پایگاه داده پیشرفته

دکتر شجاعی مهر
علیرضا سلطانی نشان

۱ آذر ۱۴۰۲

فهرست مطالب

۱ مفاهیم

۱.۱ تراکنش

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

- Create
- Read
- Update
- Delete

۲.۱ قوانین ACID

۱.۲.۱ 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.

۲.۲.۱ 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.

۳.۲.۱ 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.

۴.۲.۱ 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.

۳.۱ 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:

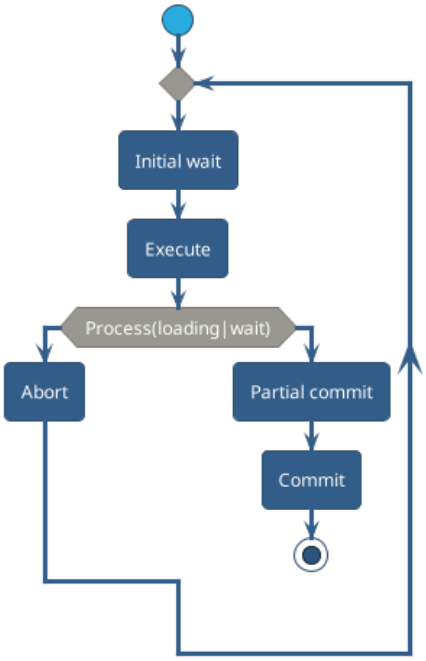
- Commit: تراکنش درنهایت تایید و انجام می‌شود
- Abort: تراکنش در نهایت سقط یا صرفه نظر می‌شود

۱.۳.۱ 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

۴.۱ همروندی

۱.۴.۱ مزیت همروندی

۱. افزایش سرعت گذردهی یا throughput

۲. کاهش میانگین زمان پاسخدهی به تراکنش مورد نظر

۲.۴.۱ 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

۵.۱ زمان‌بندی

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

۶.۱ 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:

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

۷.۱ 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

۸.۱ 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

۹.۱ 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

۱۰.۱ 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.

۱.۱۰.۱ 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].

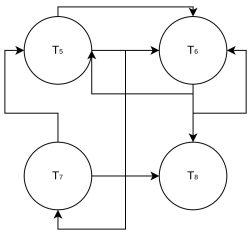
[۱] Kill transaction

برای مثال تراکنش‌های زیر را در نظر بگیرید:

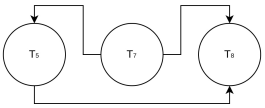
جدول ۷: تراکنش‌های بانکی

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.

۱۱.۱ 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.

۱۲.۱ مثال اول پی در پی پذیری در دید

جدول ۸: پی در پی پذیری در دید

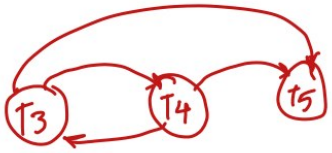
T_5			W(Q)		
T_6	R(Q)				
T_7		W(Q)			W(Q)
T_8				R(Q)	

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.



۱۳.۱ 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.

۱۴.۱ 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.

این بخش تکمیل نشده است. با استفاده از یک مسئله که عکس آن در کلاس گرفته شده است.

۱۵.۱ 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.

۱۶.۱ زمانبندی ترمیم پذیر یا Recoverable scheduling

زمانبندی را ترمیم پذیر یا RC می‌گوییم که اگر برای تمام T_jهایی که از T_i می‌خوانند کامیت اثباط تراکنش T_i قبلی صورت گرفته باشد تا T_j بتواند مقدار زمانبندی وابسته قبلی را دریافت کند.

۱.۱۶.۱ سقوط‌های آبشاری یا Cascading Aborts

مثال ۱:

جدول ۱۰: زمانبندی ترمیم ناپذیر

T_1	R(A)	W(A)	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 تراکنشش RollBack خواهد شد و به صورت صحیح کامل نشده است.

مثال ۲:

جدول ۱۱: زمانبندی ترمیم ناپذیر

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

در این مثال هم مانند مثال قبل چون در تراکنش اول عمل Abort رخ داده است کل تراکنش RollBack می‌شود و R(A) نمیتواند مقدار A را بخواند فلذا این جدول از تراکنش‌ها درست نیستند و ترمیم ناپذیر اند.

مثال ۳:

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

در این مثال تراکنش T_1 به دو قسمت تقسیم می‌شود. زمانی که کامیت کرده است و زمانی که دیتا ساقط شده است. وقتی که عمل W(A) صورت میگیرد در این قسمت تراکنش اول کامیت می‌شود و در راستای آن تراکنش T_2 به دلیل وابستگی به منبع A به تراکنش بالایی خود با موفقیت می‌تواند مقدار را دریافت کند. این جدول تراکنش فاقد سقوط آبشاری است. چرا که در ابتدا قسمتی از عملیات کامیت شده و قسمت دیگر سقوط کرده است.

۱۷.۱ 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.

جدول ۱۲: نمونه‌ای از فرایند ACA

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

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.

۱۸.۱ زمانبندی‌های محض یا Strict

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

جدول ۱۳: نمونه‌ای از زمانبندی محض

T_1	W(F) C	W(G) C		
T_2			R(F)	R(G)

مثال: زمانبندی زیر را از نظر محض بودن، ترمیم پذیری و ACA بررسی کنید.
نکته یکی از قوانین ترمیم پذیری عدم وجود سقوط‌های آبشاری است، پس اگر یک زمانبندی ACA باشد پس ترمیم پذیر می‌باشد.

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

- زمانبندی بالا محض نیست چرا که بعد از هر بار نوشتن باید عمل کامیت صورت گیرد.
- زمانبندی بالا ترمیم پذیر است چرا که هیچ سقوطی رخ نداده است بلکه تراکنش T_1 در انتها کامیت شده و میتواند در تراکنش T_2 خوانده شود.

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)

۱۹.۱ پروتکل‌های کنترل همروندی

بعد از دیدن دستور ۳ کار انجام می‌شود:

۱. اجرای دستور

۲. به تاخیر انداختن دستور

۳. نپذیرفتن دستور یا سقوط آن

۱.۱۹.۱ پروتکل‌های مبتنی بر قفل

در این نوع پروتکل واحدی به نام Lock Manager وارد تراکنش‌ها می‌شود و بررسی میکند اگر ناسازگاری ،ww wr یا rw وجود نداشته باشد اجازه خواندن را به تراکنش داده خواهد شد

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 باینری

۲. قفل‌های اشتراکی/انحصاری یا Shared Exclusive Lock: یک قفل برای خواندن (S) استفاده می‌کند و یک قفل برای نوشتن Mutex یا 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
- قفل گذاری درست باعث می‌شود تا زمانبندی درست داشته باشیم

مثال:

$$S_1 = R_1(A)\ W_1(A)\ A_1\ W_2(A)\ W_2(B)\ C$$

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

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 خطی:

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

حل:

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

۲۰.۱ بن بست و تحطی

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

جدول ۱۴: 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. 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.

۲۱.۱ پروتکل‌های قفل دو مرحله‌ای ۲PL ۲