فصل ۱۵ : کنترل همروندی

اسلاید ۳

پروتکل های مبتنی بر قفل گذاری:

قفل یک مکانیزم کنترل همروندی در دسترسی به مقادیر داده می باشد

مقادیر داده می توانند به دو صورت قفل گذاری گردند :

1 - قفل انحصاری : که داده مورد نظر برای خواندن و نوشتن با قفل انحصاری که توسط دستور Lock-x درخواست می شود محافظت خواهد شد.

2 - قفل اشتراکی : که داده مورد نظر توسط دستور Lock-s تنها برای خواندن قفل گذاری می گردد.

درخواست های قفل گذاری مدیریت همروندی را فراهم می نمایند . یک تراکنش زمانی می تواند پردازش شود که درخواست قفل گذاری آن تایید شده باشد.

اسلاید ۴

ماتریس سازگاری قفل گذاری :

	S	X
S	true	false
X	false	false

یک تراکنش زمانی می تواند یک قفل بر روی یک مورد ایجاد نماید که درخواست قفل گذاری او با قفل های موجود تراکنش های دیگر بر روی آن مقدار داده سازگار باشد.

قفل اشتراکی بر روی یک مقدار داده می تواند توسط هر تعداد تراکنش درخواست کننده قرار داده شود ، اما ، اگر یک تراکنش قفل انحصاری بر روی یک مقدار داده ایجاد نماید ، هیچ تراکنش دیگری نمی تواند بر روی آن مقدار داده قفل گذاری نماید.

اگر یک درخواست قفل گذاری نتواند انجام شود ، تراکنش درخواست کننده قفل باید منتظر بماند تا تمامی قفل های ناسازگار دیگر تراکنش ها بر روی آن مقدار داده آزاد گردند و پس از آن می تواند قفل خود را ایجاد نماید.

اسلاید ۵

مثالی برای قفل گذاری یک تراکنش :

```
T2 :    lock-S(A);
        Read(A);
        Unlock(A);
```

Lock-S(B);

Read(B);

Unlock(B);

Display(A+B);

قفل گذاری بالا نمی تواند قابلیت سریالی بودن (serializability) را تضمین نماید . اگر در بین خواندن مقدار A و B مقدار یکی از آنها بروز رسانی شود ، در نتیجه جواب جمع غلط خواهد بود.

یک پروتکل قفل گذاری مجموعه ای از قواعدی است که تمامی تراکنش ها برای درخواست و آزاد سازی قفل ها از آن پیروی می نمایند . پروتکل های قفل گذاری مجموعه زمانبندی های ممکن را محدود می نمایند.

اسلاید ۶

تله های(چالش ها) موجود در پروتکل های مبتنی بر قفل گذاری:

به زمانبندی کوتاه زیر دقت نمایید :

T3	T4
Lock-X(B)	
Read(B)	
B:=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

هیچ کدام از تراکنش های T3 و T4 نمی توانند اجراء شوند. اجرای دستور Lock-S(B) نمی تواند اجرا شود بخاطر اینکه T4 منتظر است تا T3 قفل خود را از B رها کند. همچنین اجرای Lock-X(A) بخاطر آنکه T3 منتظر آن است که T4 قفل خود را از روی A بردارد . این شرایط را بن بست می نامند.

برای برطرف نمودن بن بست باید یکی از تراکنش های T4 یا T3 دستورات خود را عقبگرد (rolled back) نموده و قفل خود را آزاد نماید.

اسلاید ۷

امکان بوجود آمدن بن بست در اغلب پروتکل های قفل گذاری وجود دارد. همچنین اگر مدیریت کنترل همروندی بطور نامناسب طراحی شده باشد احتمال وقوع پدیده گرسنگی وجود خواهد داشت. برای مثال زمانی که یک تراکنش به قفل انحصاری بر روی یک مقدار داده نیاز دارد در حالی که دنباله ای از تراکنش های دیگر بر روی آن مقدار داده قفل اشتراکی نهاده باشند.

تراکنش های مشابه در هنگام وقوع بن بست مکرراً rolled back خواهند شد.

مدیریت همروندی می تواند طوری طراحی شود که از گرسنگی جلوگیری نماید.

اسلاید ۸

پروتکل قفل گذاری 2 مرحله ای (2PLP)

این پروتکل زمانبندی را تا سطح Conflict serializable تضمین می نماید.

فاز اول : فاز رشد (فاز قفل گذاری)

- تراکنش ها اقدام به قفل گذاری می نمایند
- تراکنش ها قفل ها را در این فاز آزاد نمی نمایند

فاز دوم : فاز قفل برداری

- تراکنش ها اقدام به قفل برداری می نمایند
- تراکنش ها در این مرحله قفل گذاری نمی نمایند

این پروتکل serializability را تضمین می کند و ثابت کرده که تراکنش ها می توانند در مرحله قفل گذاری خودشان سریالی باشند. همان نقطه ای که یک تراکنش آخرین قفل خود را قرار می دهد.

اسلاید ۹

پروتکل قفل 2 مرحله ای اطمینانی از این که روند پردازش به بن بست نرسد نمی دهد.

همچنین امکان بوجود آمدن rolled back های آبشاری هم در آن وجود دارد.

برای دوری از شرایط مذکور از پروتکل اصلاح شده ای به نام قفل 2 مرحله ای شدید استفاده می شود که در آن یک تراکنش باید تمامی قفل های انحصاری خود را تا زمان commit یا abort شدن نگه دارد.

قفل 2 مرحله ای شدید حتی می تواند شدید تر هم شود (R2PL) ، که تمامی قفل ها را تا زمان commit یا abort شدن نگه می دارد .

در این پروتکل تراکنش ها می توانند در طول انجام کار تا زمانی که commit نمایند سریالی باشند.

اسلاید ۱۰

حالتهایی وجود دارد که زمانبندی Conflict Serializable با استفاده از قفل گذاری 2 مرحله ای بدست نمی آید.

در مفهوم زیر برای دستیابی به Conflict Serializability قفل 2 مرحله ای نیاز است ، هرچند اطلاعات اضافی وجود نداشته باشد :

تراکنش Ti که از قفل 2 مرحله ای پیروی نمیکند مفروض است و ما تراکنش Tj را پیدا می کنیم که از قفل 2 مرحله ای استفاده می کند ، و زمانبندی که برای این دو وجود دارد Conflict Serializable نیست.

اسلاید ۱۱

تبدیلات قفل:

قفل 2 مرحله ای با تبدیلات قفل:

- ❖ فاز اول :
  - می توان یک قفل اشتراکی بر روی یک مقدار قرار داد
  - می توان یک قفل انحصاری بر روی یک مقدار قرار داد
  - می توان یک قفل اشتراکی را به انحصاری تبدیل نمود (upgrade)
- ❖ فاز دوم:
  - می توان یک قفل اشتراکی را برداشت
  - می توان یک قفل انحصاری را برداشت
  - می توان یک قفل انحصاری را به اشتراکی تبدیل نمود (downgrade)

این پروتکل serializability را بیمه می کند. اما هنوز به برنامه نویس جهت وارد نمودن دستورات مختلف قفل گذاری وابسته است.

اسلاید ۱۲

انتساب خودکار قفل ها:

- تراکنش Ti استاندارد دستورات خواندن و نوشتن را بدون قفل گذاری مستقیم رعایت می کند.
- عملیات خواندن D بصورت زیر پردازش شده :

if Ti has a lock on D

then

Read(D)

else begin

If necessary wait until no other transaction has a **lock-X** on D

grant Ti a **lock-S** on D;

Read(D)

end

اگر Ti بر روی داده D قفل دارد آن وقت عملیات خواندن انجام می شود

در غیر اینصورت

در صورت لزوم تا زمانی که تراکنش دیگری قفل انحصاری بر روی آن مقدار نداشته باشد صبر می کند

قفل اشتراکی خود را بر روی مقدار D ایجاد می کند

عملیات خواندن انجام می شود و پایان

اسلاید ۱۳ ادامه اسلاید قبل

نوشتن D بصورت زیر پردازش شده :

if*Ti* has a **lock-X** on *D*

then

write(*D*)

else begin

if necessary wait until no other trans. has any lock on *D*,

if*Ti* has a **lock-S** on *D*

then

upgradelock on *D* to **lock-X**

else

grant*Ti* a **lock-X** on *D*

write(*D*)

end;

اگر Ti بر روی مقدار D قفل انحصاری دارد آنگاه نوشتن انجام می شود

در غیر اینصورت

در صورت لزوم تا زمانی که تراکنش دیگری قفل بر روی D نداشته باشد صبر می کند

اگر Ti قفل اشتراکی بر روی D داشته باشد آن وقت

آن قفل را بر روی D به قفل انحصاری upgrade می کند

در غیر اینصورت قفل انحصاری خود را بر روی Dقرار می دهد

مقدار D را می نویسد و پایان

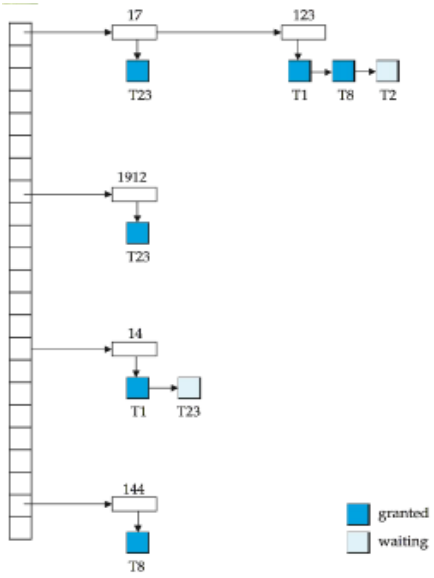
تمامی قفل ها پس از Commit یا abort شدن آزاد می شوند.

اسلاید۱۴

اجراء قفل گذاری :

- یک مدیریت قفل گذاری می تواند بطور جداگانه برای تراکنش هایی که درخواست پردازش قفل گذاری و قفل برداری ارسال می کنند اجراء شود.
- یک مدیر قفل گذاری به یک درخواست قفل گذاری توسط ارسال تایید قفل گذاری (و یا پیام پرسش برای Roll back تراکنش در مواقع وقوع بن بست) پاسخ می دهد.
- تراکنش های درخواست کننده تا زمانی که پاسخ آنها صادر شود منتظر می مانند.
- مدیر قفل گذاری جهت نگهداری قفل های اختصاص یافته و درخواست های در حال انتظار از یک ساختار داده به نام جدول قفل استفاده می نماید.
- Lock table معمولا محلی از حافظه است (Memory hash table) که در آن بر روی نام مقادیر داده ای که قفل گذاری شده اندانديس گذاری شده است.

اسلاید۱۵



- مستطیل های تو پر نشانگر قفل های اختصاص یافته می باشند و مستطیل های خالی نشانگر درخواست های در حال انتظار.
- جدول قفل همچنین نوع قفل های تخصیص یافته یا درخواست شده را ثبت می نماید.
- درخواست جدید به انتهای صف درخواست های مقدار داده مورد نظر اضافه می شود و اگر قفل درخواست شده با تمامی قفل های قبلی سازگار باشد به درخواست کننده اختصاص داده می شود.
- درخواست قفل گشایی با حذف قفل پاسخ داده می شود و پس از آن باقی درخواست های قفل گذاری چک می شوند تا در صورت امکان اختصاص یابند.
- اگر تراکنشی abort شود ، تمامی قفل های مربوط به آن و درخواست های در حال انتظار آن تراکنش حذف خواهند شد. مدیر قفل گذاری ممکن است یک لیست از قفل های هر تراکنش برای پیاده سازی مؤثر این موضوع تهیه کند.

اسلاید۱۶

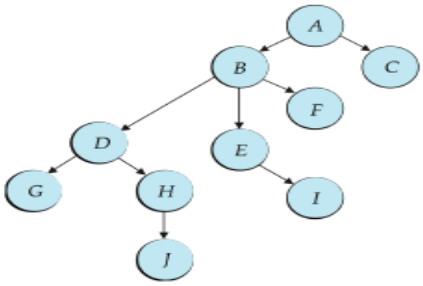
پروتکل های مبتنی بر گراف :

- پروتکل های مبتنی بر گراف جایگزینی برای قفل 2 مرحله ای می باشند.
- یک رابطه ترتیب جزئی بر روی مجموعه D={d1,d2...dn} و بر روی همه مقادیر داده وضع می شود:
  - اگر di--->dj : پس هر تراکنش که می خواهد به di و dj دسترسی داشته باشد باید قبل از دسترسی به dj به di دسترسی داشته باشد.
  - مجموعه D به عنوان یک گراف مستقیم بدون دور شناخته می شود که گراف پایگاه داده نامیده می شود.

- پروتکل درخت یک نمونه ساده از پروتکل گراف است.

اسلاید۱۷

پروتکل درخت :



۱. تنها قفل انحصاری مجاز می باشد
۲. اولین قفل توسط Ti بر روی هرکدام از مقادیر داده می تواند قرار گیرد. پس از آن تنها زمانی می توان مقدار داده Q را قفل گذاری نمود که قبلاً والد آن را خود Ti قفل گذاری کرده باشد.
۳. یک مقدار داده در هر زمانی می تواند قفل گشایی گردد.
۴. یک مقدار داده که توسطTi قفل گذاری شده و سپس قفل آن آزاد شده است نمی تواند مجدداً توسط Ti قفل گذاری شود.

اسلاید۱۸

پروتکل های مبتنی بر گراف (ادامه مطلب) :

- پروتکل درخت سطح conflict Serializability را تضمین می کند و همچنین دچار بن بست نمی شود.
- در پروتکل قفل گذاری درخت به نسبت پروتکل قفل 2 مرحله ای عملیات قفل گشایی می تواند زودتر صورت پذیرد.
  - زمان های انتظار کوتاه تر هستند و همروندی بیشتر
  - این پروتکل دچار بن بست نشده و نیاز به rollback ندارد
- اشکال ها :
  - پروتکل قابل بازیابی بودن یا عقبگرد های آبشاری نداشتن را تضمین نمی کند :
    - ✓ نیاز به نشان دادن وابستگی های Commit برای اطمینان از قابلیت بازیابی
  - ممکن است تراکنشها نیاز به قفل نمودن مقادیر داده ای داشته باشند که به آنها دسترسی ندارند :
    - ✓ سربار قفل گذاری زیاد و زمان انتظار اضافی بوجود می آید
    - ✓ امکان افت همروندی وجود دارد
- زمانبندی هایی که در قفل 2 مرحله ای امکان پذیر نیستند در پروتکل درخت امکان پذیر هستند و بلعکس.

اسلاید۱۹

اداره کردن بن بست :

تراکنش های زیر را در نظر بگیرید:

T1:	write (X)	T2:	write (Y)
	Write (Y)		write (X)

زمانبندی دارای بن بست :

T_1	T_2
lock-X on A write (A)	
	lock-X on B write (B) wait for **lock-X** on A
wait for **lock-X** on B	

اسلاید۲۰

اداره کردن بن بست :

- زمانی که یک مجموعه از تراکنش ها را داشته باشیم که هر تراکنش در آن مجموعه بخاطر تراکنش دیگری در آن مجموعه در انتظار باشد سیستم به بن بست خواهد رسید.
- **جلوگیری از بن بست** : پروتکل های جلوگیری از بن بست به ما اطمینان می دهند که سیستم هیچوقت وارد وضعیت بن بست نخواهد شد.(برای این منظور)
  - هر تراکنش نیاز دارد تمامی داده های مورد نیاز خود را قبل از اجراء قفل گذاری نماید (اعلان قبلی)
  - یک رابطه ترتیب جزئی برای تمام مقادیر داده وجود دارد و لازم است که یک تراکنش بتواند بر روی مقادیر داده قفل گذاری کند ، تنها در جهت مشخص شده توسط رابطه ترتیب جزئی

اسلاید ۲۱

استراتژی های پیشگیری از بن بست :

- در طرح های زیر تراکنش جهت جلوگیری از بن بست تنها از مهر زمانی استفاده می کند.
- طرح انتظار- مرگ (پس نگرفتنی) :
  - تراکنش قدیمی تر ممکن است منتظر بماند تا یک تراکنش جوانتر مقدار داده ای را رها کند. اماتراکنش های جوان تر هیچوقت منتظر تراکنش قدیمی نمی مانند و rolleback خواهند شد.
  - یک تراکنش ممکن است قبل از اینکه به داده ی مورد نیاز خود دست یابد چندین بار بمیرد.
- طرح زخم – انتظار (پس گرفتنی)
  - تراکنش های قدیمی تر بجای صبر کردن برای تراکنش های جوان تر آنها را زخمی می کنند. تراکنش های جوانتر ممکن است برای قدیمی تر ها صبر نمایند.
  - احتمال rollback در مدل زخم و انتظار کمتر است.

اسلاید ۲۲

پیشگیری از بن بست (ادامه)

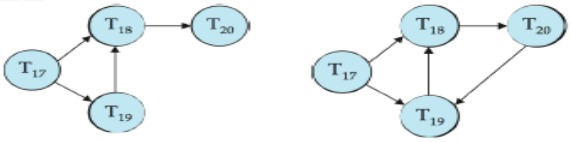
- در هر دو روش قبلی (wait-die and wound-wait) تراکنش هایی که عقبگرد (rollback) می نمایند مجدداً با مهر زمانی اصلی خود آغاز به کار می نمایند. در نتیجه تراکنش های قدیمی تر نسبت به تراکنش های جوانتر دارای اولویت هستند و از گرسنگی دوری می شود.
- طرح های مبتنی بر وقفه :
  - یک تراکنش تنها در یک بازه زمانی مشخص در انتظار یک قفل خواهد ماند. پس از آن زمان انتظار خارج شده و تراکنش عقبگرد (rollback) خواهد نمود.
  - در نتیجه بن بست امکان پذیر نخواهد بود
  - برای اجرا ساده است اما پدیده گرسنگی در آن امکان پذیر است .همچنین تعیین مقدار بازه زمانی مناسب برای آن سخت است.

اسلاید ۲۳

تشخیص بن بست :

- بن بست می تواند توسط گراف انتظار توضیح داده شود که متشکل از زوج G=(V,E) است :
  - V مجموعه ای از رئوس است (تمام تراکنش های موجود در سیستم)
  - E مجموعه ای از یال ها است و هر عنصر یک زوج مرتب Tj <--- Ti
- اگر Tj <--- Ti در E وجود داشته باشد ، پس از Ti به Tj یک یال مستقیم وجود دارد که نشان می دهد Ti منتظر است تا Tj یک مقدار داده را رها نماید.
- زمانی که Ti یک مقدار داده را درخواست می کند که در حال حاضر در اختیار Tj است ، آنگاه در گراف انتظار یک یال از Ti به Tj رسم می شود. این یال زمانی حذف خواهد شد که Tj مقدار داده ای را که مورد نیاز Ti است را در اختیار نداشته باشد.
- تنها در زمانی که گراف انتظار دارای دور باشد سیستم دارای بن بست می باشد. برای پیدا کردن دور ها باید الگوریتم تشخیص بن بست بصورت دوره ای فراخوانی گردد.

اسلاید ۲۴



گراف انتظار بدون دور گراف انتظار دارای دور

اسلاید ۲۵

بازیابی بن بست :

- زمانی که بن بست تشخیص داده شد :
  - جهت رفع بن بست ، برخی از تراکنش ها باید عقبگرد نمایند (قربانی شوند) . تراکنشی به عنوان قربانی انتخاب می گردد که کمترین هزینه را داشته باشد.

- عقبگرد(rollback) – تعیین آنکه تراکنش تا کجا باید عقبگرد نماید
  - ✓ عقبگرد کامل : تراکنش ساقط شده و مجدداً از ابتدا شروع می گردد.
  - ✓ مؤثر تر از آن عقبگرد تراکنش تنها تا جایی است که برای رفع بن بست لازم می باشد.
- اگر اغلب تراکنش مشابهی جهت قربانی شدن انتخاب گردد احتمال پدیده گرسنگی وجود دارد . تعداد عقبگرد ها در فاکتور هزینه برای جلوگیری از گرسنگی در نظر گرفته می شود.

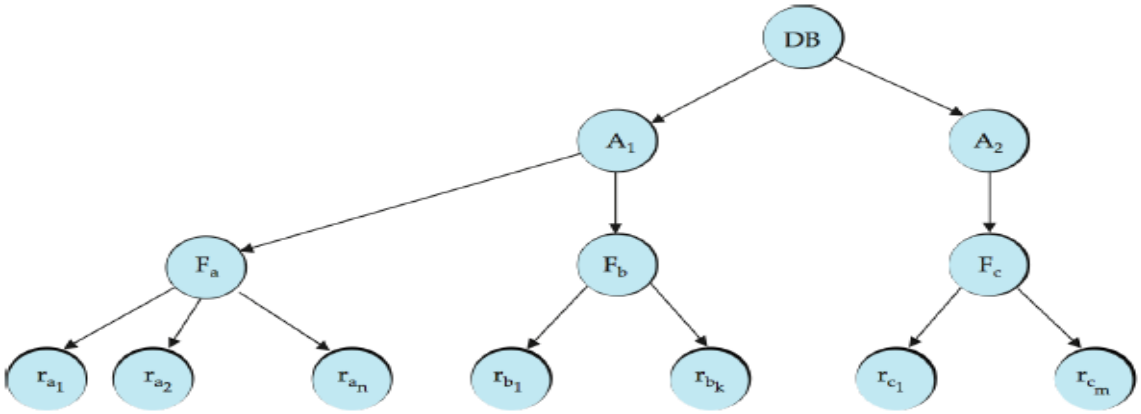
اسلاید ۲۶

قفل گذاری در اندازه های مختلف (دانه بندی) :

- مقادیر داده در اندازه های مختلف با تعریف سلسله مراتب دانه بندی بطوری که دانه های کوچک و بزرگ با هم ترکیب شوند تبیین می گردند.
- می توان بصورت گرافیکی نمایش داد ، مانند یک درخت (اما با پروتکل قفل گذاری درخت اشتباه نگیرید)
- هنگامی که یک تراکنش بر روی یک گره از درخت بطور صریح قفل می گذارد فرزندان آن گره مشابه همان قفل ، بصورت ضمنی قفل می شوند.
- دانه بندی قفل گذاری (سطحی از درخت که در آن قفل گذاری می شود) :
  - دانه بندی ریز (پایین در درخت) : همروندی بالا ، سربار قفل گذاری زیاد
  - دانه بندی درشت (بالا در درخت) : سرباز قفل گذاری کم ، همروندی کمتر

اسلاید ۲۷

مثالی از سلسله مراتب دانه بندی :



سطوح مختلف ، شروع از سطح دانه درشت (سطح بالا) :

- پایگاه داده
- منطقه
- فایل
- رکورد

اسلاید ۲۸

حالتهای مختلف قفل هدفمند :

- علاوه بر قفل هایS و X سه نوع قفل دیگر با دانه بندی های متفاوت وجود دارد :
  - به قصد اشتراکی (IS) : نشان میدهد که جایی در پایین درخت بطور صریح با قفل اشتراکی قفل گذاری شده است.
  - به قصد انحصاری(IX) : نشان می دهد که جایی در پایین درخت بطور صریح قفل اشتراکی یا انحصاری قرار میگیرد.
  - اشتراکی و به قصد انحصاری (SIX) : زیر درخت منشعب شده از آن گره بصورت صریح با قفل اشتراکی قفل گذاری شده و جایی در سطح پایین تر درخت قفل انحصاری وجود دارد.
- این شیوه از قفل گذاری (intention locks) این امکان را میدهد که ریشه را در حالت اشتراکی و یا انحصاری قفل گذاری نماییم بدون آنکه تمام گره های فرزند آن را بررسی نماییم.

اسلاید ۲۹

ماتریس سازگاری برای همه حالات Intention Lock (قفل هدفمند)

- ماتریس سازگاری قفل ها برای تمامی حالات قفل گذاری

	IS	IX	S	SIX	X
IS	true	true	true	true	false
IX	true	true	false	false	false
S	true	false	true	false	false
SIX	true	false	false	false	false
X	false	false	false	false	false

اسلاید ۳۰

طرح قفل گذاری با دانه بندی چندگانه:

- تراکنش Ti می تواند با توجه به قوانین زیر گرهی مانند Q را قفل گذاری نماید:
  - باید قفل گذاری با توجه به جدول سازگاری قفل گذاری باشد
  - ابتدا ریشه درخت باید قفل گذاری شود که این قفل گذاری می تواند در هر حالتی باشد
  - گره Q تنها زمانی می تواند توسط Ti بصورت S یا IS قفل گذاری شود که قبلاً والد Q توسط Ti بصورت IS یا IX قفل گذاری شده باشد.
  - گره Q می تواند توسط Ti در حالت های X ، SIX یا IX قفل گذاری شود ، اگر قبلا والد گره Q توسط Ti در حالت های IX یا SIX قفل گذاری شده باشد.

- Ti تنها زمانی می تواند گره Q را قفل گذاری نماید که قبل از آن گرهی را قفل گشایی ننموده باشد (مشابه قفل 2 مرحله ای)
- Ti تنها زمانی می تواند گره Q را قفل گشایی نماید که در آن لحظه هیچ یک از فرزندان Q توسط Ti قفل نباشند.
- مشاهده می نمایید که قفل گذاری از ریشه به سمت برگ انجام می شود(بالا به پایین) و قفل گشایی از برگ به سمت ریشه می باشد (پایین به بالا)

اسلاید ۳۱

پروتکل های مبتنی بر مهر زمانی :

- هر تراکنش که وارد سیستم می شود ، برای آن یک مهر زمانی صادر می گردد. اگر تراکنش قدیمی Ti مهر زمانی TS(Ti) را داشته باشد ، و به تراکنش جدید Tj مهر زمانی TS(Tj) اختصاص داده شود آنگاه داریم : TS(Ti)<TS(Tj)
- پروتکل ، اجرای همروند را به گونه ای مدیریت می نماید که ترتیب Serializability را برچسب های زمانی تعیین می کنند.
- برای دستیابی بدین منظور ، دو مقدار مهر زمانی برای هر داده Q توسط پروتکل نگهداری می شود :
  - W-timestamp(Q) : بزرگترین مهر زمانی مربوط به هر تراکنشی که نوشتن Q را بطور کامل اجراء نموده.
  - R-timestamp(Q) : بزرگترین مهر زمانی مربوط به هر تراکنشی که خواندن Q را بطور کامل اجرا نموده .

اسلاید ۳۲

پروتکل های مبتنی بر مهر زمانی : (ادامه)

- پروتکل ترتیب برچسب زمانی تضمین می کند که هر دستور تداخلی خواندن یا نوشتن در ترتیب اجرای مهر زمانی می باشد.
- تراکنش Ti که میخواهد داده Q را بخواند در نظر بگیرید :

۱. اگر TS(*Ti*) ≤ **W**-timestamp(*Q*) باشد ، آنگاه تراکنش Ti نیاز به خواندن مقدار Q ای خواهد داشت که مقدار آن باز نویسی شده است (overwrite)
   - ✓ در نتیجه عملیات خواندن مردود شده و تراکنش Ti عقبگرد(rolled back) می کند
۲. اگر TS(*Ti*)≥ **W**-timestamp(*Q*) آنگاه عملیات خواندن اجراء می شود و بیشترین مقدار مهر زمانی خواندن Q را به عنوان مهر زمانی خواندن آن تنظیم می کند ، TS(*Ti*)

اسلاید ۳۳

پروتکل های مبتنی بر مهر زمانی : (ادامه)

- تراکنش Ti که میخواهد داده Q را بنویسد در نظر بگیرید :

۱. اگر TS(*Ti*) < R-timestamp(*Q*) آنگاه مقدار Q که Ti آن را تولید می نماید قبل از آن مورد نیاز بوده است و سیستم عهده دار این موضوع است که آن مقدار هرگز تولید نخواهد شد :
   - ✓ در نتیجه عملیات نوشتن مردود شده و تراکنش Ti عقبگرد (rolled back) می کند
۲. اگر TS(*Ti*) < W-timestamp(*Q*) آنگاه Ti اقدام به نوشتن یک مقدار فاقد ارزش نموده است

✓ در نتیجه عملیات نوشتن مردود شده و تراکنش Ti عقبگرد (rolled back) می کند

۳. در غیر اینصورت عملیات نوشتن انجام شده و مهر زمانیTi به عنوان مهر زمانی نوشتن Q تنظیم می شود.

اسلاید ۳۴

مثال استفاده از پروتکل :

یک ترتیب زمانبندی برای مقادیر داده چندین تراکنش با مهر های زمانی 1 ، 2 ، 3 ، 4 ، 5

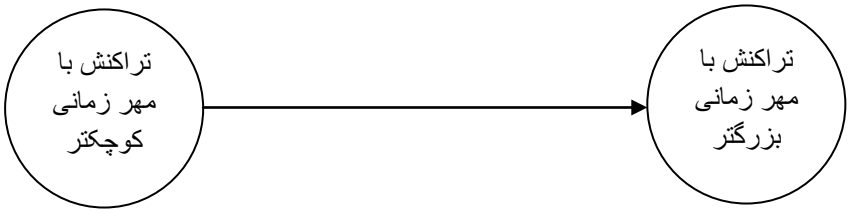
T_1	T_2	T_3	T_4	T_5
				read (X)
	read (Y)			
read (Y)				
		write (Y)		
		write (Z)		
				read (Z)
	read (Z)			
	abort			
read (X)				
			read (W)	
		write (W)		
		abort		
				write (Y)
				write (Z)

جهت سریالی بودن ، از چپ به راست (T5 , T2 , T1 , T3 , T4)

اسلاید ۳۵

صحت پروتکل ترتیب مهر زمانی :

- پروتکل ترتیب مهر زمانی serializability را تضمین میکند تا زمانی که تمام یال های گراف تقدم به فرم زیر باشند :



بنابراین در گراف تقدم دور نخواهیم داشت

- پروتکل ترتیب مهر زمانی به دور بودن از بن بست را بیمه می کند و هیچ تراکنشی دائما منتظر نمی ماند.

- اما زمانبندی ممکن است cascade free نباشد و همچنین ممکن است قابل بازیابی نباشد.

اسلاید ۳۶

قابل بازیابی بودن و آبشاری نبودن :

- مسائل مربوط به پروتکل ترتیب مهر زمانی :
  - فرض کنید که Ti سقط شده ، اما Tj مقداری را خوانده که توسط Ti نوشته شده است
  - آنگاه Tj می بایست سقط شود ، اگر Tj اجازه داشت که زودتر commit نماید ، این زمانبندی قابل بازیابی نیست.
  - علاوه بر این هر تراکنشی که مقدار داده ای را که Tj نوشته است خوانده باید سقط شود
  - این می تواند باعث عقبگرد آبشاری گردد (cascading rollback) که آن زنجیره ای از عقبگرد ها می باشد.
- راه حل اول :
  - تراکنش باید طوری سازماندهی شود که در انتهای کار خود عملیات نوشتن را انجام دهد.
  - تمامی نوشتن های یک تراکنش در عملیاتی اتمیک انجام شود و در حین نوشتن هیچ تراکنشی اجراء نشود
  - یک تراکنش که سقط می شود با مهر زمانی جدیدی مجدداً شروع به کار نماید.
- راه حل دوم : ایجاد محدودیت توسط قفل گذاری : منتظر commit نمودن مقدار داده می مانیم قبل از آنکه اقدام به خواندن آن نماییم.
- راه حل سوم : استفاده از وابستگی در commit کردن برای اطمینان از قابل بازیابی بودن (اگر Ti خواست commit کند و از Q استفاده کرده که Tj از آن استفاده نموده است : اگر Tj کامیت کرده باشد آنگه Ti می تواند commit نماید)

اسلاید ۳۷

قانون نوشتن توماس :

- نسخه تغییر یافته ای از پروتکل ترتیب مهر زمانی است که در آن عملیات نوشتن نا معتبر ممکن است تحت شرایطی خاص نادیده گرفته شود.
- زمانی که Ti می خواهد مقدار داده Q را بنویسد ، اگر TS(*Ti*) <W-timestamp(*Q*) باشد آنگاه Ti اقدام به نوشتن مقدار نا معتبر Q می نماید.
  - بجای آنکه Ti مثل پروتکل ترتیب مهر زمانی عقبگرد نماید ، از این عمل نوشتن صرف نظر می گردد.
- در مواقع دیگر این پروتکل نیز همانند پروتکل ترتیب مهر زمانی عمل می نماید.
- قانون نوشتن توماس پتانسیل همروندی بیشتری را مقدور می سازد .
  - برخی زمانبندی های view-serializable را مقدور می سازد که آنها conflict-serializable نیستند.

اسلاید ۳۸

View Serializability

- S و S' دو زمانبندی با تراکنش های مشابه می باشند . S و S' معادل دیداری (**view equivalent**) می باشند اگر برای هر مقدار داده Q دارای سه شرط زیر باشند :

۱. اگر در زمانبندی S تراکنش Ti یک مقدار اولیه از Q را بخواند آنگاه در زمانبندی S' هم تراکنش Ti باید همان مقدار اولیه را از Q بخواند.

۲. اگر در زمانبندی S تراکنش Ti دستور خواندن Q را اجرا کند و مقدار آن توسط تراکنش Tj تولید شده باشد ، آنگاه در زمانبندی S' هم تراکنش Ti باید مقدار Q را بخواند که توسط عملیات یکسان نوشتن توسط Tj انجام شده باشد .

۳. تراکنشی که آخرین عملیات نوشتن در زمانبندی S را انجام می دهد همچنین باید آخرین عملیات نوشتن در زمانبندی S' را انجام دهد .

همانطور که مشاهده می کنید هم ارزی دیداری (**view equivalent**) تنها مبتنی بر خواندن و نوشتن ها می باشد.

اسلاید ۳۹

View Serializability (ادامه)

- زمانبندی S ، View serializable می باشد اگر معادل دیداری (**view equivalent**) زمانبندی سریال باشد .
- هر زمانبندی conflict serializable ، view serializable نیز می باشد.
- زمانبندی زیر که یک زمانبندی view serializable است ، conflict serializable نمی باشد .

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- چه زمانبندی سریالی معادل زمانبندی فوق می باشد؟
- هر زمانبندی view serializable که نوشتن کورکورانه داشته باشد conflict serializable نخواهد بود.

اسلاید ۴۰

نکاتی دیگر درباره serializability :

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

- در زمانبندی زیر نتیجه ای مشابه نتیجه زمانبندی سریال <T1 , T5> تولید شده ، با این وجود این زمانبندی نه conflict serializable است و نه view equivalent .
- تجزیه و تحلیل چنین هم ارزی هایی نیازمند تجزیه و تحلیل عملیات هایی غیر از خواندن و نوشتن است.

اسلاید ۴۱

تست نمودن view serializability :

- تست گراف تقدم conflict serializability نمیتواند بطور مستقیم برای تست view serializability مورد استفاده قرار گیرد.
  - علاوه بر آن برای آزمون view serializability با اندازه های گراف تقدم هزینه نمایی خواهیم داشت.
- مسئله بررسی اینکه آیا یک زمانبندی view serializable است یا نه جزو مسائل NP-complete می باشد.
  - بدین ترتیب احتمال وجود داشتن یک الگوریتم کارآمد بعید به نظر می رسد.
- هر چند الگوریتم های عملی که برخی شرایط کافی برای view serializability را بررسی می کنند هنوز هم می توانند مورد استفاده قرار گیرند.

اسلاید ۴۲

پروتکل مبتنی بر اعتبار :

- اجرای تراکنش Ti در سه فاز به پایان می رسد:
  - فاز خواندن و اجراء : در این فاز تراکنش Ti تنها بر روی حافظه موقت می نویسد.
  - فاز اعتبار سنجی : تراکنش Ti یک آزمون اعتبار را می گذراند برای تعیین آنکه آیا متغیر های محلی می توانند بدون نقص در سریالی بودن نوشته شوند
  - فاز نوشتن : اگر تراکنش Ti معتبر باشد بروز رسانی ها بر روی پایگاه داده انجام می شوند و در غیر اینصورت Ti عقبگرد (rollback) خواهد نمود.
- هر سه فاز در اجرای همروند تراکنشها می توانند با هم اجرائ شوند اما هر تراکنش می بایست از این سه مرحله به ترتیب عبور نماید.
  - فرض کنید برای سادگی کار فاز اعتبار سنجی و نوشتن همزمان با هم انجام می شود ، بصورت اتمیک و سریالی
    - ✓ و این یعنی که در یک لحظه تنها یک تراکنش اعتبار سنجی و نوشتن خود را اجراء می نماید.
- همچنین به آن کنترل همروندی خوش بینانه گفته می شود ، چرا که امیدوار است که تراکنش ها بطور کامل اجراء شده و فاز اعتبار سنجی را به خوبی طی نمایند .

اسلاید ۴۳

پروتکل مبتنی بر اعتبار : (ادامه)

- هر تراکنش Ti سه مهر زمانی دارد :
  - Start(Ti) : زمانی که Ti شروع به اجرا نموده
  - Validation(Ti) : زمانی که تراکنش Ti وارد فاز اعتبار سنجی خود شده
  - Finish(Ti) : زمانی که تراکنش Ti فاز نوشتن خود را به پایان رسانده
- ترتیب serializability برای افزایش همروندی توسط مهر زمانی فاز اعتبار سنجی تعیین می شود

- بنابراین TS(Ti) مقدار اعتبار سنجی تراکنش(Validation(Ti)) را دریافت می کند .

- این پروتکل سودمند بوده و اگر میزان تداخل پایین باشد میزان بیشتری از همروندی را فراهم می سازد.
  - چراکه میزان سریالی بودن و ترتیب از قبل تعیین نشده است و همچنین
  - تعداد نسبتاً کمی از تراکنش ها عقبگرد خواهند داشت.

اسلاید ۴۴

آزمون اعتبار برای تراکنش Tj

- به ازای هر Ti داریم TS (Ti) < TS (Tj) ، اگرچه یکی از آنها شرایط زیر را داشته باشد :
  - finish(Ti) < start(Tj)
  - start(Tj) < finish(Ti) < validation(Tj) و مجموعه مقادیر نوشته شده توسط Ti با مجموعه مقادیر داده ای که Tj خوانده تداخل ندارند.

در اینصورت اعتبار سنجی کامل شده و Tj می تواند commit نماید ، در غیر اینصورت اعتبار از دست رفته و Tj سقط می شود.

- اثبات : در صورتی که شرط اول برقرار باشد اجرای همپوشانی(overlapped execution) وجود نداشته ، یا شرط دوم برقرار است که :
  - نوشتن Tj تحت تأثیر خواندن Ti قرار نمی گیرد چرا که این اتفاق پس از آنکه Tiبه پایان رسیده صورت پذیرفته است.
  - نوشتن Ti تحت تأثیر خواندن Tj قرار نمی گیرد چرا که هیچکدام از مقادیری که توسط Ti نوشته شده توسط Tj خوانده نشده.

اسلاید ۴۵

زمانبندی تولید شده توسط اعتبار سنجی

- مثالی از زمانبندی تولید شده با استفاده از اعتبار

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)

اسلاید ۴۶

شما های چند نسخه ای

- شما های چند نسخه ای جهت افزایش همروندی نسخه های قدیمی از مقادیر داده را نگهداری میکنند .
  - Multiversion Timestamp Ordering
  - Multiversion Two-Phase Locking
- هر نوشتن کامل باعث تولید و ثبت یک نسخه جدید از مقادیر داده می شود.
- از مهر زمانی برای مشخص کردن نسخه ها استفاده می شود.
- زمانی که دستور خواندن Q صادر می شود ، با توجه به مهر زمانی تراکنش یک نسخه مناسب از Q انتخاب می شود و مقدار نسخه انتخاب شده باز گردانده می شود.
- خواندن برای دریافت نسخه مناسب دارای انتظار نخواهد بود و فورا به آن پاسخ داده می شود.

اسلاید ۴۷

مرتب سازی مهر زمانی چند نسخه ای

- هر مقدار داده Q دارای یک توالی از نسخه ها می باشد <Q1 , Q2 , Q3 , ... , Qm>. . هر نسخه از Q_kدارای سه مقدار داده می باشد :
  - محتوا : مقدار نسخه Q_k
  - مهر زمانی نوشتن Q_k: مهر زمانی تراکنشی که نسخه Q_kرا ایجاد نموده است (نوشته)
  - مهر زمانی خواندن Q_k: بزرگترین مهر زمانی تراکنشی که بطور کامل نسخه Q_k را خوانده است.
- زمانی که یک تراکنش Ti نسخه جدید Q_k را از Q ایجاد می نماید مهر زمانی نوشتن Q_k و مهر زمانی خواندن آن با مقدار مهر زمانی تراکنش Ti مقدار دهی می گردد.
- مهر زمانی خواندن Q_k بروز رسانی می شود هرگاه یک تراکنش Tj ، Q_k را بخواند و مهر زمانی تراکنش Tj بزرگتر از مهر زمانی خواندن Q_k باشد.

اسلاید ۴۸

مرتب سازی مهر زمانی چند نسخه ای (ادامه)

- فرض کنید که تراکنش Ti دستور خواندن یا نوشتن Q را صادر نماید . اجازه میدهد Q_k علامت گذاری کند نسخه Q را که مهر زمانی نوشتن آن بزرگترین مهر زمانی نوشتن ، کوچکتر یا مساوی TS(Ti) است.
  - اگر تراکنش Ti دستور خواندن Q را صادر کند ، آنگاه مقدار بازگشتی محتوای نسخه Q_k است.
  - اگر تراکنش Ti دستور نوشتن Q را صادر کند :

    ۱. اگر TS(Ti) < R-timestamp(Q_k) آنگاه تراکنش Ti عقبگرد نموده.

۲. اگر $TS(Ti) = W\text{-}timestamp(Q_k)$ محتوای Q_k بازنویسی (overwritten) شده.

۳. در غیر اینصورت نسخه جدیدی از Q ایجاد شده.

- مشاهده می کنید که
  - خواندن ها همیشه موفقیت آمیز هستند
  - یک نوشتن توسط Ti رد میشود اگر تراکنش دیگری مانند Tj که (در ترتیب سریالی شده توسط مقادیر مهر زمانی تعریف شده) باید نوشته ی تراکنش Ti را بخواند ، مقدار نسخه ای که توسط تراکنش قدیمی تر از Ti ایجاد شده را خوانده باشد.
- پروتکل serializability را تضمین می کند.

اسلاید ۴۹

قفل گذاری 2 مرحله ای چند نسخه ای :

- تفاوت های بین تراکنش های فقط خواندنی و تراکنش های بروز رسانی
- **تراکنش های بروز رسانی** قفل های خواندن و نوشتن را ذخیره کرده و همه قفل ها را تا انتهای تراکنش نگه می دارند .که تراکنش های بروز رسانی از قفل 2 مرحله ای شدید پیروی می کنند.
  - در نتیجه هر عملیات نوشتن یک نسخه جدید از مقدار داده تولید و نوشته می شود.
  - هر نسخه از یک مقدار داده دارای یک مهر زمانی خاص خود می باشد و مقدار آن از طریق شمارنده ts-counter بدست می آید که به ازای هر پردازش موفق افزایش می یابد.
- **تراکنش های فقط خواندنی** دارای یک مهر زمانی هستند که قبل از شروع اجرای آنها توسط خواندن مقدار فعلی ts-counter تخصیص می یابد. آنها برای انجام خواندن ها از پروتکل مرتب سازی مهر زمانی چند نسخه ای پیروی می کنند .

اسلاید ۵۰

قفل گذاری 2 مرحله ای چند نسخه ای : (ادامه)

- زمانی که یک تراکنش بروز رسانی اقدام به خواندن یک مقدار داده می کند :
  - این کار شامل ایجاد یک قفل اشتراکی بر روی آن و خواندن آخرین نسخه از آن می شود .
- زمانی که اقدام به نوشتن یک مقدار می کند :
  - این کار شامل ایجاد یک قفل انحصاری بر روی آن ، سپس ایجاد یک نسخه جدید از آن مقدار داده و مقدار دهی مهر زمانی آن نسخه با مقدار بی نهایت خواهد بود.
- زمانی که تراکنش بروز رسانی Ti کامل شد ، پردازش commit انجام می شود :
  - Ti مقدار مهر زمانی نسخه هایی که ایجاد کرده را با ts-counter + 1 مقدار دهی می کند
  - Ti مقدار شمارنده ts-counter را یک واحد افزایش می دهد
- تراکنش های فقط خواندنی که بعد از افزایش مقدار ts-counter شروع شده اند مقادیر بروز رسانی شده توسط Ti را مشاهده خواهند نمود.

- تراکنش های فقط خواندنی که قبل از افزایش مقدار شمارنده ts-counter شروع به کار نموده اند مقادیر قبل از بروز رسانی توسط Ti را مشاهده خواهند نمود.
- تنها زمانبندی های serializable تولید می شوند.

اسلاید ۵۱

مسائل پیاده سازی (MVCC)

- ایجاد نسخه های متعدد سربار ذخیره سازی را افزایش می دهد.
  - تاپل های اضافی
  - فضای اضافی جهت ذخیره سازی اطلاعات هر نسخه برای هر تاپل
- نسخه هایی می توانند وجود داشته باشند فاقد ارزش :
  - برای مثال : اگر Q دارای 2 نسخه Q5 و Q9 باشد و قدیمی ترین تراکنش فعال مهر زمانی بزرگتری از 9 داشته باشد ، آنگاه Q5 هیچگاه دوباره نیاز نخواهد شد.

اسلاید ۵۲

جداسازی با عکس فوری (**Snapshot Isolation**)

- انگیزه : پرس و جو های با قابلیت تصمیم گیری که مقادیر با حجم بالا را می خوانند با تراکنش های OLTP که سطر های کمی را بروز رسانی می کنند تعارض همروندی دارند
  - نتایجی با عملکرد ضعیف
- راه حل اول : برای تراکنش های فقط خواندنی از وضعیت پایگاه داده یک عکس فوری منطقی تهیه شود و برای تراکنش های خواندنی و نوشتنی از قفل گذاری معمولی استفاده شود.
  - قفل گذاری 2 مرحله ای چند نسخه ای
  - کار خوبی است ، اما سیستم چگونه باید تراکنش های فقط خواندنی را تشخیص دهد؟
- راه حل دوم :برای تمامی تراکنشها از وضعیت پایگاه داده عکس فوری تهیه می شود ، تنها بروز رسانی ها از قفل 2 مرحله ای استفاده می کنند برای جلوگیری از بروز رسانی های همزمان.
  - مشکلات : انواع ناهنجاری ها مثل منجر شدن به از دست رفتن بروز رسانی ها
  - راه حل جزئی : سطح جدا سازی عکس فوری (در اسلاید بعدی)
    - ✓ پیشنهاد شده توسط برنسون و همکارانش SIGMOD 1995
    - ✓ مورد استفاده بسیاری از سیستم های پایگاه داده :
      - برای مثال : Oracle, PostgreSQL, SQL Server 2005

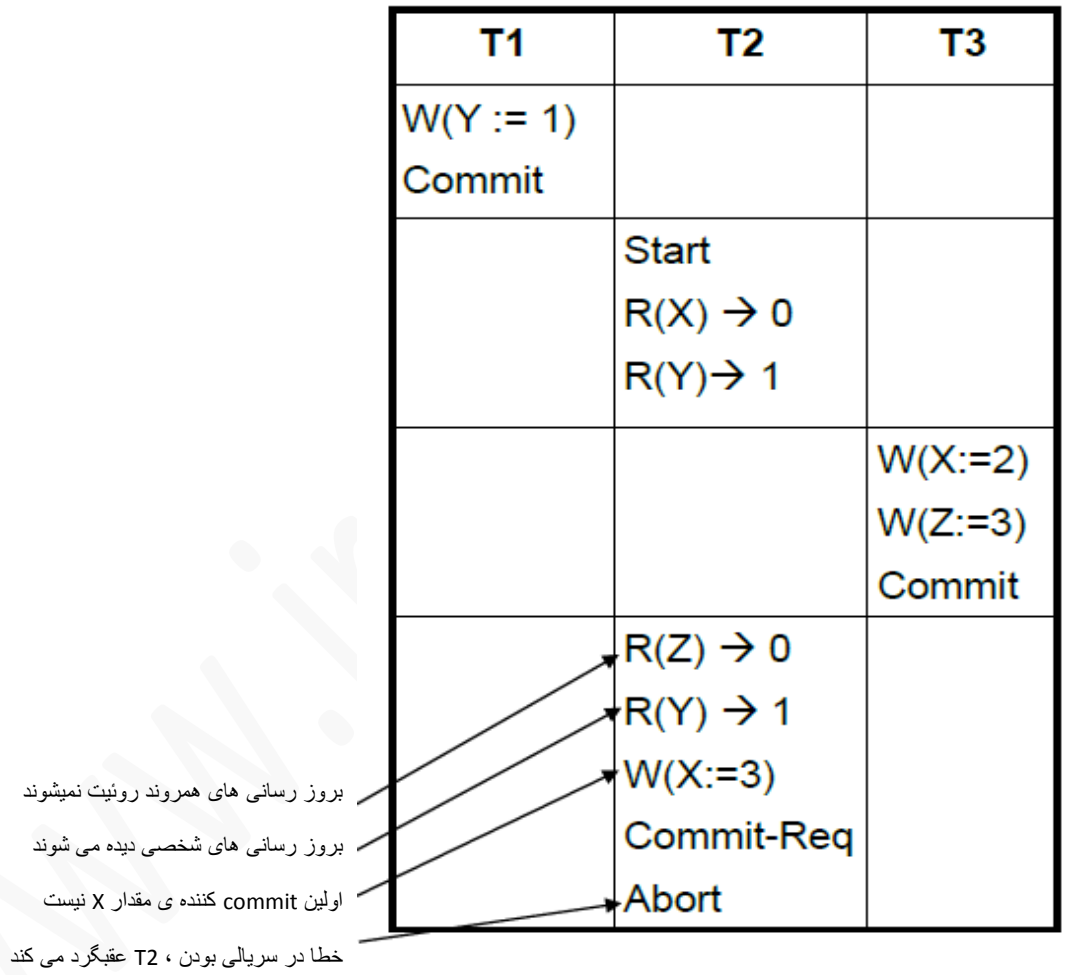
اسلاید ۵۳

جداسازی با عکس فوری (**Snapshot Isolation**)

- تراکنش T1 با جداسازی توسط عکس فوری شروع به کار می کند

- یک عکس فوری از داده های commit شده در ابتدای کار تهیه می شود
- خواندن و تغییر داده ها در عکس فوری خود او صورت میگیرد
- بروز رسانی های تراکنش های همروند برای T1 قابل روئیت نیستند
- نوشتن های T1 در زمان commit نمودن آن کامل می شوند
- **قانون هرکه اول commit کرده برنده است :**

✓ Commit میکند اگر ، داده ای را که T1 قصد دارد بنویسد هیچ تراکنش همروند دیگری آن را ننوشته باشد.



اسلاید ۵۴

مزایای Snapshot Isolation

- از عملیات خواندن هیچوقت جلوگیری نمی شود
  - همچنین از فعالیت دیگر تراکنشها جلوگیری نمی شود
- کارایی مشابه سطح Read Committed دارد
- جلوگیری از ناهنجاری های معمول
  - No [i]dirty read
  - No [ii]lost update

- No non-repeatable read
- No phantoms (مقادیر برگزیده قابل تکرار هستند)

- مشکلات Snapshot Isolation
  - جداسازی عکس فوری همیشه به اجرای سریالی منجر نمی شود
    - ✓ Serializable : در میان دو تراکنش همروند یکی تاثیرات دیگری را می بیند
    - ✓ در جداسازی عکس فوری : هیچکدام تاثیرات دیگری را نمیبینند
  - در نتیجه : محدودیت های جامعیت می تواند نقض شود

اسلاید ۵۵

Snapshot Isolation

- مثالی از مشکلات SI :
  - T1: x:=y
  - T2: y:=x
  - در ابتدا x=3 و y=17 است
    - ✓ در اجرای سریال : x=?? , y=??
    - ✓ اگر هر دو تراکنش توسط جداسازی عکس فوری و در یک زمان شروع نمایند : x=?? , y=??
  - این پدیده را نوشتن ضربدری (اریب) گویند
  - پدیده ضربدری در عملیات درج نیز رخ می دهد
    - به عنوان مثال :
      - ✓ پیدا کردن بزرگترین شماره سفارش از میان سفارش ها
      - ✓ ایجاد یک سفارش جدید با شماره سفارشی که برابر بزرگترین شماره سفارش قبلی + 1 است

اسلاید ۵۶

ناهنجاری های جداسازی عکس فوری (Snapshot Isolation Anomalies)

- SI زمانی که تراکنش ها مقادیر داده مختلفی را تغییر میدهند serializability را در هم می شکند ، هرکدام مبتنی بر وضعیت قبلی مقداری عمل می کنند که دیگری تغییر داده است.
  - در عمل عمومیت زیادی ندارد
    - ✓ برای مثال : TPC-C benchmark به درستی طبق جداسازی عکس فوری اجرا می شود
    - ✓ زمانی که تراکنش ها در بازیابی داده های مختلف تداخل می کنند ، و همچنین معمولاً مقدار مشترکی وجود دارد که هر دو بازیابی نموده اند (مثل جمع کل) بنابراین SI یکی از آن دو را سقط خواهد نمود.
  - اما آیا این اتفاق می افتد؟
    - ✓ توسعه دهندگان نرم افزار باید مراقب نوشتن های ضربدری باشند
- SI همچنین می تواند باعث ایجاد تراکنش فقط خواندنی غیر معمول گردد ، که در آن ، تراکنش فقط خواندنی ممکن است وضعیت متناقضی را رؤیت کند حتی اگر آنهایی که بروز رسانی را انجام داده اند Serializable باشند.

اسلاید ۶۰

سطوح ضعیف سازگاری (Weak Levels of Consistency)

- سازگاری درجه دو: تفاوت آن با قفل گذاری 2 مرحله ای در این است که قفلهای اشتراکی میتوانند در هر زمانی آزاد شوند و قفل ها در هر زمانی میتوانند گرفته (گذاشته) شوند
  - قفلهای انحصاری باید تا انتهای تراکنش نگه داشته شوند
  - Serializability تضمین نمی شود، برنامه نویس بایستی اطمینان حاصل کند که وضعیتی دارای خطا برای پایگاه داده رخ نخواهد داد.
- ثبات کرسر (Cursor stability)
  - جهت عملیات خواندن، هر تاپل قفل می شود، خواندن انجام می شود و قفل فوراً آزاد می گردد
  - قفل های انحصاری تا پایان کار تراکنش نگه داشته می شوند
  - حالت خاصی از سازگاری درجه دو

اسلاید ۶۱

سطوح ضعیف سازگاری در SQL (Weak Levels of Consistency in SQL)

- SQL امکان اجراهای غیر Serializable را میدهد
  - Serializable : سطح پیشفرض می باشد
  - Repeatable read : فقط اجازه میدهد که رکوردهای commit شده خوانده شوند و تکرار عملیات خواندن باید مقدار یکسانی رابرگرداند (بنابراین قفل های خواندن باید حفظ شوند)
    - ✓ با این وجود، نیازی به جلوگیری از پدیده phantom[1] نیست

      T1 ممکن است برخی از رکوردهای درج شده توسط T2 را ببیند ، اما باقی رکوردهای درج شده توسط T2 را نبیند
  - Read committed : مشابه سازگاری درجه دو می باشد، اما اغلب سیستمها آن را به صورت ثبات کرسر (Cursor stability)پیاده سازی میکنند.
  - Read uncommitted : حتی امکان خواندن داده های commit نشده را نیز می دهد
- در بسیاری از سیستم های پایگاه داده، سطح Read committed به عنوان سطح پیشفرض سازگاری در نظر گرفته شده است
  - در هنگام نیاز باید به طور صریح به Serializable تغییر داده شود
    - ✓ Set isolation level serializable

[1] پدیده phantom زمانی رخ میدهد که یک عمل درج یا حذف بر روی سطری انجام شود، به طوری که این سطر متعلق به محدوده - ای از سطرها است که توسط یک تراکنش خوانده میشود. در واقع اگر دو پرس و جوی مشابه برای به دست آوردن محدودهای از رکوردها اجرا شوند، مجموعه رکوردهایی که پرس و جوی دوم باز می گرداند با نتایج پرس و جوی اول متفاوت است.

[i] dirty read : هنگامی که یک تراکنش مقدار داده ای را از سطری بخواند که آن مقدار داده توسط تراکنش دیگری که هنوز Commit نکرده تغییر داده شده باشد.

[ii] lost update : زمانی که یک تراکنش ثانویه مقدار ثانویه ای را برای یک Data Item بنویسد که یک تراکنش همروند اولیه ای قبل از آن مقداری را برای آن Data Item نوشته باشد ، در نتیجه مقداری که اول نوشته شده است از دست می رود و تراکنش های همروندی که نیاز به خواندن آن مقدار داده دارند ، مقداری اشتباه را خواهند خواند.