

درس 6 پایگاه داده پیشرفته

کنترل همروندی (2)

پروتکل های بر مبنای برجسب (مهر) زمانی Timestamp

- روش دیگر برای تعیین ترتیب سریالی استقاده از زمان شروع تراکنش هاست.
- به عبارتی هرگاه بر سردستری به داده ای دو تراکنش رقابت می کنند، از روی زمان شروع تراکنش ها و تاریخچه اجرای آن ها تصمیم گیری می شود.
- برجسب زمانی
 - به هر تراکنش، یک مهر زمانی اختصاصی و یکتا نسبت داده می شود.
 - این مهر زمانی توسط سیستم پایگاه داده، قبل از اینکه شروع به اجرا کند تعیین می شود.
- ویژگی های مهر زمانی:
 - یکتا باشد یعنی هر تراکنش مهر زمانی مختص خود را دارا باشد.
 - اگر تراکنش زمان شروعی قبل از تراکنش داشته باشد با مهر زمانی به گونه ای انتخاب شود که: $TS(T_i) < TS(T_j)$

پیاده سازی مهرزمانی

□ دوروش ساده برای پیاده سازی این الگوریتم وجود دارد :

□ استفاده از مقدار ساعت سیستم `system clock` بعنوان مهرزمانی، یعنی هرگاه یک

تراکنش شروع به اجرا کرد ساعت سیستم را به عنوان مهرزمانی اختصاص دهیم.

□ استفاده از یک شمارنده منطقی که بعد از اینکه یک مهرزمانی جدید نسبت داده شد

افزایش می یابد، به این معنا که مهرزمانی تراکنش معادل با مقدار شمارنده در لحظه

شروع تراکنش خواهد بود.

□ برچسب های زمانی تراکنش ها ترتیب پذیری آنها را تعیین می کند.

بنابراین اگر $TS(T_i) < TS(T_j)$ ، سیستم باید تضمین کند که **زمانبند (برنامه)**

ایجاد شده معادل با برنامه ترتیبی باشد که در آن تراکنش T_i قبل از T_j آمده

باشد.

پروتکل برچسب (مهر) زمانی

- برای پیاده سازی این الگو، به هر عنصر داده Q دو مقدار برچسب زمانی نسبت می دهیم:
- $W\text{-TimeStamp}(Q)$ که بزرگترین مقدار برچسب زمانی هرتراکنشی است که عمل $write(Q)$ را به صورت موفقیت آمیز انجام داده اند نشان می دهد.
- $R\text{-TimeStamp}(Q)$ که بزرگترین مقدار برچسب زمانی هرتراکنشی است که عمل $read(Q)$ را به صورت موفقیت آمیز انجام داده اند نشان می دهد.
- این برچسب های زمانی با هر دستور **خواندن** و **نوشتن** جدیدی که انجام می شود به روزرسانی می شوند.

پروتکل برچسب (مهر) زمانی $Read(Q)$

□ 1- فرض کنید که تراکنش T_i درخواست $read(Q)$ را می دهد.

□ اگر $TS(T_i) < W-TS(Q)$ ، در نتیجه T_i نیاز به خواندن داده ای دارد که قبلاً دوباره نویسی شده است. بنابراین، عمل خواندن **رد شده**، و T_i برگردانده (rolled back) می شود. (اجازه خواندن ندارد زیرا ترتیب پذیر نیست)

□ اگر $TS(T_i) \geq W-TS(Q)$ ، عمل خواندن اجرا می شود و $R-TS(Q)$ برابر مقدار پیشینه $R_TS(Q)$ و $TS(T_i)$ قرار داده می شود.

پروتکل برجسب (مهر) زمانی $Write(Q)$

□ 2- فرض کنید که تراکنش T_i درخواست $write(Q)$ را می دهد.

□ اگر $TS(T_i) < R-TimeStamp(Q)$ ، پس مقدار Q ای که T_i تولید می کند قبلا مورد نیاز بوده

است ، و سیستم فرض می کند که مقدار هیچ گاه تولید نشده است. بنابراین سیستم عمل نوشتن را رد (reject) می کند و T_i را برمی گرداند (rolled back).

□ اگر $TS(T_i) < W-TimeStamp(Q)$ ، پس T_i سعی می کند که یک مقدار قدیمی (منسوخ) را

روی Q بنویسد. بنابراین سیستم عمل نوشتن را رد می کند و T_i را برمی گرداند.

□ در غیر اینصورت، سیستم عمل نوشتن را اجرا می کند و مقدار $W-TimeStamp(Q)$ را برابر با

$TS(T_i)$ قرار می دهد.

□ **توجه:** اگر تراکنش T_i در نتیجه درخواست خواندن یا نوشتن توسط کنترل همروندی برگردانده شود، سیستم یک *timestamp* جدید به آن نسبت می دهد و آنرا دوباره شروع می کند.

مثال

برای نشان دادن این پروتکل تراکنش های T_{14} و T_{15} را در نظر بگیرید:

T_{14}	T_{15}
read(B)	read(B) $B := B - 50$ write(B)
read(A)	read(A)
display($A + B$)	$A := A + 50$ write(A) display($A + B$)

باید فرض کنیم که مهر زمانی قبل از اینکه اولین دستور تراکنش اجرا شود تخصیص داده می شود لذا با توجه به شکل زیر داریم:

$$TS(T_{14}) < TS(T_{15})$$

زمانبند تحت پروتکل مهر زمانی امکان پذیر خواهد بود.

Figure 16.13 Schedule 3.

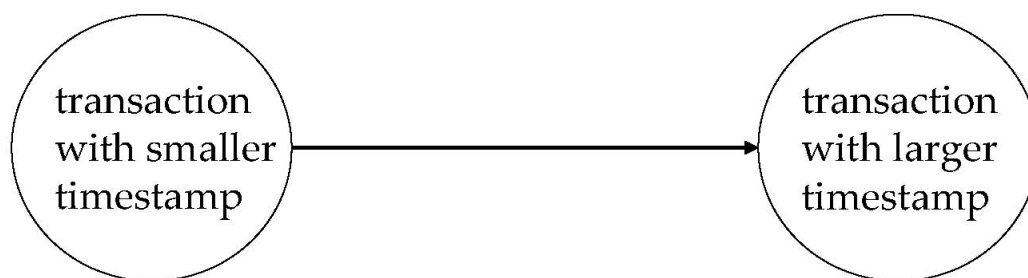
پروتکل برجسب (مهر) زمانی

- پروتکل ترتیب دهی براساس برجسب زمانی، قابلیت ترتیب پذیری را تضمین می کند.
- این به این دلیل است که اعمال دارای برخورد به ترتیب برجسب -های زمانی پردازش می شوند.
- برخی از زمانبند ها هستند که در پروتکل 2PL معتبر هستند ولی در TS نامعتبرند و بالعکس.

پروتکل برجسب (مهر) زمانی-مزایا

□ مزایای پروتکل:

1. در این پروتکل انتظار وجود ندارد، زیرا برای خواندن هیچگاه انتظار نداریم یا خواندن انجام می شود یا تراکنش هایی که می خواهند خواندن انجام دهند برگشت داده می شوند.
2. تضمین می کند بن بست نداشته باشیم، بزرگترین مزیت این پروتکل، زیرا هیچ تراکنشی منتظر نمی ماند، چون انتظار نداریم دور نداریم و بن بست نداریم.
3. هزینه نگهداری و مدیریت قفل ها را ندارد.
4. می تواند برخی از ترتیب پذیرها در برخورد را معتبر نشان دهد.



پروتکل برجسب (مهر) زمانی- معایب

□ معایب :

1. هزینه بالاست زیرا مشکل را با انتظار حل نمی کند بلکه با **برگشت** دادن حل می کند. تمام عملیاتی که برای انجام تراکنش انجام گرفته است نادیده گرفته می شوند.
2. برای تراکنش های بلندی که همروند با تراکنش های کوتاه اجرایی شوند امکان **گرسنگی** وجود دارد.
3. هزینه نگهداری و مدیریت مهر زمانی را دارد.

پروتکل برجسب (مهر) زمانی

- این پروتکل می تواند زمانبند هایی که قابل ترمیم نیستند را نیز تضمین کند. لذا می توان آن را به گونه ای توسعه داد که امکان برنامه های ترمیم پذیر را ایجاد کند.
- این روش ها عبارتند از:
- قابلیت ترمیم و اجتناب از برگشت آبشاری را می توان با انجام **تمام نوشتن** ها با هم و در پایان تراکنش تضمین کرد.
- به این معنی که هنگامی که نوشتن ها در حال انجام هستند هیچ تراکنشی اجازه دسترسی به داده های در حال نوشتن را ندارد.

توسعه پروتکل

□ قابلیت ترمیم تنها را می توان با پیگیری نوشتن های اعمال نشده `uncommitted` و موکول کردن اعمال تراکنش `Ti` به پایان موفقیت آمیز همه تراکنش هایی که مقداری را نوشته اند و تراکنش `Ti` آن را خوانده است تضمین کرد.

□ قابلیت ترمیم و اجتناب از برگشت آبشاری را همچنین می توان با استفاده از یک فرم محدود قفل کردن تضمین کرد، که بموجب آن خواندن عناصر **تثبیت** نشده تا زمانی که تراکنش **انجام** و تثبیت شود به تعویق می افتد. تمرین 16 و 26

قانون نوشتن Thomas (اصلاح پروتکل)

- در اینجا تغییری در پروتکل ترتیب‌دهی براساس برچسب زمانی ارائه می‌شود که همروندی بالقوه بیشتر را اجازه می‌دهد. مثال:

T_{16}	T_{17}
read(Q)	write(Q)
write(Q)	

- از آنجا که T_{16} قبل از T_{17} شروع شده، انتظار داریم که $TS(T_{16}) < TS(T_{17})$ باشد. در این صورت اعمال $read(Q)$ تراکنش و $write(Q)$ تراکنش با موفقیت اجرا می‌شوند.

- هنگامی که T_{16} تلاش می‌کند که $write(Q)$ را انجام دهد، داریم:

$$W - timestamp(Q) = TS(T_{17}) \quad \text{چون} \quad TS(T_{16}) < W - timestamp(Q) \quad \square$$

در نتیجه عمل رد می‌شود و تراکنش باید برگشت داده شود.

- اگرچه برگشت T_{16} در پروتکل ترتیب‌دهی براساس برچسب زمانی لازم است، این عمل لزومی ندارد. زیرا مقداری که T_{16} تلاش می‌کند بنویسد، مقداری است که هیچگاه نیازی به خواندن آن نیست.

قانون نوشتن Thomas (اصلاح پروتکل)

- هر تراکنش T_i با $TS(T_i) < TS(T_{17})$ که تلاش کند $read(Q)$ را انجام دهد برگردانده می شود، زیرا که $TS(T_i) < W-TS(Q)$ است.
- و هر تراکنش T_j با $TS(T_j) > TS(T_{17})$ باید مقدار Q نوشته شده توسط T_{17} را بخواند و نه مقدار نوشته شده توسط T_{16} را.
- این مشاهده نوع اصلاح شده ای از پروتکل ترتیب دهی براساس برچسب زمانی که در آن تحت شرایط خاص از اعمال **نوشتن قدیمی** صرف نظر می شود، نتیجه می دهد.
- قوانین خواندن پروتکل تغییر نمی کنند، اما قوانین نوشتن پروتکل کمی متفاوت از قوانین ترتیب دهی براساس برچسب زمانی می باشند.

قانون نوشتن Thomas (اصلاح پروتکل)

□ اگر تراکنش عمل $write(Q)$ را ارائه داده باشد:

۱. اگر $TS(T_i) < R - timestamp(Q)$ ، آنگاه مقدار Q که T_i ایجاد می‌کند قبلاً مورد نیاز بوده

است، و تصور می‌شود که مقدار هیچ گاه ارائه نشده است. بنابراین سیستم عمل را رد می‌کند و تراکنش را برمی‌گرداند.

۲. اگر $TS(T_i) < W - timestamp(Q)$ ، آنگاه T_i تلاش می‌کند که یک مقدار قدیمی Q را

بنویسد. بنابراین از عمل نوشتن صرف‌نظر می‌شود.

۳. در غیر اینصورت سیستم عمل را اجرا می‌کند و $W - timestamp(Q)$ را برابر با $TS(T_i)$

قرار می‌دهد.

قانون توماس امکان هم‌رندی بیشتری را فراهم می‌کند، و اجازه زمانبند های ترتیب پذیرنمایی را که ترتیب پذیر در برخورد نیستند می‌دهد.

پروتکل های مبتنی بر تعیین اعتبار *Validation based protocols*

□ در مواردی که قسمت عمده ای از تراکنش ها فقط خواندنی هستند، برخورد بین تراکنش ها پایین بوده بنابراین، بسیاری از این تراکنش ها، **اگر بدون نظارت کنترل همروندی** هم اجرا شوند، سیستم را در حالت سازگاری قرار می دهند.

□ کنترل همروندی، سربار (overhead) اجرای کد و تأخیر تراکنش ها را اعمال می کند. لذا بهتر است از الگویی که سربار کمتری را تحمیل می کند استفاده کنیم. مشکل در کاهش سربار این است که پیشاپیش نمی دانیم چه تراکنش هایی با هم برخورد دارند و بنابراین نیاز به الگوی نظارتی خواهیم داشت.

پروتکل مبتنی بر تعیین اعتبار

- در این روش، فرض می‌کنیم که هر تراکنش در سه فاز اجرا می‌شود بسته به این که فقط خواندنی یا قابل بروزرسانی باشد. این سه مرحله عبارتند از:
- **فاز خواندن و اجرا.** در این فاز، سیستم تراکنش را اجرا می‌کند. اما تمام اعمال نوشتن در متغیرهای محلی انجام شده و پایگاه داده به‌روزرسانی نمی‌شود.
- **فاز تعیین اعتبار.** بررسی می‌شود که آیا با اعمال تغییرات در پایگاه داده ترتیبی بودن نقض می‌شود یا خیر.
- **فاز نوشتن.** اگر مرحله تعیین اعتبار موفقیت‌آمیز باشد، تغییرات ایجاد شده توسط تراکنش در پایگاه داده اعمال می‌شود در غیر این صورت تراکنش برگشت داده می‌شود.
- برای اعتبارسنجی باید بدانیم چه وقتی این مراحل باید انجام شوند برای این منظور، به هر تراکنش سه برچسب زمانی نسبت داده می‌شود:

پروتکل مبتنی بر تعیین اعتبار

به هر تراکنش سه برچسب زمانی نسبت داده می شود:

۱. $Start(T_i)$. زمانی که تراکنش شروع به اجرا می کند.

۲. $Validation(T_i)$. زمانی که تراکنش وارد فاز تعیین اعتبار می شود و خواندن تمام شده است.

۳. $Finish(T_i)$. زمان پایان مرحله نوشتن.

□ ترتیب متوالی را با استفاده از تکنیک ترتیب دهی برچسب زمانی ، و استفاده از مقدار برچسب زمانی انجام می دهیم

پروتکل مبتنی بر تعیین اعتبار

- تست اعتبار برای تراکنش T_j نیاز دارد که برای تمام تراکنش های T_i که شرط $TS(T_i) < TS(T_j)$ برقرار باشد، یکی از دو شرط زیر موجود باشد:
- $Finish(T_i) < Start(T_j)$ از آنجا که T_i قبل از آغاز T_j پایان یافته است، ترتیب متوالی برقرار است.
- مجموعه عناصر داده نوشته شده توسط T_i عضو مشترکی با عناصر داده خوانده شده توسط T_j نداشته باشد، و T_i فاز نوشتن را قبل از اینکه T_j فاز تعیین اعتبار را آغاز کند به پایان برساند.
$$Start(T_j) < Finish(T_i) < Validation(T_j)$$
- این شرط اطمینان می دهد که نوشته های T_i و T_j همپوشانی ندارند. زیرا که نوشته های T_i نمی توانند روی خواندن های T_j اثر بگذارند، و از آنجا که T_j نمی تواند روی خواندن های T_i اثر بگذارد، ترتیب متوالی حفظ می شود.

پروتکل مبتنی بر تعیین اعتبار

□ مثال: زمانبند زیر را در نظر بگیرید که دو تراکنش T_{25} , T_{26} اجرا می شوند. با توجه به شکل داریم: $TS(T_{25}) < TS(T_{26})$ پس تست اعتبار، موفقیت آمیز است. زیرا نوشتن T_{15} پس از validation صورت می گیرد. بنابر این T_{25} مقادیر قبلی A, B را می خواند و لذا این زمانبند ترتیب پذیر است.

T_{25}	T_{26}
read (B)	read (B) $B := B - 50$ read (A) $A := A + 50$
read (A) < validate > display (A + B)	< validate > write (B) write (A)

□ به این روش Optimistic concurrency control

نیز گفته می شود. با توجه به اینکه تراکنش ها

خوش بینانه اجرا شده و فرض می شود

آنها قادر به تمام کردن اجرا و اعتبار سنجی

در پایان هستند.

الگوهای چند نسخه‌ای (multi version schemes)

- الگوهای کنترل همروندی که تاکنون بحث شد متوالی بودن را یا با به **تأخیر انداختن** یک عمل یا با **خاتمه دادن** به تراکنشی که درخواست عمل را کرده است تضمین می کنند.
- برای مثال یک عمل خواندن ممکن است به دلیل اینکه مقدار مناسب هنوز نوشته نشده است به تعویق افتد، یا ممکن است به دلیل **دوباره نویسی** مقدار موردنظر رد شود.
- می توان با نگهداری **کپی های قدیمی** عناصر داده در سیستم از این مشکلات جلوگیری کرد.
- در الگوهای کنترل همروندی چند نسخه ای،
- هر عمل $write(Q)$ نسخه جدیدی از Q را ایجاد می کند.
- هنگامی که یک درخواست $read(Q)$ را می دهد، مدیر کنترل همروندی یکی از نسخه های Q را برای خواندن انتخاب می کند.
- الگوی کنترل همروندی باید تضمین کند که نسخه انتخاب شده متوالی بودن را تضمین می کند.

ترتیب دهی چند نسخه ای براساس برچسب زمانی

- در این روش به هر تراکنش در سیستم، یک برچسب زمانی ایستا و یکتا $TS(T)$ منسوب می شود. و به هر عنصر داده Q یک دنباله از نسخه ها $\langle Q_1, Q_2, \dots, Q_m \rangle$ منسوب می شود. و هر نسخه Q_i سه فیلد داده دارد :
- محتوی: که مقدار نسخه Q_k را داراست.
- $W - timestamp(Q_k)$ که برچسب زمانی تراکنشی را که نسخه را بوجود آورده نشان می دهد.
- $R - timestamp(Q_k)$ که بزرگترین برچسب زمانی تراکنش هایی را که عمل خواندن را بصورت موفقیت آمیز انجام داده اند نشان می دهد.

ترتیب دهی چند نسخه ای براساس برچسب زمانی

تراکنش T_i نسخه جدیدی از Q ، Q_k را با انجام $write(Q)$ بوجود می آورد و مقدار $W-timestamp(Q_k)$ و $R-timestamp(Q_k)$ برابر با $TS(T_i)$ قرار داده می شود.

$R-timestamp(Q_k)$ هر زمان که تراکنش T_j ، Q_k را می خواند و $R-timestamp(Q_k) < TS(T_j)$ برقرار می باشد به روزرسانی می شود.

ترتیب دهی چند نسخه ای براساس برچسب زمانی

الگوی چندنسخه‌ای ترتیب‌دهی براساس برچسب زمانی که ارائه خواهد شد متوالی بودن را تضمین می‌کند. این الگو بصورت زیر کار می‌کند.

- فرض کنید که تراکنش T_i درخواست $read(Q)$ یا $write(Q)$ را بدهد. و Q_k

نسخه ای از Q را نشان دهد که $W - timestamp$ آن بزرگترین $W - timestamp$

کوچکتر یا مساوی $TS(T_i)$ باشد. (Largest WTS less than or equal)

۱. اگر T_i درخواست $read(Q)$ را بدهد، مقدار برگردانده شده محتوی نسخه Q_k می‌باشد.

۲. اگر T_i درخواست $write(Q)$ را بدهد و $TS(T_i) < R - timestamp(Q_k)$ ، سیستم

تراکنش T_i را برمی‌گرداند. از طرف دیگر اگر $TS(T_i) = R - timestamp(Q_k)$ ، سیستم

محتوی Q_k را دوباره نویسی می‌کند؛ درغیراینصورت نسخه جدیدی از Q را می‌سازد.

ترتیب دهی چند نسخه ای براساس برجسب زمانی

- توجیه قانون اول روشن است. تراکنش جدیدترین نسخه ای را که قبل از آن از نظر زمانی آمده است می خواند.
- براساس قانون دوم تراکنش در صورتی که برای نوشتن بسیار دیر شده باشد، برگردانده می شود.
- نسخه هایی که دیگر مورد نیاز نیستند براساس قانون زیر حذف می شوند.
- به فرض دو نسخه Q_k و Q_j از عنصر داده موجودند و هر دو نسخه $W-TS$ ای کوچکتر از برجسب زمانی قدیمی ترین تراکنش در سیستم دارند. بنابراین نسخه **قدیمی تر** از میان Q_k و Q_j دیگر مورد استفاده قرار نخواهد گرفت، و می تواند حذف شود.



ترتیب دهی چند نسخه ای براساس برچسب زمانی

- الگوی چندنسخه ای ترتیب دهی براساس برچسب زمانی دارای این خصوصیت مطلوب است که :
 - هیچ درخواست خواندنی بی نتیجه نمی ماند و هیچ وقت مجبور به انتظار نمی شود.
 - در یک سیستم پایگاه داده مرسوم که خواندن عمل معمول تری نسبت به نوشتن است، این مزیت از نظر کاربردی مهم است.
- این الگو با وجود این دارای دو خصوصیت نامطلوب است.
 - اولی اینکه خواندن یک عنصر داده نیاز به به روزرسانی فیلد R-TS دارد که موجب دو دسترسی به دیسک به جای یکی می شود.
 - دومی اینکه، برخورد بین تراکنش ها از طریق بازگرداندن به جای انتظار حل می شود.
- این الگوی چندنسخه ای ترتیب دهی براساس برچسب زمانی ، قابلیت ترمیم و اجتناب از نابودی آبخاری را تضمین نمی کند، اما می توان آنرا طوری تغییر داد که قابل ترمیم باشد و از نابودی آبخاری اجتناب کند.