**سوال 5 فصل 15 : به وسیله یک مثال نشان دهیدschedule های ممکنی وجود دارند که تحت tree protocol هستند ولی تحت پروتکل two phase locking نیستند و برعکس.**

**راه حل :** Database graph با ساختار درختی زیر را در نظر بگیرید :



schedule ممکن که تحت tree protocol است ولی تحت 2pl نیست



Schedule ممکنی که تحت 2pl است ولی تحت tree protocol نیست



**سوال 6 فصل 15 : تعمیم زیر را بر روی tree locking protocol در نظر بگیرید که اجازه هر دو قفل shared و exclusive داده شده :**

* **یک تراکنش می تواند یا یک تراکنش فقط خواندنی باشد که در این حالت تنها می تواند درخواست lock s بدهد یا می تواند یک تراکنشupdate باشد که در این حالت تنها می تواند درخواست lock x بدهد.**
* **هر تراکنش باید قوانین tree protocol را دنبال کند. تراکنش های فقط خواندنی ممکن است در ابتدا هر data item ای را قفل کنند در حالیکه تراکنشهای update باید در ابتدا ریشه را قفل کنند.**

**نشان دهید پروتکلserializability را تضمین میکند و رها از بن بست است(بن بست ندارد)**

ارزشی ندارد اگر پروتکلmodify شود و به تراکنشهای update اجازه دهد در ابتدا روی هر data item ای قفل بزند، در این صورت اجراهای non serializable میتواند رخ دهد. به طور مستقیم ، یک اجرای طولانی از تراکنش فقط خواندنیT0 ممکن است اولویت داشته باشد به تراکنش update (به روز رسانی) T1 روی گره a، اما به نگه داشتن قفل روی گره فرزند b ادامه دهد. بعد از اینکه T1،a را update کرد و commit شد، تراکنش فقط خواندنیT2،a را میخواند و بنابراین T1 مقدمبرT2 است. با این حالT2 حالا از T1 سبقت گرفته و lock s را روی گره b و گره فرزندشc گذاشته(با lock x امکان پذیر نیست) و c را میخواند و commit میشود. پس از آن تراکنشT3، c را update میکند و commit میشود.پس از آنT0 روی c، lock s میگذارد و آن را میخواند. بنابراین یک سیکل (دور) وجود دارد

در واقع، دو تراکنش فقط خواندنی دو ترتیب متفاوت از همان دو تراکنش update را می بینند. به خاطر اینکه تراکنش های update نیاز دارند همیشه از ریشه شروع شوند، پروتکل تضمین می کند که موقعیت بالا نمی تواند رخ دهد. در مثال بالا T3 مجبور بود از ریشه شروع شود و بنابراین T0 نمی توانست serialized باشد بعد ازT3. و این به صورت رسمی اثبات میشود به وسیله نتیجه چندین تراکنش فقط خواندنی ( ما باید بپذیریم، خیلی شهودی نیست). ما آن را بعد از معرفی چندین نماد، ارائه می دهیم:

مجموعه ای از تراکنشهایupdate

مجموعه ای از تراکنشهای فقط خواندنی

مجموعه ای از data item هایی که به وسیلهTi قفل شدند

Q پروتکل جدید است. فرض کنیدTi و Tk تراکنشهای update هستند وTj یک تراکنش فقط خواندنی است، سپس اگر Tj با Ti و Tk در مجموعه ای از data item ها که به آن دسترسی دارد همپوشانی داشته باشد، چونTi وTk هر دو از ریشه شروع میشوند،Ti و Tk باید روی مجموعه data item هایی که دسترسی دارند همپوشانی داشته باشند. به عبارت دیگر :

یک schedule دلخواه از تراکنشها در پروتکلQ را در نظر بگیرید. K تعداد تراکنشهای فقط خواندنی است که در این schedule شرکت می کنند. ما به وسیله استنتاج نشان می دهیم که کوچکترین چرخه(حلقه) ای در (→,T) وجود ندارد، جایی که Ti → Tj است به این معنی است که Ti اولویت دارد برTj برای conflict روی همانdata item . اجازه دهید چرخه را بدون از دست دادن کلیت درنظر بگیریم

اینجا ما می دانیم کهm>=2k است بنابراین باید در هر طرف یک تراکنش فقط خواندنی،یک تراکنش update قرار بگیرد، در غیر این صورت حلقه تشکیل نمیشود.

* K=0 : در این حالت هرTi در$Q^{x}$ است بنابراینQ میشود tree protocol اصلی که serializability و رهایی از بن بست را تضمین می کند.
* K=1 : Ti یک تراکنش یکتا در $Q^{S}$ است. ما می توانیم Ti را با $T\_{i}^{x}$ جایگزین کنیم. نتیجهschedule پیروی می کند از tree protocol اصلی کهserializability و رهایی از بن بست را تضمین می کند.
*  K>1 : اجازه دهید $Q^{s}˄T\_{i}$ سپس {*Ti*−1, *Ti*+1 } ∈ *QX*باشد. از آنجایی که *m* ≥ 2*k* ≥ 4, *i* −1 ≠ *i* +1 (ماژول m) است. با توجه به این فرض ها تعریف تعریف پروتکل آورده شده است

بنابراین  *Ti*−1 → *Ti*+1 یا *Ti*+1 → *Ti*−1 . در هر حالت حلقه اصلی کمینه نیست ( درحالت اول،Ti میتواند از حلقه حذف شود. در حالت دوم یک حلقه وجود دارد که Ti-1 وTi+1 و Ti را درگیر می کند.) به وسیله این تناقض می تواند هیچ حلقه ای وجود نداشته باشد و بنابراین پروتکل جدیدserializability را تضمین میکند. برای نشان دادن رهایی از بن بست بایدwait ها را برای گراف درنظر گرفت. اگر یالی ازیک تراکنش آپدیت Tj به تراکنش آپدیت دیگر(Ti) وجود داشته باشد، به آسانی می توان دید کهTi قبل از Tj ، ریشه را قفل می کند. به طور مشابه اگر یک یال ازیک تراکنش آپدیت Tj به یک تراکنش فقط خواندنی Tk وجود داشته باشد که به نوبه خود می تواند برای یک تراکنش آپدیتTi صبر کند(wait) می توان دید که Ti باید قبل از Tj ریشه را قفل کند. به عبارت دیگر یک تراکنش آپدیت قدیمی تر نمی تواند برای یک تراکنش آپدیت جوان تر صبر کند و بنابراین هیچ سیکلی در گراف انتظار می تواند وجود نداشته باشد.