استفاده از مهر زمانی برای کنترل همروندی: برای هر مهر زمانی T_i $TS(T_i)$

حال برای هر فقره داده مثل D_i هم دو مهر زمانی تعریف می‌کنیم.

۱- مهر زمانی خواندن $TS\text{-}R(D_i)$: برابر بزرگترین مهر زمانی تراکنشهایی که داده D_i را خوانده یا به عبارت دیگر مهر زمانی جوانترین تراکنشی که D_i را خوانده

۲- مهر زمانی نوشتن داده $TS\text{-}R(D_i)$: برابر با مهر زمانی بزرگترین تراکنشی که داده D_i را نوشته یا تغییر داده.

حال با توجه به این مجموعه (سه) مهر زمانی:

- مهر زمانی تراکنش‌ها $TS(T_i)$
- ~ ~ خواندن $TS\text{-}R(D_k)$
- ~ ~ نوشتن $TS\text{-}W(D_k)$

نظمی ایجاد می‌کنیم که همروندی ایجاد شود.

الگوریتم‌های مختلفی در این رابطه هست، یکی از معروفترین الگوریتم‌ها، الگوریتم پروتکل To است. To => Time Stamp ordering

الگوریتم To ⇐

۱) در عمل خواندن، تراکنش T_i دستور خواندن $R(D)$ را اجرا می‌کند.

الف) اگر $TS(T_i) < TS\text{-}W(D)$ * $TS(T_i)$ کوچکتر از مهر زمانی نوشتن داده D باشد

(این یعنی در تراکنش جدیدتری این داده را نوشته، ممکن است تراکنش جدید هنوز به تثبیت نرسیده باشد، بنابراین یک تراکنش قدیمی می‌خواد این داده را بخونه، اگر اون تراکنش جدیدی که این داده را نوشته به تثبیت نرسه با مشکل مواجه می‌شود)

* دستور خواندن اجرا نمی‌شود و تراکنش T_i طرد می‌شود، چون مشکل خواندن داده نامعتبر اتفاق می‌افتد.

ب) در غیر اینصورت یعنی اگر بزرگتر باشد $TS(T_i) \geq TS\text{-}w(D)$ دستور خواندن اجرا می‌شود

الان یک تراکنش جدید داریم که می‌خواد بخونه پس باید زمانهر داده D آپدیت شود و $TS\text{-}R(D)$ عوض شده چون یکی دیگه اومده خونده TS قدیمی $TS(T_i)$ که جدید تر است $\max =$

این در عمل خواندن بود.

$$TS\text{-}R(D) = \max\{TS\text{-}R(D),\ TS(T_i)\}$$

۲) در عمل نوشتن: تراکنش T_i دستور نوشتن $w(D)$ را صادر می‌کند.

الف) اگر $TS(T_i) < TS\text{-}R(D)$ پس درخواست نوشتن رد می‌شود و تراکنش T_i طرد می‌شود.

$TS(T_i) < TS\text{-}R(D)$ یعنی یک تراکنش اومده جدید تر D را خونده، پس اگر اطلاعات D را بنویسم و تغییر دهم پس اشتباه ایجاد می‌شه در تراکنشی که داده را خونده، پس درخواست نوشتن رد می‌شود و تراکنش T_i طرد می‌شود.

ب) اگر $TS(T_i) < TS\text{-}w(D)$ هم باشد باز درخواست نوشتن رد می‌شه و تراکنش T_i طرد می‌شه

چرا؟ چون یک تراکنش جدید تر نوشتیم و می‌خواهیم دوباره بنویسم، این عمل نوشتن بیهوده است،

ج) به غیر از حالت الف و ب، درخواست نوشتن اجابت می‌شود و قرار می‌دهیم

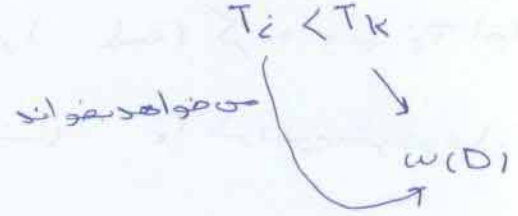
$$TS\text{-}w(D) = \max\{TS\text{-}w(D),\ TS(T_i)\}$$

$TS\text{-}w(D)$: زمان‌مهر تراکنش که آخرین بار او را نوشته

$TS(T_i)$: زمان‌مهر قدیمی

قضیه: پروتکل TO توالی پذیر تعارضی را تضمین می کند. اثبات باخودتان.

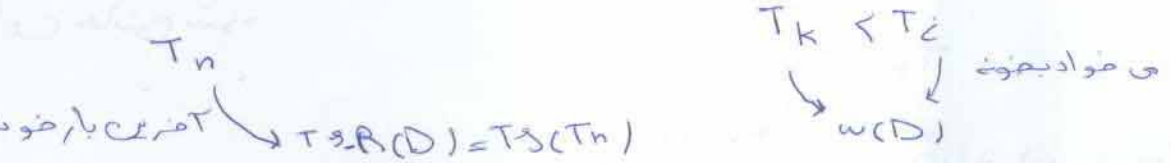
مثال: فرض کنید یک تراکنش T_k داریم که آمده داده D را write کرده و تراکنش کوچکتری که داریم (یعنی قدیمی تر، T_i) یعنی مهر زمانی T_i از مهر زمانی T_k کمتر است یعنی قدیمی تر است، یعنی زودتر شروع شده، می خواد این داده رو بخونه، می گیم اجازه نداری بخونی طردش می کنیم چون ممکن است تراکنش مدون ترک که داده رو نوشته به تشبیت نرسیده باشد T_k، پس این می شه خواندن داده ناجور و T_i طرد می شود



خواندن داده ناجور رخ می دهد و T_i طرد می شود. $TS(T_i) < TS(T_k)$

حالا عکس این:

یک تراکنش قدیمی تر آمده D را نوشته (write) پس چون قدیمی تر است پس احتمالاً به تشبیت هم رسیده و حالا یک تراکنش T_i می خواهد این را بخواند، اجازه خواندن داده می شود منتهی تضمین می کنیم آخرین بار چه کسی D را خونده؟ فرض کنید یک تراکنش T_8 ای بوده، که قبلاً D را خونده، پس $TS\_R(D)$ T_8 باید با مهر زمانی این تراکنش T_n یکی شود $TS(T_n)$، حالا چون اجازه می دهیم این تراکنش بیشتر T_i و این را هم بخواند، پس مهر زمانی D باید عوض بشود و بشود ماکزیمم مهر زمانی قبلی ($TS(T_n)$ و T_8 و تراکنش جدید که اجازه دادیم بخونه.



$$TS\_R(D) = \max\{TS(T_n), TS(T_i)\}$$

جواب:
فرض بر این است که تراکنش‌ها معمولی است، بلندنیستند، معمولاً در حدود ۵ ثانیه، ۱۰ ثانیه هستند و در این حالت معمولاً تموم شدن و به ثبت هم رسیدن

جواب سوال رفیع‌زاده: از کجا بفهمیم تراکنش قبلی تموم شده یا نه؟

در مثال طرد تسلسلی هم اتفاق می‌افتد، تضمین کردن به این معنا نیست که مشکلات رو نداره.

برای عمل نوشتن:

یک تراکنش T_i می خواهد داده‌ی D را بنویسد ولی یک تراکنش جدیدتری این داده را read کرده و به T_i اجازه نمی‌دهد، پس

الف) T_i طرد می‌شود و عمل نوشتن انجام نمی‌شود.

$T_i < T_k$
T_i می خواهد بنویسد
$R(D)$

ب) باز T_i می خواهد داده‌ای را بنویسد که تراکنش جوون‌تر (T_k) آن را نوشته می‌گیم این الان تغییرش داده، تو می‌خوای دوباره تغییر بدی؟ اجازه نوشتن نمی‌دهیم T_i طرد می‌شود، چون نوشتن بیهوده است.

ج) غیر از الف و ب اجازه می‌دهیم، یعنی T_i می خواهد داده را بنویسد $W(D)$ اجازه داده می‌شود. T_k یعنی هم بزرگتر از آخرین کسی که D را خوانده و احتمالاً کار قبلی با D تمام شده و آن را خوانده و حتی از چیزی که D را نوشته هم بزرگتر است، به احتمال زیاد کار این هم تموم شده

$T_k < T_i$
T_k D را خوانده یا نوشته
$W(D)$ اجازه داده می‌شود

فرض کنید T_n آخرین تراکنشی باشد که D را نوشته و زمانمهر نوشتن D می‌شه.

الان یک تراکنش دیگه T_i اومد D

رو نوشت ممکن است $T_n \rightarrow D$

زمانمهر D عوض بشه و

زمانمهر نوشتن D می‌شه max زمانمهر آخرین تراکنشی که D را نوشته یعنی $TS(T_n)$ و حالا یکی دیگه هم نوشت کی بوده؟ $TS(T_i)$

$$TS\text{-}W(D) = \max\{TS(T_n), TS(T_i)\}$$

بین آنهایی بینیم کدوم یک بزرگتر است و آن رو می‌گذاریم زمانمهر نوشتن

تعریف مهر زمانی: زمانمهر مقدار منحصر به فردی است که مدیر هر تراکنش به هر تراکنش می‌دهد، ترکیبی است از زمان شروع تراکنش و ID تراکنش.

مهر زمانی تراکنش اگر کوچکتر باشد معنی اش این است که تراکنش سازودتر شروع شده یا قدیمی تر است.

⇐ مثلاً در مثال سن کسی که متولد سال ۵۰ است با متولد سال ۶۸ مقایسه کنیم

عدد ۵۰ از ۶۸ کوچکتر است. و متولد ۵۰ قدیمی تر از متولد ۶۸ است.

کنترل همروندی با استفاده از چند نسخه سازی: چند نسخه سازی به این معناست که چند نسخه از یک داده ساخته می‌شود، ایجاد می‌شود، و در سیستم پایگاه داده نگهداری می‌شود.

دو نوع چند نسخه سازی داریم:

۱- مبتنی بر مهر زمانی

۲- ~ ~ 2PL

Multi Version time ordering (MVTO) :

۱- چند نسخه سازی مبتنی بر مهر زمانی :

در این روش به مرور زمان از هر داده D ، تعدادی نسخه مثلاً $D_1, D_2, \dots, D_n$

ایجاد می‌شود ، جوری که فرض کنید D_n آخرین نسخه‌ای باشد که ایجاد شده (عمل نوشتن تثبیت‌شده) و تراکنشی بخواهد داده را بخواند آخرین نسخه‌ای را در اختیارش می‌گذاریم ، پس برای خواندن هیچ‌گاه تراکنش طرد تسلسلی نمی‌شود.

حسن : تراکنش خوانده هیچ‌وقت طرد نمی‌شود.

عاه اگر تراکنشی بخواهد داده‌ای را بنویسد ، نسخه جدیدی از آخرین و آن D را ایجاد کرده و در اختیار آن قرار می‌دهیم و خود این جدیده می‌شود آخرین نسخه

حسن : تراکنش‌های خوانده طرد نمی‌شود.

اشکال : مصرف حافظه‌ی آن زیاد است ، هی نسخه ایجاد می‌کنیم .

چند نسخه سازی مبتنی بر 2PL : ترکیب روش چندنسخه‌سازی و 2PL .

در روش قفل‌گذاری معمولی اگر تراکنشی روی داده‌ای قفل نوشتن داشت اجازه نمی‌داد (قفل انحصاری)
که تراکنش دیگر آن را بخواند ، در این روش می‌خواهیم اجازه دهیم .

اگر تراکنشی مثلاً T_1 روی داده D قفل نوشتن دارد و تراکنش T_2 بخواهد
آن را بخواند یک نسخه جدید از روی D ایجاد کرده و در اختیار T_2 قرار می‌دهیم که بخواند
سیستم
منتظر می‌ماند برای تثبیت T_1 بررسی می‌کند در صورتی که تراکنش T_2 به پایان رسیده و خودش
اجازه‌ی تثبیت می‌دهد ، در این روش ممکن است تضعیف همروندی ایجاد شود

روش تایید و تصدیق برای کنترل همروندی:

در این روش به ازای دو قفل نوشتن و خواندن قفل دیگری به نام قفل تایید داریم

این روش ۴ مرحله دارد.

۱- مرحله خواندن: تراکنشها آزادانه می توانند هر داده ای را بخوانند.

۲- مرحله محاسبات: تراکنشها محاسبات خود را انجام داده و در متغیرهای محلی نگه می دارند.

۳- مرحله دریافت قفل تایید: قبل از نوشتن و به تثبیت رسیدن، تراکنش باید مهر تایید بگیرد. سیستم بررسی می کند اگر این تراکنش نوشتنش با تراکنشهای دیگه "تثبیت‌شده" برخورد نداره، مهر تایید صادر می کند

۴- مرحله نوشتن و تثبیت: تراکنشی که مهر تایید گرفته می تواند داده های خود را بنویسد و به مرحله تثبیت برسد.

برای امتحان:

جزوه

+

≡ فصل اول از کتاب دوم راکلو همی

+

نرمالسازی و وابستگی تابعی از کتاب اول راکلو همی

نقطه درس مونده { جامعیت، امنیت و ـــ .

سوال: در 3nf ارتباط یک به یک چی بود که ادغام می کردیم؟ مثلاً اطلاعات آقای فلانی اطلاعاتش در یک جدول هست «اسم و مشخصات...»، جدول دیگه هم شهر محل تولدش، ارتباط این دو جدول یک به یک است، چون اون آقا دو جا متولد نشده

نمی‌شه هم قم متولدشده باشه هم کاشان ، چون فقط قم متولدشده می‌توانم
اون جدول رو ادغام کنم .