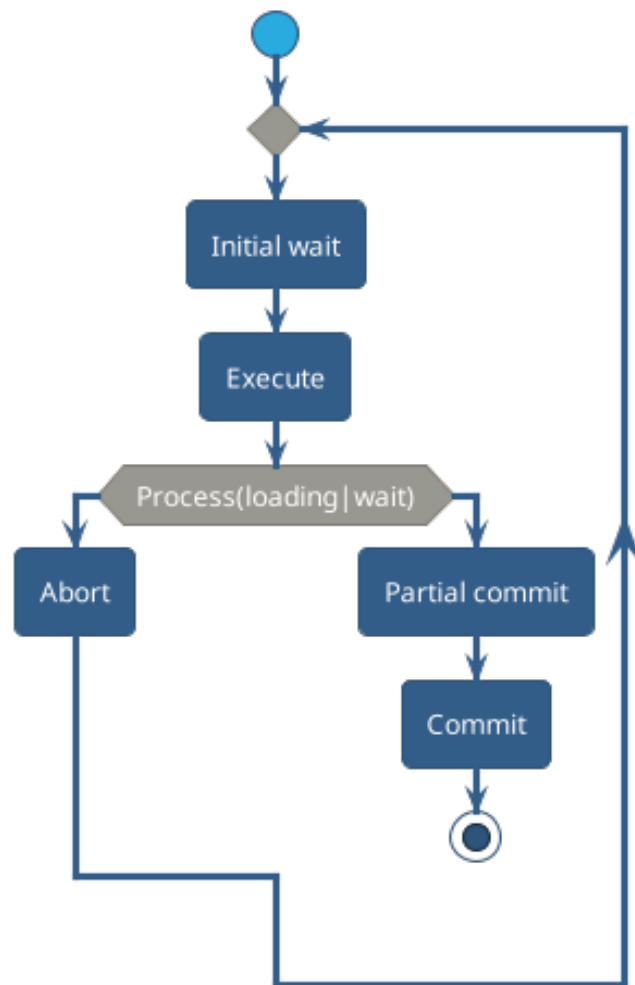
یادآوری: هر تراکنش دو حالت در پایان پیدا می کند:

- Commit: تراکنش در نهایت تایید و انجام می شود
- Abort: تراکنش در نهایت سقط یا صرفه نظر می شود

### ۱.۵.۲ وضعیت تراکنش

نکته: Abort در دو شرط اتفاق می افتد:

۱. زمانی که اجرای تراکنش به خطای Run time دچار شود.
۲. خرابی و نقص سیستم که روی اجرای تراکنش تاثیر می گذارد که کامل نشود



شکل ۱: نمودار شروع فرایند تراکنش‌ها

## ۳ همروندی

### ۱.۳ مزیت همروندی

۱. افزایش سرعت گزردهی 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. معمولاً 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 تضادم

اگر  $p_i$  و  $q_j$  دو تراکنش باشند:

۱.  $i \neq j$

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 دائماً 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 کلاً 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.<sup>۱</sup>

<sup>۱</sup> Kill transaction

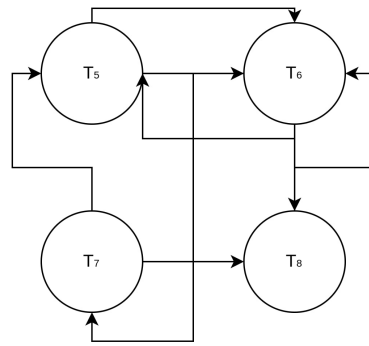


برای مثال تراکنش‌های زیر را در نظر بگیرید:

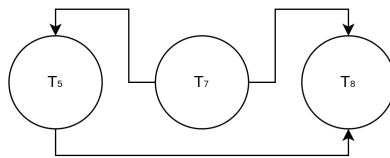
جدول ۷: تراکنش‌های بانکی

$T_5$			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.

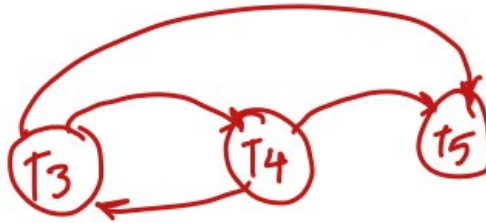
$$T_6 < \dots < T_7$$

$$T_6 < T_5 < T_8 < T_7$$

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:

$$T_3 < \dots < T_5$$

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 تراکشن  $T_5$  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 تراکشن‌ها حلقه رخ داده است.

### ۱۰.۳.۳ نمادگذاری

کامپیوتر چگونه پی در پی پذیری در دید را متوجه می‌شود؟ با استفاده از نمادگذاری (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<sup>۲</sup> 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 اصطلاحاً Rollback صورت گیرد.

<sup>۲</sup> 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 حتماً به طور صحیح و کامل انجام شود. منظور از صحیح بودن آن است که حتماً 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 RollBack Dirty Read 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 شدند (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 دائماً در حال بررسی وجود Commit 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 دقیقاً بعد از کامیت تراکنش اول صورت گیرد.

## ۳.۴ سقوط‌های آبشاری یا Cascading Aborts

در جدول ۱۲، دقیقاً 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 تثبیت 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  $T_j$  دقیقاً 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 خواندن 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  $rw$  یا  $wr$  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 حالت (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 (S): 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
- اگر 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 بخواند 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<sup>۳</sup> 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.



## ۲.۵ بن بست و قحطی

سوال: چه زمانی بن بست یا DeadLock رخ می دهد؟ زمانی که یک پردازش (تراکنش) منتظر بدست آوردن قفل باشد. مهم ترین راهکار برای کم کردن بن بست حذف یا Abort تراکنش باعث بن بست است.

جدول ۱۶: شکل کلی بن بست

$T_1$	$T_2$
Lock(A)	Lock(B)
Lock(B)	Lock(A)

جدول ۱۷: نمونه ای از تراکنش هایی که به بن بست بر خورده اند

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

جدول بالا به دلیل ناسازگاری WR و RW به بن بست بر می خورد. چرا که در تراکنش  $T_3$  برای نوشتن روی منبع B قفل نوشتن گذاشته شده است ولی Unlock نشده است و تراکنش  $T_4$  نمی تواند 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 تراکنش ها رخ داده است که دائما منتظر آزاد سازی قفل یکدیگر هستند تا بتوانند بقیه عملیات را انجام دهند. در  $T_4$  قفل خواندن روی منبع A گذاشته می شود 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 می دهد 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 بن بست همین است. بایستی در نظر داشت که با ساقط کردن یک تراکنش نمی توان به تنهایی مشکل بن بست را حل کرد بلکه باعث ایجاد مشکل جدیدی به نام قحطی خواهد شد. برای مثال یک تراکنش که قصد زدن قفل x روی داده ای است منتظر دنباله ای از تراکنش ها بماند که همگی می خواهند قفل s را روی همان منبع (داده) بزنند و این انتظار به پایان نرسد می گویم در این حالت تراکنش تعریف قفل x روی منبع دچار قحطی شده است.

جدول ۱۸: قحطی

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

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

### ۳.۵ پروتکل‌های قفل دو مرحله‌ای ۲PL

برای توضیح این پروتکل‌ها تراکنش‌های زیر را در نظر بگیرید:

جدول ۱۹: 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 اصلاً 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 زمانبندی پی در پی پذیر در برخورد باشد آنگاه تمام مشکلات مربوط به همروندی تراکنش‌ها برطرف خواهد شد.

### ۴.۵ مرحله‌ای که در فرایند پروتکل ۲PL 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 ۲۰: 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 ۲۱: 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

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 می‌کند. 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 دیتابسی 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 ۲۲: 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 معمولاً 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 بن بست و گرسنگی و سقوط آبخاری وجود ندارد! عملکرد این پروتکل با خواندن دو پروتکل ترکیبی آن حاصل می شود. اما کمترین میزان همروندی را خواهد داشت.

جدول ۲۳: زمانبندی S<sub>9</sub>

T <sub>1</sub>	X(A)	X(B)	Dec(A. amount)	W(A)	Inc(B. amount)	W(B)	C	U(A)	U(B)
T <sub>2</sub>								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 بنویسید:

جدول ۲۴: زمانبندی S<sub>10</sub>

T <sub>1</sub>	R(Q)				W(Q)				
T <sub>2</sub>		R(A)					R(Q)		
T <sub>3</sub>			R(Q)			R(P)		W(P)	
T <sub>4</sub>				W(Q)					W(A)

## قفل باینری

$$L_1(Q)R_1(Q)L_2(A)R_2(A)[L_3(Q)L_4(Q)]W_1(Q)U_1(Q)C_1R_3(Q)L_3(P)R_3(P)[L_2(Q)] \\ W_3(P)U_3(Q)U_3(P)C_3[W_4(Q)]L_4(A)[Deadlock]$$

## قفل S/X

$$S_10 = S_1^*(Q)R_1(Q)S_2^*(A)R_2(A)S_3^*(Q)R_3(Q)[X_4(Q)][X_1(Q)]S_3^*(P)R_3(P)S_2^*(Q)R_2(Q)U_2(A)U_2(Q)C_2 \\ X_3(P)W_3(P)U_3(Q)U_3(P)C_3W_1(Q)U_1(Q)C_1W_4(Q)X_4(A)W_4(A)U_4(Q)U_4(A)C_4$$

## نکته

در این روش قفل گذاری به دلیل وجود ناسازگاری RW یا WR نمی توان به نوشتن ها به دلیل خواندن های قبلی روی منبع مشترک قفل اختصاص داد. بلکه بایستی کار خواندن ها روی آن منبع مشترک به طور کامل تمام شود تا قفل را آزاد کند و سپس به نوشتن ها اختصاص یابد.

## مثال ۲

همین تمرین را با پروتکل های C۲PL S۲PL و SC۲PL انجام دهید.

## ۶ پروتکل‌های مبتنی بر مهر زمانی Timestamp

مهم‌ترین کاربرد را در دیتابیس‌های توزیع شده دارد. هر تراکنش به محض ورود، یک مهر زمانی تصاعدی به آن داده می‌شود. مهر زمانی تراکنش  $T_i$  را به صورت  $TS(T_i)$  نمایش می‌دهیم. مسلم است که برای دو تراکنش  $T_i$  و  $T_j$  که  $T_j$  دیرتر وارد سیستم شده است به صورت  $TS(T_i) < TS(T_j)$  می‌باشد. بر این اساس، پروتکل مبتنی بر مهر زمانی، تراکنش‌ها را به ترتیب مهر زمانی آن‌ها به صورت پی در پی پذیر اجرا می‌کند. اکنون که این برگه نوشته می‌شود timestamp آن به صورت 1703397265 می‌باشد. بعد از آن دوباره اقدام به تولید یک زمان دیگر کردیم که به مقدار 1703397306 رسیدیم. با مقایسه این دو زمان می‌توانید دریابید که زمان اول زودتر از زمان دوم نوشته شده است پس کوچکتر می‌باشد اما سن آن چند ثانیه بیشتر می‌باشد یا به طور کلی به شکل زیر آن را می‌نویسم:

$$1703397265 < 1703397306 \quad (۲)$$

### نکته

برای هر داده Q مهر زمانی خواندن و نوشتن آن به صورت زیر تعریف می‌شود:

- زمان  $W - TS(Q)$ : مهر زمانی نوشتن داده Q که برابر است با بزرگترین مهر زمانی تراکنشی که به طور موفقیت آمیز روی Q نوشته است.
- زمان  $R - TS(Q)$ : مهر زمانی خواندن Q که برابر است با بزرگترین مهر زمانی تراکنشی که به طور موفقیت آمیز Q را خوانده است.

## ۱.۶ قواعد

این قواعد تضمین می‌کنند که دستورات خواندن و نوشتن با هم برخورد دارند و به ترتیب مهر زمانی اجرا خواهند شد و زمانبندی‌های مربوطه پی در پی پذیر هستند.

### ۱.۱.۶ قاعده خواندن

تراکنش  $T_i$  شامل یک دستور  $Read(Q)$  است آنگاه:

۱. اگر  $TS(T_i) < W - TS(Q)$  آنگاه تراکنش  $T_i$  داده‌ای را می‌خواند که مقدارش انگار بعداً نوشته می‌شود. پس در این صورت با دستور خواندن تراکنش موافقت نمی‌شود و تراکنش رد (Reject) خواهد شد.

جدول ۲۵: بررسی مهر زمانی در قاعده خواندن

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

۲. اگر  $TS(T_i) \geq W - TS(Q)$  آنگاه دستور خواندن تراکنش  $T_i$  اجرا می‌شود و مهر زمانی خواندن Q با Max بین مهر زمانی تراکنش  $T_i$  و مهر زمانی خواندن Q مقداردهی می‌شود

جدول ۲۶: بررسی مهر زمانی در قاعده خواندن

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

## منظور از Reject شدن در چیست؟

وقتی می‌گوییم تراکنش Reject یا رد خواهد شد یعنی آنکه یا ممکن است منتظر اجرا (Wait) بماند یا ساقط و مجدداً اجرا (Abort and Rollback) شود.

### ۲.۱.۶ قاعده نوشتن

تراکنش  $T_i$  شامل یک دستور Write(Q) است آنگاه:

۱. اگر  $TS(T_i) < W - TS(Q) \vee TS(T_i) < R - TS(Q)$  باشد، آنگاه با دستور نوشتن تراکنش موافقت نمی‌شود و تراکنش  $T_i$  رد یا Reject می‌شود (اشاره به ناسازگاری WW و WR).

۲. در غیر این صورت دستور نوشتن اجرا می‌شود و مهر زمانی نوشتن Q نیز با  $TS(T_i)$  مقارنه می‌شود (اگر به صورت برعکس مورد بالا باشد یعنی:  $TS(T_i) > R - TS(Q) \wedge TS(T_i) > W - TS(Q)$ )

## نکات

- اگر هیچ تراکنشی به حالت رد شدن نرود این پروتکل‌ها فاقد بن‌بست خواهند بود اما ممکن است ترمیم پذیر نباشد.
- گراف زمانبندی‌های مهر زمانی همواره پی در پی پذیر خواهند بود چرا که همواره زمان در حال تغییر است و از زمان کوچک (سن بالاتر) به سمت زمان بزرگ‌تر (سن کوچک‌تر) می‌رود و هیچ دوری تشکیل نمی‌شود.

## مثال

زمانبندی زیر را با پروتکل‌های SYPL و CYPL و مهر زمانی بنویسید.

$$R_1(A), R_2(B), R_1(B), W_1(B), R_3(C), W_2(A)$$

## ۷ روش‌های مدیریت بن‌بست

همان‌طور که در بخش‌های قبلی نوشته شد، زمانی سیستم دچار بن‌بست می‌شود که تو تراکنش قفل‌های یکدیگر را درخواست کرده باشند یا اینکه مجموعه‌ای از تراکنش‌ها برای همیشه منتظر یکدیگر برای دریافت قفل باشند. مهم‌ترین استراتژی‌های مدیریت بن‌بست موارد زیر می‌باشد

### ۱.۷ چشم‌پوشی یا Ignore

در عمل اتفاق بن‌بست بسیار پایین و هزینه مقاله با آن گران است به همین خاطر یکی از روش‌های مقابله با بن‌بست چشم‌پوشی کردن از آن است. یک نوع وابستگی خارج از سطح سیستم است که معمولاً برنامه‌نویس یا مدیر سیستم (یا هر عامل دیگری) مسئول برطرف کردن آن است.

### ۲.۷ فرصت یا Timeout

تراکنشی که به حالت انتظار می‌رود، فقط برای مدت زمان معینی منتظر می‌ماند (اگر سرویس نگرفت) پس از آن ساقط می‌شود و دوباره بایستی درخواست اجرایش انجام شود. این روش یک روش ساده است که از بن‌بست جلوگیری می‌کند ولی امکان بروز قحطی در آن زیاد است. مقدار timeout در سیستم کار مشکل و پیچیده‌ای است.

### ۳.۷ پیشگیری یا Prevention

استفاده از پروتکل‌هایی که نبود بن‌بست را تضمین می‌کنند، مانند:

- پروتکل SC2PL
- پروتکل C2PL
- استفاده از timestamp به شرطی که رد شدن در آن به معنای انتظار نباشد بلکه به معنای Rollback باشد.

### ۴.۷ اجتناب یا Avoidance

در این روش بن‌بست تشخیص داده می‌شود و از اجرای دستوری که موجب آن شود جلوگیری به عمل می‌آورد. در این روش از مهر زمانی مخصوص بن‌بست استفاده می‌کنیم. دو حالت دارد:

- روش Wait die: تراکنش پیرتر (با timestamp کمتر) منتظر می‌ماند که تراکنش جوان‌تر قفل مربوطه را آزاد کند. در مقابل تراکنش جوان‌تر هرگز منتظر نمی‌ماند بلکه ساقط می‌شود (می‌میرد). ممکن است یک تراکنش چندین بار بمیرد.
- روش Wound wait: در این روش تراکنش پیرتر به جای انتظار تراکنش جوان‌تر را می‌کشد (kill) ولی تراکنش جوان‌تر منتظر تراکنش پیرتر می‌ماند. در این روش برخلاف روش Wait die تراکنش پیرتر اولویت بالاتری دارد و با گشتن تراکنش جوان فرایند اجرا را به قبضه خود در می‌آورد.

## نکته

تعداد تراکنش‌های جوان‌تر که منتظر هستند بیشتر از تراکنش‌هایی با مهر زمانی پیرتر است. در هر دو روش گفته شده، تراکنشی که ساقط می‌شود با همان مهر زمانی اصلی خود (مثلاً ثانیه ۱۰ وارد شده و زمان کنونی ثانیه ۱۰۰ است پس ۱۰ از ۱۰۰ پیرتر خواهد بود) آغاز به کار مجدد می‌کند که به همین دلیل باعث می‌شود تراکنش جوان پس از چند بار مردن، بالاخره پیر شود و بتواند تراکنش را قبضه کند و از بروز قحطی جلوگیری کند.

## ۵.۷ تشخیص و رفع بن‌بست یا Detection and resolve

در این روش از گراف انتظار استفاده می‌کنیم. به ازای تمام حالاتی که بین تراکنش‌های  $T_i$  و  $T_j$  حالت‌های  $ww$  و  $WR$  و  $RW$  رخ می‌دهد را یک یال می‌کشیم. اگر در گراف دور ایجاد شود بن‌بست رخ داده است. در این حالت باید تراکنشی که تعداد دورهای بیشتری دارد را kill کنیم تا دور از بین برود یا اینکه تشخیص دهیم که کدام تراکنش کار بیشتری انجام داده است که اگر اکثر فرایندها را انجام داده در پایان کار آن را kill کنیم.



## ۸ واحد مدیریت ترمیم و الگوریتم‌ها

دو تراکنش زیر چه قوانینی را تقض کرده‌اند؟ توضیح دهید

$T_1$	$T_2$
:	:
:	:
:	:
:	:
Failure	:
:	Commit
:	Failure

در تراکنش  $T_1$  قانون Atomicity رعایت نشده است، چرا که حین اجرا به مشکل برخورد و تراکنش اول به پایان نرسیده است. یکی از مهم‌ترین قوانین ACID بخش Atomicity است که تراکنش یا باید کامل انجام شود که کامیت شود یا نهایتاً سقوط کند. در تراکنش  $T_2$  قانون Durability رعایت نشده است. یعنی تراکنش به صورت کامل انجام شده است و حتی کامیت هم صورت گرفته اما در حافظه و یا قسمتی که مربوط به ثبت داده این تراکنش است منعکس نشده است.

### ۱.۸ انواع خطاهای مربوط به سیستم دیتابیس

#### ۱.۱.۸ خطاهای تراکنشی

۱. خطای منطقی: تراکنش با توجه به قوانینی که برای آن تعریف کردیم، اشتباه انجام شود، مانند خطای تقسیم بر صفر. این نوع خطا بر روی کل دیتابیس تاثیر گذار نیست

۲. خطای سیستمی: مانند Kill شدن تراکنش‌ها به هر دلیلی

#### ۲.۱.۸ خطاهای سیستمی

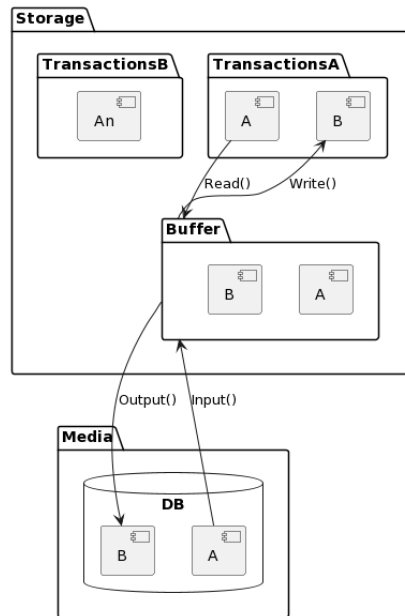
خطای سیستمی می‌تواند مربوط به سخت‌افزار و نرم‌افزار شود، برای مثال سیستم هنگ کرده باشد، رم پر شده باشد یا از نظر سخت‌افزاری مشکلی برای آن پیش آمده باشد.

#### ۳.۱.۸ خطای رسانه‌ای

خرابی حافظه (دیسک) که می‌تواند سخت‌افزاری هم باشد اما بخش بسیار مهم یک سیستم دیتابیس را شامل می‌شود که می‌تواند روی همه داده‌ها تاثیر گذار باشد.

#### یادآوری

واحد مدیریت همروندی مسئول برقرار خاصیت Isolation است در حالی که واحد مدیریت ترمیم مسئول انجام شدن کامل تراکنش‌ها و خاصیت ماندگاری است.



شکل ۴: روال دسترسی به داده برای انجام تراکنش

## ۲.۸ روال دسترسی به داده‌ها برای انجام تراکنش

داده‌های دیتابیس روی رسانه‌ها قرار دارند که به منظور پردازش آنها توسط تراکنش‌ها، باید به طور موقت به بخشی از حافظه اصلی منتقل شوند که آنرا بافر<sup>۴</sup> می‌گوییم. آوردن داده از دیسک به بافر و بازگرداندن نتایج از بافر به دیسک (معمولاً به صورت بلاک‌های حافظه) توسط عملگرهایی (توابع) انجام می‌شود که `input()` و `output()` نامیده می‌شود. هر تراکنشی که می‌خواهد با داده‌ای کار کند، یک کپی از آن داده در بافر را در ناحیه کاری<sup>۵</sup> خاص خود می‌برد و دسترسی‌های بعدی آن تراکنش روی کپی محلی اش انجام می‌شود و پس از اتمام کار، این کپی محلی به بافر منتقل می‌شود. باز هم برای این منظور نیاز به عملگرهایی داریم که آنها را به ترتیب `Read()` و `Write()` می‌نامیم. شکل ۴ این موضوع را نشان می‌دهد. به طور خلاصه، خواندن و نوشتن روی داده‌ها در `workarea` رخ می‌دهد که بعد از آن انتقال داده‌ها اول روی حافظه اصلی و منطقی (بافر) صورت می‌گیرد و در نهایت از حافظه اصلی به حافظه جانبی و یا دیسک منتقل می‌شود.

### نکته

اگر داده در بافر حضور نداشته باشد درخواست خواندن را به دیسک ارسال می‌کند.

## ۳.۸ الگوریتم‌های ترمیم

الگوریتم‌های ترمیم برای حفظ جامیت و پایداری شامل دو مرحله می‌شوند، (به طور کلی همان بک‌آپ و ری‌استور کردن داده‌ها می‌باشد)

### ۱.۳.۸ مرحله اول

در این مرحله، حین اجرای تراکنش اطلاعاتی برای ترمیم نیاز است را در جایی ثبت می‌کند.

### ۲.۳.۸ مرحله دوم

در این مرحله، بعد از آنکه سیستم بالا آمد و وضعیت کلی مناسبی داشت اطلاعاتی که در مرحله قبل ثبت شده در این مرحله خوانده می‌شود تا خواص `Atomicity` و `Durability` برقرار باشد.

<sup>۴</sup> Buffer  
<sup>۵</sup> Workarea

## طبق خاصیت Durability

اثرات تراکنشی که انجام شده باید دائمی و همیشگی باشد.

## طبق خاصیت Atomicity

تراکنشی که نیمه کاره متوقف شده است نباید هیچ اثری در دیتابیس داشته باشد.

## ۴.۸ عملیات Redo

زمانی عملیات را Redo می‌کنیم که تراکنش به صورت کامل کامیت شده و در انعکاس اطلاعات در حافظه به مشکل خورده است (توجه شود تمام فرایند از اول انجام می‌شود).

## ۵.۸ عملیات Undo

زمانی عملیات را Undo می‌کنیم که خاصیت جامعیت نقض شده باشد. برای مثال تراکنش ساقط شده باشد. به خاطر همین دستورات را مرحله به مرحله خنثی می‌کنیم (عملیات Undo از پایین به بالا انجام می‌شود. از آخرین دستور تا اولین دستور).

## ۶.۸ رویکردهای الگوریتم ترمیم

- کارنامه ۶
- رونوشت

## ۷.۸ رویکرد کارنامه