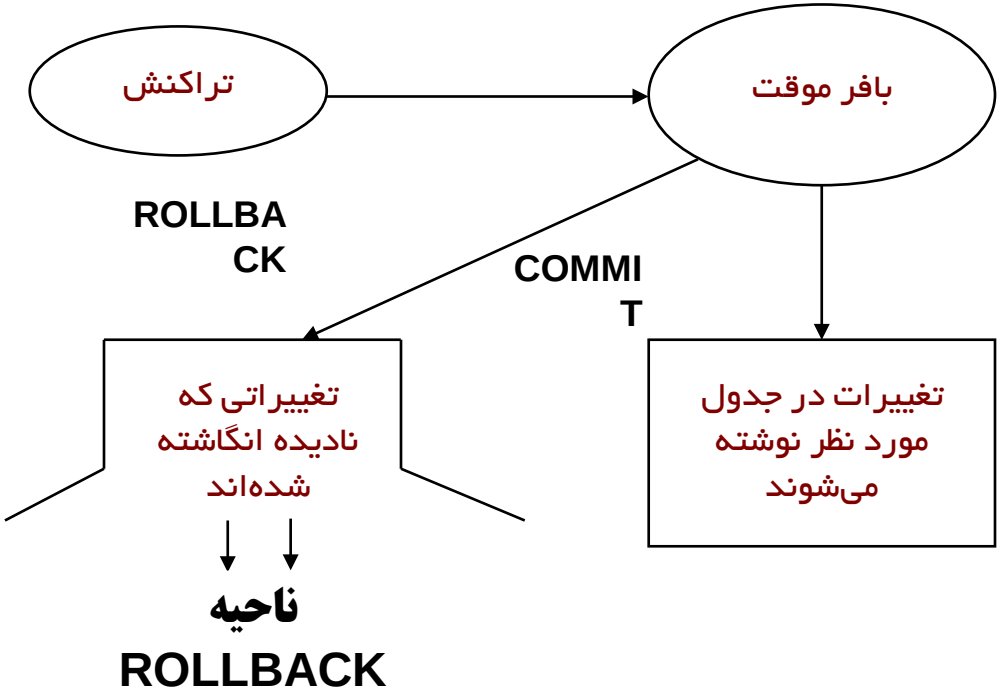
فصل دوم

مفهوم تراکنش(بخش اول)

ارائه دهنده : محمد فرجی مهماندار

مفهوم تراکنش در پایگاه داده ها

- تراکنش، واحد برنامه نویسی است که شامل یکسری عملیات مرتبط برای دسترسی و تغییر اطلاعات یک بانک اطلاعاتی است که در جهان واقعی در حکم یک عمل واحد تلقی می شوند.تراکنش عبارتست از واحد سازگار و قابل اطمینان یک پردازش مشخص در پایگاه داده
- هر برنامه ای که توسط کاربر در محیط بانک اطلاعاتی اجرا میشود تراکنش نام دارد. تراکنش یک واحد منطقی از کاراست و معمولا شامل چندین عمل بانک اطلاعاتی است.



مفهوم تراکنش در پایگاه داده ها

- **تعریف تراکنش**
  - مثال: انتقال مقدار ۵۰ دلار ازحساب A به حساب B

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B*)

مفهوم تراکنش در پایگاه داده ها

- هر تراکنش معادل یک رخداد در دنیای واقعی است.
- تراکنش همواره به DBMS تسلیم می شود و DBMS در اعمال هرگونه کنترل و حتی به تعویق انداختن و ساقط کردن آن آزادی عمل دارد.
- واحد کار DBMS تراکنش است.
- هر تراکنش شامل مجموعه ای از عملیات است که با دستورشروع تراکنش(begin transaction) آغاز و با یک عمل commit ویا undo پایان می پذیرد.
- تراکنش ممکن است بارها اجرا شود ولی هربار اجرا یک تراکنش محسوب می شود.
- هدف اصلی تراکنش ها حفظ جامعیت و صحت بانک اطلاعاتی است. چرا که در بانک اطلاعاتی آنچه در درجه اهمیت دارد داده است نه برنامه. داده های بانک اطلاعاتی را مانا(PERSISTENT) می نامند، برنامه ها می آیند و میروند اما داده ها می مانند.

نکات مهم در باره تراکنش

- طراحی صحیح (correctness) : برنامه نویس باید عملیات اجرایی یک تراکنش را بصورت واحد ویکپارچه طراحی کند و این به خود DBMS ربطی ندارد.
- خواندن اطلاعات : هر مورد اطلاعاتیمورد نیاز یک تراکنش باید فقط یک بار خوانده شود.
- نوشتن اطلاعات : هر مورد اطلاعاتی مورد عمل در تراکنش در صورت تغییر فقط یکبار نوشته شود.
- همروندی : هردو تراکنشی که در طول اجرا یک اشتراک زمانی داشته باشد و همزمانی نوع خاصی از همروندی است که شروع یکسان دارند
- کنترل همروندی : زمانی که در یک سیستم چندین کاربر داریم باید کنترل همرندی داشت چون ممکن است کاربران بخواهند به داده مشترک دستیابی داشته باشند. به عبارت دیگر مجموعه فعالیت هایی است برای آنکه دو یا چند تراکنش که بر روی داده مشترک کار می کنند بتوانند بدون تداخل با یکدیگر اجرا شوند.

تضمین جامعت بانک اطلاعاتی :

آقای جیم گری (Jim Gary) در سال ۱۹۸۱ ثابت کرد که چهار کنترل زیر لازم است روی تمامی تراکنش ها در بانک اطلاعات اعمال گردد تا صحت و جامعیت آن تضمین شود این کنترل ها به خواص ACID معروفند.

۱ – یکپارچگی (Atomicity)

۲– همخوانی (Consistency)

۳– انزوا(Isolation)

۴– پایایی (Durability)

ویژگیهای تراکنش

۱– یکپارچگی (Atomicity):

این خاصیت به همه یا هیچ موسوم است .منظور این است که یا تمام دستورات تراکنش باید اجرا شود یا هیچکدام از آنها نباید اجرا شود. مثلا تراکنشی می خواهدمبلغی را از حسابی به حساب دیگر منتقل کند فرض کنید بخش اول کار برداشت پول در یک ماشین و بخش دوم کار واریز پول در ماشینی دیگر انجام شود. حال در نظر بگیرید پس ازانجام بخش اول برداشت پول ارتباط با ماشین دوم ناگهان قطع شود بدیهی است که دراینحالت باید پول برداشت شده به همان حساب بازگردانده شود.

۲– همخوانی (Consistency):

این خاصیت به این صورت بیان می گردد که:(هر تراکنشی اگر به تنهایی اجرا شود بانک اطلاعات را ا ز حالتی صحیح به حالت صحیح دیگری منتقل می کند) یعنی خاصیت می گوید که هر تراکنش باید تمامی قوانین جامعیت بانک اطلاعاتی رارعایت کند. تراکنش ممکن است دو نوع پایان داشته باشد :

الف) پایان نا موفق که آن را سقوط (Roolback)می نامند.

ب)پایان موفق که آن را انجام (Commit) می نامند.

ویژگیهای تراکنش

۳– انزوا (Isolation) :

در بانک اطلاعاتی ممکن است تراکنش های همروند وجود داشته باشد . بر طبق خاصیت انزوا ، همروندی تراکنش ها باید کنترل شود تا اثر مخرب بر روی هم نداشته باشند به عبارت دیگراثر تراکنش های همروند، روی یکدیگر چنان است که گویا هر کدام در انزوا انجام می شود. یعنی به هنگام سازیT1 توسط تراکنش دیگری مثل T2 قابل مشاهده نیست مگر اینکه T1 عمل COMMITرا اجرا کند.

۴– پایایی (Durability) :

بر اساس این خاصیت تراکنش هایی که به مرحله انجام (COMMIT)برسند اثرشان ماندنی است و هرگز به طور تصادفی از بین نمی روند. مثلا اگر مبلغی به حسابی واریز شود تراکنش مربوطه انجام یافته، حتی در صورت وقوع آتش سوزی در آن شعبه بانک، مشتری نباید متضرر شود یعنی عمل واریز قبل از اعلام انجام موفق باید در جای دیگری نیز ثبت شده باشد.

مثال تراکنش و خصوصیات تراکنش

1. **read**(A)
2. $A := A - 50$
3. **write**(A)
4. **read**(B)
5. $B := B + 50$
6. **write**(B)

- خاصیت سازگاری
  - مجموع مقادیر A و B پس از اجرای تراکنش تغییر نمی کند
- اتمی بودن
  - اگر تراکنش پس از مرحله ۳ و قبل از مرحله ۶ متوقف گردد، سیستم تضمین می کند که تغییرات در بانک ثبت نگردند.
- پایداری
  - پس از آنکه اجرای تراکنش مورد تایید قرار گرفت و تراکنش کامل گردید ، این تغییرات در بانک پایدار خواهند بود.

مفهوم Isolation

- T_1 T_2
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)

     read(A), read(B), print(A+B)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*

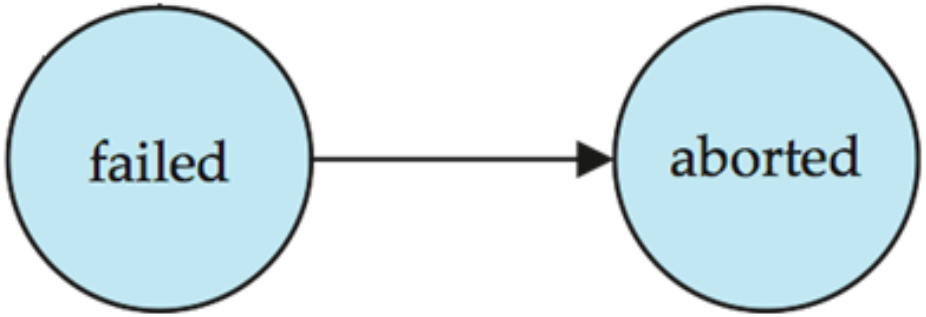
- جداسازی
  - اگر بین مراحل ۳و۶ یک تراکنش دیگر اجازه دستیابی به تغییرات در بانک را داشته باشد باعث ناسازگاری در بانک خواهد گردید.
  - جداسازی تراکنش ها می تواند با اجرای ترتیبی آنها تضمین شود ولی ما نیاز به اجرای همزمان تراکنش ها داریم

حالت های اجرای تراکنش

- Active (فعال): وقتی تراکنش شروع به اجرا می کندیا در حال اجراست
- Partially committed
  - پس از اجرای آخرین دستور تراکنش به این حالت می رود.
  - زمانی پیش می اید که تراکنش همه کارهای مربوطه را انجام داده و کافی است بخش های مختلفی را که با آنها سرو کار داشته جهت پایان دادن به کار هماهنگ کند. این مرحله در تراکنشهای طولانی ، به ویژه در بانک اطلاعاتی نامتمرکز ، اهمیت بسیار دارد.در نتیجه، قبل از پایان دادن به یک تراکنش لازم است همه بخش ها باهم هماهنگ باشد.
- Failed

زمانی اتفاق می افتد که سیستم به هر دلیلی تشخیص دهد که تراکنش نباید ادامه یابد. این تصمیم در مواقعی اتخاذ می شود که یا تراکنش مرتکب خطا شود ویا سر راه تراکنش های دیگر قرار گیرد.

حالتهای اجرای تراکنش(حالات پایانی)



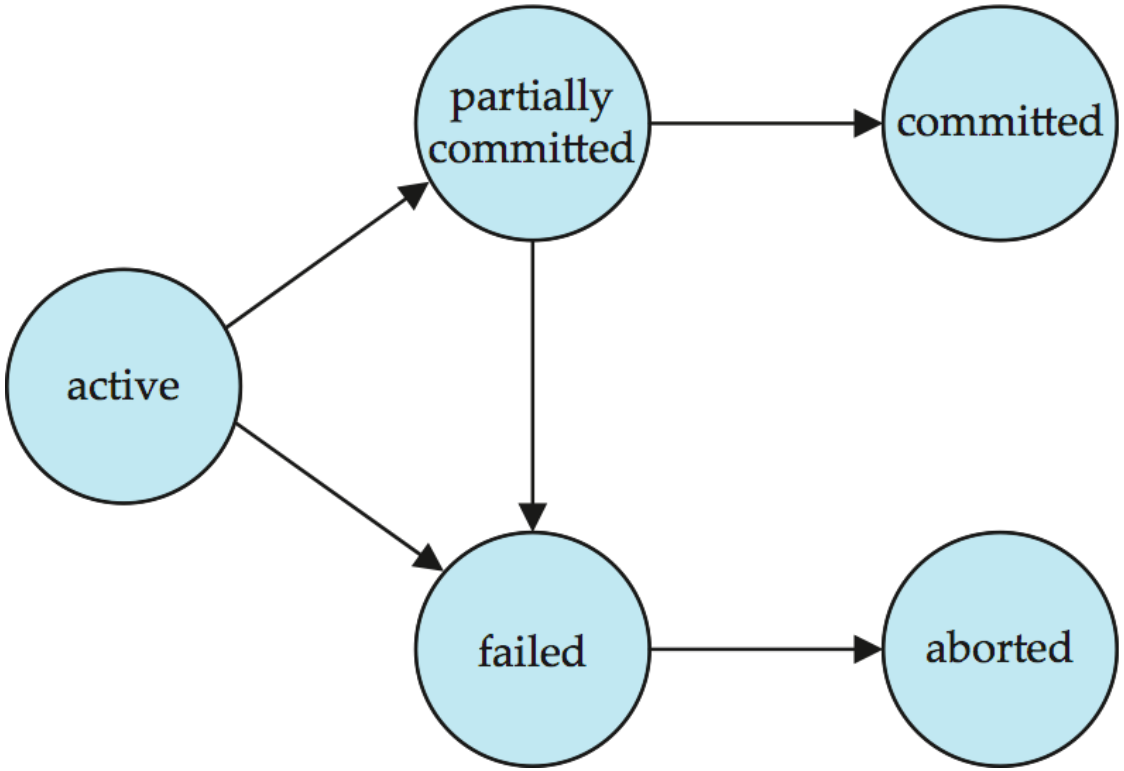
- Aborted(سقوط)

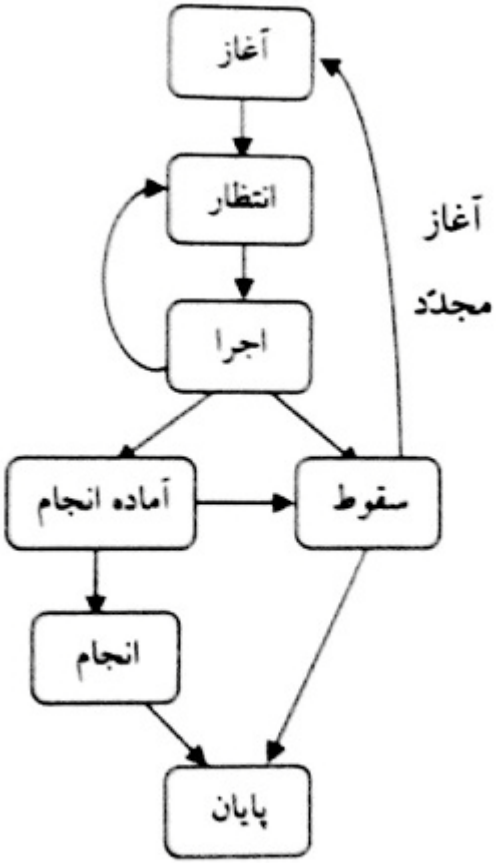
- در این حالت در حین اجرای تراکنش اشکالی پیش آمده است که منجر به توقف اجرای آن شده است.
- در صورت بروز اشکال در اجرای یک تراکنش، برای حفظ یکپارچگی اطلاعات، اثرات احتمالی بخشی از تراکنش که اجرا شده روی بانک اطلاعاتی باید خنثی شود. به این حالت برگشت Rollbackگفته می شود.

حالتهای اجرای تراکنش(حالات پایانی)

- دوگزینه پس از ساقط شدن یک تراکنش مطرح است
  - restart the transaction
    - can be done only if no internal logical error
  - kill the transaction
- Committed(انجام)
  - مرحله ای است که همه بخش ها برای نهایی کردن کار توافق کرده اند و عملیات تراکنش بطورکامل موفقیت آمیز انجام شده و اثر آن نیز ثبت شده است پس از انجام تراکنش خنثی کردن تغییرات احتمالی تراکنش روی بانک غیر ممکن است. در این صورت علاوه بر اعلام پایان موفق تراکنش به کاربر تضمین می گردد که در صورت بروز هرگونه اتفاقی، آنچه که این تراکنش انجام داده خدشه دار نمی شود.

حالتهای اجرای تراکنش





مراحل اجرای تراکنش

Oracle and Transactions

- Oracle is transaction oriented; that is, Oracle uses transactions to ensure data integrity.
- A *transaction* is a series of SQL data manipulation statements that does a logical unit of work.
  - For example, two UPDATE statements might credit one bank account and debit another.
  - If your program fails in the middle of a transaction, Oracle detects the error and rolls back the transaction
  - Thus, the database is restored to its former state automatically.

Oracle and Transactions

- You use the COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION commands to control transactions.
- COMMIT makes permanent any database changes made during the current transaction.
- ROLLBACK ends the current transaction and undoes any changes made since the transaction began.
- SAVEPOINT marks the current point in the processing of a transaction.
- SET TRANSACTION
  - sets transaction properties such as read/write access and isolation level.

اجرای همروند تراکنش ها

کنترل همروندی :

مجموعه فعالیت هایی است برای آنکه دو یا چند تراکنش که بر روی داده مشترک کار می کنند بتوانند بدون تداخل با یکدیگر اجرا شوند.

واحد کنترل همروندی وظیفه اجرای همروند تراکنش ها با حفظ خاصیت انزوا را برعهده دارد

ترمیم :

مجموعه فعالیت هایی است که تضمین می کند داده ذخیره شده در پایگاه داده ماندگار شود

تضمین خواص پایایی ویکپارچگی تراکنش نیز بر عهده واحدی به نام واحد مدیریت ترمیم می باشد.یعنی اگر تراکنش با موفقیت خاتمه یافت، طبق خاصیت پایایی اثر کارهای آن تراکنش روی بانک اطلاعاتی دائمی وپایدار گردد وچنانچه به هر دلیل،فقط برخی از دستورات تراکنش اجرا شدند، اثر این دستورات اجرا شده خنثی(ملغی – بلااثر) گردد.

اجرای همروند تراکنش ها

- محیط سیستم پایگاه داده یک محیط چند کاربری است و ممکن است تعدادی تراکنش بخواهند به داده مشترک دسترسی پیدا کنند.
- در یک سیستم پایگاه داده امکان اجرای همزمان/همروند تراکنش ها وجود دارد.
- همروندی یعنی اجرای غیر متوالی دو یا چند تراکنش.
- واحد کنترل همروندی وظیفه اجرای همروند تراکنش ها با حفظ خاصیت انزوا را برعهده دارد
- سیستم باید به دو یا چند تراکنش امکان دهد تا به داده مشترک به طور همزمان دستیابی داشته باشند.
- بدیهی است که همروندی تراکنش ها بایستی کنترل شودو عملهای تراکنش هایی که به طور غیر متوالی اجرا می شوند و به داده مشترک دسترسی دارند هماهنگ شوند.
- اگر تراکنش ها روی داده های مشترک کار نکنند کنترل همروندی لازم نخواهد بود.

اجرای همروند تراکنش ها

- مزایای همروندی دو یا چند تراکنش
  - افزایش بهره وری CPU
    - هرتراکنش شامل چندین مرحله است برخی I/O را درگیر می کنند برخی CPU، امکان موازی سازی دستورات تراکنش هاو موازی سازی تراکنش ها کارایی را بالا می برد.
  - افزایش گذر دهی throughput
    - تعداد تراکنش های اجرا وتمام شده در واحد زمان بیشتر شده
  - کاهش میانگین زمان پاسخ دهی
    - تراکنش های کوتاه نیازی به منتظر ماندن تا پایان تراکنش های بلند نیستند.

مثال، اجرای همزمان (همروند) تراکنش ها

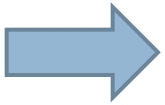
- در اجرای همروند، دو تراکنش می توانند به چندین حالت اجرا شوند.
- مثال:

T_1	T_2
read (A)	read (A)
$A := A - 50$	$temp := A * 0.1$
write (A)	$A := A - temp$
read (B)	write (A)
$B := B + 50$	read (B)
write (B)	$B := B + temp$
commit	write (B)
	commit

اجرای همزمان تراکنش ها (حالت ۱)

- نه داده ای گم می شود و سازگاری برقرار است.

T_1	T_2
read (A)	read (A)
$A := A - 50$	$temp := A * 0.1$
write (A)	$A := A - temp$
read (B)	write (A)
$B := B + 50$	read (B)
write (B)	$B := B + temp$
commit	write (B)
	commit



T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

اجرای همزمان تراکنش ها (حالت ۲)

- ممکن است داده ای گم شود و یکپارچگی نقض شود.

T_1	T_2
read (A)	read (A)
$A := A - 50$	$temp := A * 0.1$
write (A)	$A := A - temp$
read (B)	write (A)
$B := B + 50$	read (B)
write (B)	$B := B + temp$
commit	write (B)
	commit

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

کنترل همروندی تعبیه می شود تا مانع از بروز مشکل شود

کنترل همروندی

- عدم کنترل همروندی باعث ایجاد سه پیامد منفی است:
  - به هنگام سازی گم شده (Lost Update Problem) :
  - این مشکل موقعی بروز می کند که تراکنشی بلافاصله بعد از تراکنش دیگری که مقداری را برای داده ای نوشته است بخواهد مقدار جدیدی برای آن د اده بنویسد.
  - خواندن داده تثبیت نشده (The Uncommitted dependency Problem =dirty_read)
    - تراکنشی بخواهد مقدار موقتا تغییر یافته توسط تراکنش دیگر را بخواند
  - بازیابی ناسازگار (The Inconsistency Analysis Problem)

مشکل بروزرسانی گم شده (Lost Update Problem)

تراکنش T_1 در نقطه ای از زمان داده مشترک t را تغییر می دهد (مثلا به ۷۰) پس از آن تراکنش دیگر T_2 همان داده را تغییر می دهد (مثلا به ۱۰۰) آنگاه تراکنش اولی مقدار t را می خواند وانتظار همان مقداری را که نوشته دارد غافل از اینکه دیگری آن را تغییر داده است

مثال :

- فرض کنیم تراکنش A مقدار داده t را در لحظه t_1 می خواند
- تراکنش B همان مقدار را در لحظه t_2 می خواند.
- تراکنش A مقدار داده t را در لحظه t_3 تغییر می دهد.
- تراکنش B همان مقدار t را که در لحظه t_2 خوانده در لحظه t_4 تغییر می دهد.
- در لحظه t_4 مقدار تغییر داده شده t در لحظه t_3 گم می شود.

Transaction A	time	Transaction B
Read (t)	t1	
	t2	Read (t)
Update t	t3	
	t4	Update t

The Uncommitted dependency Problem

خواندن داده تثبیت نشده (dirty_read):

تراکنش T_1 در نقطه ای از زمان داده را تغییر می دهد. سپس تراکنش T_2 همان داده را می خواند ولی بعدا تراکنش T_1 به دلیلی ساقط می شود و داده به حال اولیه برمی گردد. تراکنش T_2 بی خبر از همه جا از آن داده استفاده می کند و به راه خود ادامه می دهد ودچار خطا واشتباه می شود.

Transaction A	x=50	Transaction B
	x=15	Update x
Read(x)		
		Rollback

Transaction A	Time	Transaction B
		Update x
Update x		
		Rollback

The Inconsistency Analysis Problem

T_1	A=40 ,B=50 ,C=30 SUM=120	T_2
Read (A)	A=40	
Read (B)	**B=50**	
	C=30	Read (C)
	C=20	C=C-10
	C=20	Write(C)
	A=40	Read A
	A=50	A=A+10
	A=50	Write(A)
		Commit
Read (C)	**C=20, SUM=110**	

- بازیابی ناسازگار :

این مشکل وقتی بروز می کند که تراکنش T_1 مقدار دوداده نظیر B,A را بروز رسانی می کند و تراکنش T_2 یکی از این دوداده مثل Aرا پس از بروز رسانی می خواند و مقدارداده دیگر Bرا پیش از بروز رسانی .

- لذا نتیجه T_2 نادرست است.

کنترل همروندی

- این اشکالات بخاطر این رخ داد که تراکنش ها اثر همدیگر را ندیدند.کنترل همروندی باید شرایطی را فراهم آورد که تراکنش ها اثر همدیگر را ببینند.
- کنترل همروندی بخشی از DBMS است که مانع از بروز مشکلات ناشی از همروندی می شود. به عبارت دیگر امکان تاثیر تراکنش ها را برهم از بین می برد.
- کنترل همروندی با استفاده از زمانبندی(**برنامه ریزی**) انجام می شود.
- **برنامه ریزی Scheduling**
  - هر DBMS برای اجرای کنترل همروندی یک **زمانبند** ایجاد می کند که این **زمانبند** به طور پویا ایجاد شده و تراکنش ها در آن وارد و اجرا می شوند.

زمانبند

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

انواع زمانبند :

- خطی یا ترتیبی یا پی در پی (Serial)
  - تراکنش ها پشت سرهم اجرا می شوند.
- همروند (Concurrent)
  - تراکنش ها تماما با هم اجرا می شوند.
- انواع زمانبند همروند :
  - ترتیب پذیریا پی در پی پذیر(Serializable)
  - ترتیب ناپذیر(non serializable)

همروندی موقعی معتبر است که ترتیب پذیر باشند.

A serial schedule in which T_1 is followed by T_2 :

زمانبند های معادل (Equivalent Schedule)

زمانبند A معادل B است اگر روی یک پایگاه داده واحد جواب یکسانی را تولید کنند.

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

پی در پی پذیری (Serializable)

- **زمانبند** ای که خروجی آن معادل اجرای یک **زمانبند** پی در پی (ترتیبی یاخطی) باشد
- نتیجه حاصل ازاجرای همروند چند تراکنش باید دقیقا شبیه اجرای تک تک آنها به طور مجزا باشد.
- در اجرای همروند تراکنش ها مکانیزیمی شبیه **زمانبند** ترتیبی خواهیم داشت.
- برای اجرای n تراکنش n! **زمانبند** خطی امکان پذیر است.
- یک اجرای همروند از تراکنش ها درست است اگر معادل با یکی از اجراهای پی در پی آنها باشد. در این صورت می گوییم این اجرای همروند پی در پی پذیر است.
- دو روش اصلی پی در پی پذیری عبارتند از :

✓Conflict Serializable

✓View Serializable

پی در پی پذیری (ترتیب پذیری)

ترتیب پذیری بیان صحت کنترل همروندی در سیستم های پایگاه داده هاست.

مزایای ترتیب پذیری:

- سیستم پایگاه داده ای که که اجراهایش ترتیب پذیر باشد فهمش ساده تر است. از دیدکاربر چنین سیستم همانند یک پردازشگر پی در پی تراکنش ها به نظر می رسد.
- یک سیستم پایگاه داده که اجراهای پی در پی پذیر را تولید می کند از تداخلهای تغییرات گم شده و تحلیل های ناسازگار جلوگیری می کند.

مفهوم برخورد(تعارض) در دستورات تراکنش ها

- دستورات تراکنش :
  - کنترلی مانند begin, End, abort, commit
  - اجرایی مانند Write, Read (بی برخورد ، با برخورد)
- تعریف برخورد (conflict) :
  - دو دستورالعمل l,ابا هم برخورد دارند اگر هر دو روی یک مورد اطلاعاتی عمل کنند و حداقل یکی از آنها دستور Write باشد.
- دستورالعمل هایی که برخورد ندارند :
  - دو دستورالعملی که به یک تراکنش تعلق نداشته باشند و روی یک مورد اطلاعاتی کار نکنند.
  - اگر بر روی یک مورد اطلاعاتی کار می کنند هر دو عمل خواندن را انجام دهند.

مفهوم برخورد(تعارض) در دستورات تراکنش ها

- اگر دو دستورالعمل (از یک زمانبند) با هم برخورد نداشته باشند و از نظر زمانی متوالی باشند گوییم زمانبند قابلیت جابجایی دارد.

	Read (Q)	Write (Q)
Read (Q)	بی برخورد	برخورد دار
Write (Q)	برخورد دار	برخورد دار

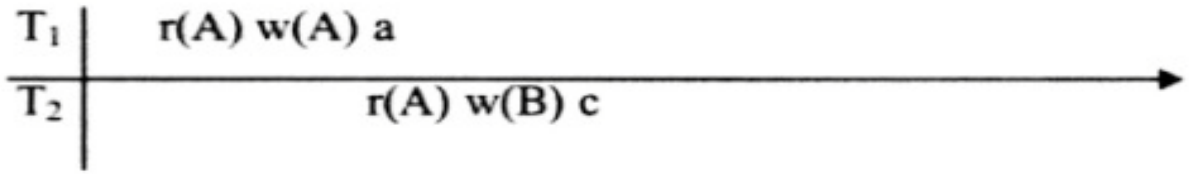
پی در پی پذیری در برخورد (Conflict Serializability)

- اگر زمانبند S بتواند با جابجا کردن دستورهای بدون برخورد به یک زمانبند ای مانند S’ تبدیل شودآن ها را معادل دربرخورد (conflict equivalent) می گویند.
- پی در پی پذیری در برخورد (Conflict Serializability):

زمانبندSرا پی در پی پذیری در برخورد گویند اگر معادل در برخورد با یک زمانبند خطی یا پی در پی (ترتیب پذیر) باشد.

یک زمانبندی پی‌درپی

$S_1 : r_1(A)w_1(A)a_1w_2(A)w_2(B)c_2$



ترتیب اجرای پی‌درپی آنها T_1 و سپس T_2 می‌باشد و می‌نویسیم:

$S_1 : T_1 < T_2$

ترتیب پذیری در برخورد (مثال)

- زمانبند S3 می تواند به S6 تبدیل شود. یک زمانبند پی در پی (ترتیبی) وقتی که تراکنش T2 به دنبال T1 باشد، با تعویض دستورات بدون برخورد به دست می آید.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

مثال از زمانبند ترتیب ناپذیر در برخورد

مثالی از **زمانبند** ای که ترتیب پذیردر برخورد نیست.

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

- در **زمانبند** بالا قادر به تعویض دستورات برای رسیدن به یک **زمانبند** ترتیبی < T3, T4 > یا < T4, T3 > نخواهیم بود.
- توجه: ممکن است **زمانبند** ای ترتیب پذیر باشد اما ترتیب پذیر در برخورد نباشد. به بیان دیگر لازمه ترتیب پذیری یک **زمانبند** ترتیب پذیری در برخورد نیست.

پی در پی پذیری دردید

- پی در پی پذیری در دید نوع دیگری از زمانبند های معادل با پی در پی پذیری است.
- زمانبندی های معادل در دید :
  - زمانبندهای S و S’ را معادل در دید (هم ارزی دیدی) گویند اگر داشته باشیم :
  - اگر تراکنش Ti مقدار Q را در زمانبند Sبخواند، تراکنش Ti نیز همان مقدار را در زمانبندS’ بخواند.
  - برای هر داده Q ، اگر تراکنش Ti در زمانبند S داده Q را از تراکنش Tj می خواند در زمانبند S’ نیز باید داده Q را از Tj بخواند.
  - برای هر داده Q ،آخرین تراکنشی از زمانبندی S که روی Q می نویسد، همان تراکنشی باشد که در زمانبندی S’ آخرین بار روی Q می نویسد.
  - همان طور که دیده شد هم‌ارزی دید با دستورات Read, write کار دارد.
  - با اعمال این شرط ها تضمین می کنیم که هر تراکنش در هر دو زمانبندی مقادیر یکسانی را بخواند. ثانیا با تضمین یکسان بودن مقادیر نوشته شده روی هر داده، اطمینان حاصل می شود که وضعیت نهایی بانک اطلاعات در هردو زمانبندی یکسان است.

معادل (هم ارزی) دردید(مثال)

- **زمانبند** های زیر معادل در دید نیستند. چرا؟

Schedule ۱

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

T_1	T_2
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	

مقدار A که توسط T2 خوانده شده بوسیله T1 تولید شده
درصورتی که در برنامه ۲ چنین نیست.

هم ارزی دردید(مثال)

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 1

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 3

پی در پی پذیردردید (توالی پذیر نمایی)

- زمانبند S را پی در پی پذیر دردید گویند اگر معادل در دید با یک زمانبند ترتیبی باشد.
- هر زمانبند پی در پی پذیر در برخورد نیز پی در پی پذیر در دید است.
- مثال: از زمانبند ای که ترتیب پذیر دردید هست ولی ترتیب پذیر در برخورد نیست. هم ارزی در دید دارد.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

- این زمانبند ترتیب پذیر در دید است زیرا معادل با زمانبند ترتیبی (T_{27},T_{28},T_{29}) است.

ترتیب پذیردردید

- هر زمانبند ترتیب پذیر در برخورد، ترتیب پذیر در دید هم هست. البته عکس این مطلب لزوما درست نیست.
- اما زمانبند ترتیب پذیر در دید که ترتیب پذیر در برخورد نیست دارای نوشتن کورکورانه است .
- نوشتن کورکورانه(blind writes): اگر در یک برنامه یک تراکنش تنها دارای دستور write باشد و هیچ دستور read دیگری نداشته باشد.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

نکته دیگر در مورد ترتیب پذیری

- در **زمانبند** زیر نتایج یکسانی با برنامه ترتیبی $< T_1, T_5 >$ تولید می کند ولی نه ترتیب پذیری در برخورد دارد و نه ترتیب پذیری در دید.

A= 1000 ,
B=2000

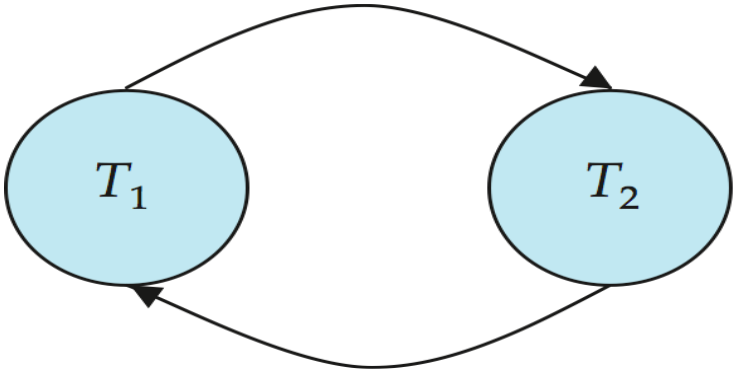
A= 960
B=2040

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)

Final State Serializable

گراف تقدم (Precedence graph)

- فرض کنید زمانبند دارای مجموعه ای از تراکنش های $T_1, T_2, ..., T_n$ است.
- **گراف تقدم** (Precedence graph) ، یک گراف جهت دار است که رئوس آن تراکنش ها هستند.
- از تراکنش T_i به T_j یک لبه جهت دار رسم می شود اگر دو تراکنش برخورد داشته و T_i به داده هایی که در آن برخورد وجود دارد زود تر دسترسی دارد.
- حال دو تراکنش T_1 و T_2را در نظر بگیرید اگر هم T_1بر T_2 تقدم داشته باشد وهم T_2 بر T_1(یعنی وجود حلقه) در این صورت دستور های برخورددار آنها را نمی توان پس و پیش کرد یعنی پی در پی پذیر نیستند.



تشخیص ترتیب پذیری با گراف تقدم

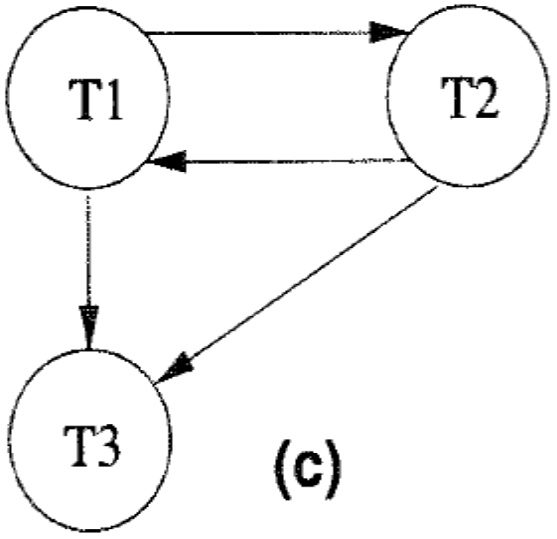
- از Ti به Tj خطی جهت دار رسم می شود اگر یکی از شرایط زیر رخ دهد:

۱. Ti دستور Write(Q) را اجرا کند قبل ازاینکه Tj دستور Read(Q) را اجرا کند.

۲. Ti دستور Read(Q) را اجرا کند قبل ازاینکه Tj دستور Write(Q) را اجرا کند.

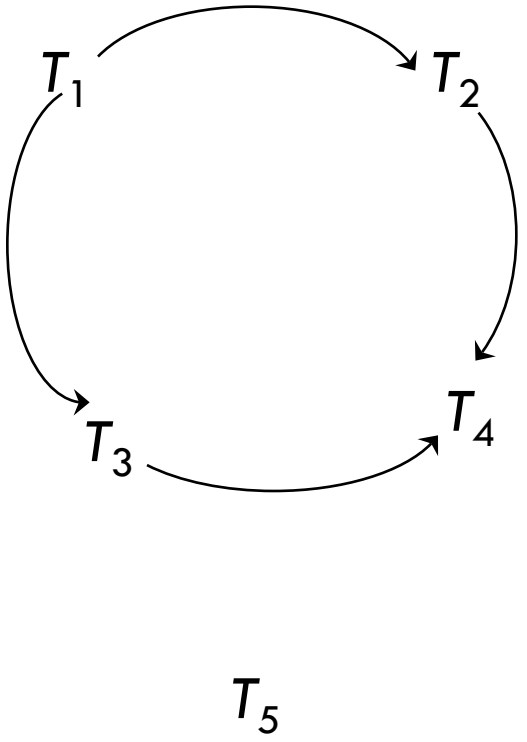
۳. Ti دستور Write(Q) را اجرا کند قبل ازاینکه Tj دستور Write(Q) را اجرا کند.

T1	T2	T3
R(A)		
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit



گراف تقدم

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

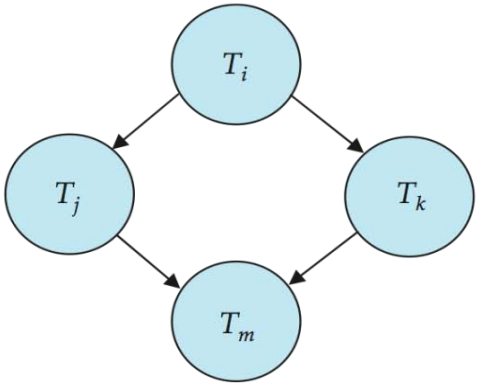


قضیه بنیادی در تئوری ترتیب پذیری

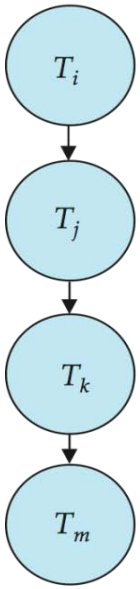
- قضیه: فرض کنیم T مجموعه ای از تراکنش ها باشد، زمانبند S را ترتیب پذیر در برخورد گویند اگر و تنها اگر در گراف تقدم آن دور وجود نداشته باشد.
- برای پیدا کردن ترتیب اجرای تراکنش ها در زمانبند ترتیب پذیر معادل که به آن اصطلاحا پی در پی پذیری (ترتیب پذیری) گفته می شود از روش **مرتب سازی توپولوژیکی** استفاده می شود.
- هر گاه در گراف تقدم زمانبند S، یال $T_i \rightarrow T_j$ وجود داشته باشد این یال باید در گراف تقدم زمانبند s' که زمانبند ترتیب پذیر معادل S است نیز وجود داشته باشد. در این صورت یک ترتیب خطی از تراکنش ها برای زمانبند ترتیبی معادل بدست می آید.

مثال

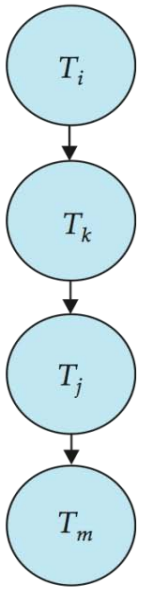
- اگر گراف تقدم زمانبند s چنین باشد. دو ترتیب خطی زیر را داریم:



(a)



(b)



(c)

- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.
  - (Better algorithms take order n + e where e is the number of edges.)
- This is a linear order consistent with the partial order of the graph.

تشخیص ترتیب پذیری در دید

- گراف تقدم بدست آمده درتشخیص ترتیب پذیری در برخورد نمی تواند برای تشخیص ترتیب پذیری در دید به کار رود.
  - توسعه آن برای تشخیص ترتیب پذیری در دید هزینه ای نمایی دارد.
  - مساله تشخیص ترتیب پذیری در دید یک مساله NP-Complete است و یافتن الگوریتم کارا برای حل این مساله اگر نه ناممکن بسیار نامحتمل است.
  - برای تشخیص ترتیب پذیری در دید از گرافی به نام پلی گراف استفاده می شود.

زمانبندهای ترمیم پذیر (Recoverable)

- اگر تراکنش T' یک مورد داده ای را بخواندکه قبلا توسط تراکنش T نوشته شده است و عمل commit در T قبل از Commit در T' باشد در اینصورت زمانبند را ترمیم پذیر گویند.

- The following schedule (Schedule 11) is not recoverable if T_9 commits immediately after the read

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

زمانبندهای ترمیم پذیر (Recoverable)

- زمانبندی را ترمیم پذیر (Recoverable - RC) گوییم اگر برای تمام T_jها که از T_i می خوانند، تثبیت تراکنش T_i قبل از تثبیت تراکنش T_j صورت گیرد.
- به عبارت دیگر اجرایی است که در آن یک تراکنش زمانی Commit می کند که تمامی تراکنش هایی که از آنها خوانده است(اثر پذیرفته است)، Commitکرده باشد.
- اجرای ترمیم پذیر Commit را به تعویف می اندازد. و یک تراکنش زمانی Commit میکند که همه قبلی ها که از آنها خوانده است Commit کنند و زمانی Abort می کند که حداقل یکی از آنهایی که خوانده است Abortکند.
- به این دلیل اجرای ترمیم پذیر گویند چون تضمین می کند، مقادیر نهایی شده در پایگاه داده ماندگارند وپایگاه داده شامل اثر هیچ تراکنش نهایی نشده ای نیست که از طریق به تعویق انداختن Commit حاصل می شود.

پدیده سقوط های آبشاری(Cascading Aborts)

- زمانبندی ترمیم پذیر ممکن است سبب ساقط شدن تراکنش های دیگر یه صورا آبشاری (Cascading Abort) گردد.این مشکل می تواند سبب از دست رفتن حجم قابل توجهی از کارهایی که تا بحال انجام شده اند گردد. در مثال زیر اگر T_{10}دچار خرابی شود T_{11} و T_{12} نیز مجبور به سقوط خواهند شد.

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

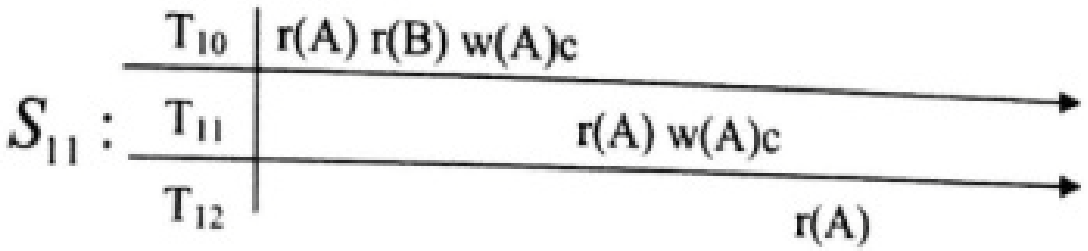
- پدیده انتشار سقط نامطلوب است چون :

۱ – مصرف بیهوده منابع ۲ – کلی باید منبع هدر داد تا بفهمیم کی از کی خوانده است(برای Abrt کردنشان)

۳– تعداد زیادی تراکنش باید سقط شوند.

زمانبند فاقد سقوط های آبشاری(Avoiding Cascading Aborts)

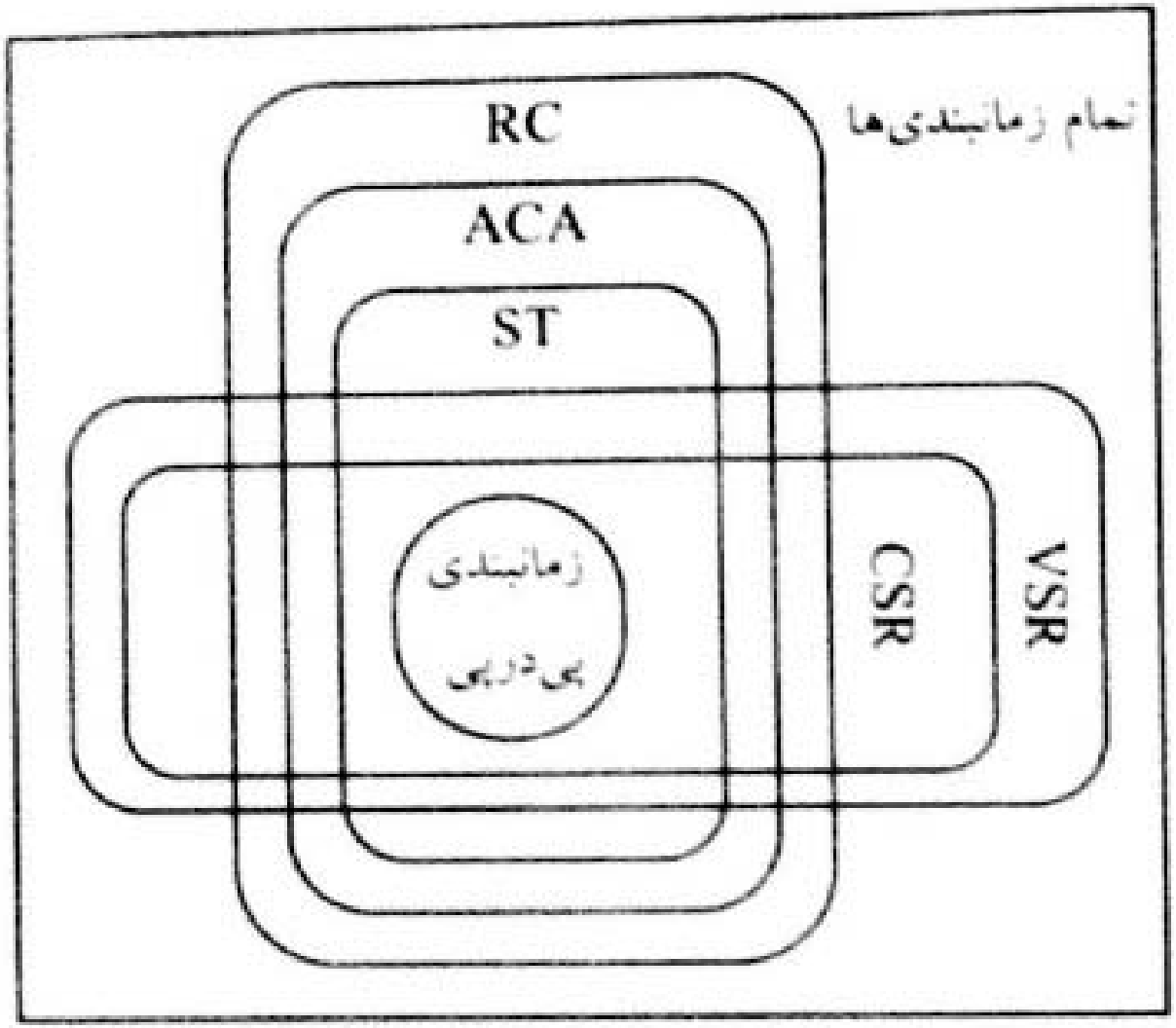
- زمانبند فاقد سقوط های آبشاری(ACA):
- گوییم یک سیستم پایگاه داده از انتشار ساقط شدن جلوگیری می کند اگر هر تراکنش تنها داده ای را بخواند که توسط تراکنش های نهایی(commit) شده نوشته شده باشد.
- هر زمانبند (برنامه) ACA قابل ترمیم نیز هست.
- خواندن از بعد از Commit یعنی فقط اثر تراکنش های نهایی شده خوانده شود یعنی به تعویق انداختن Read



زمانبند محض یا سخت گیر(Strict - ST)

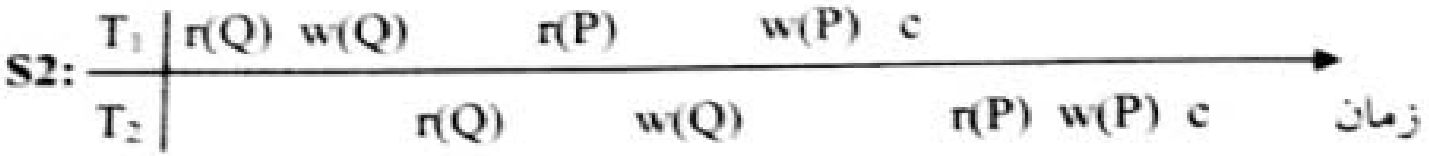
- زمانبندی را محض یا سختگیر(ST) گوییم چنانچه برای هر دو تراکنش T_i و T_j ،اگر T_i داده ای را پس از نوشتن T_i، بخواند یا بنویسد ، این عمل T_j بعد از خاتمه(تثبیت یا سقوط) T_i **اجرا شود. به عبارت دیگر زمانبندی محض اجازه خواندن یا نوشتن هیچ داده ای را نمی دهد مگر آنکه تراکنشی که روی آن داده نوشته است خلتمه یلفته باشد.**
- هر زمانبند محض(ST) زمانبند ACA نیز هست.
- اجرای محض اجراهایی هستند که عمل Read وWrite را به تعویق می اندازد و تنها بر روی قلم داده ای می نویسد که اثر تراکنش نهایی شده باشد و تنها از قلم داده ای می خواند که اثر تراکنش نهایی شده باشد.

رابطه بین تمام زمانبندی ها

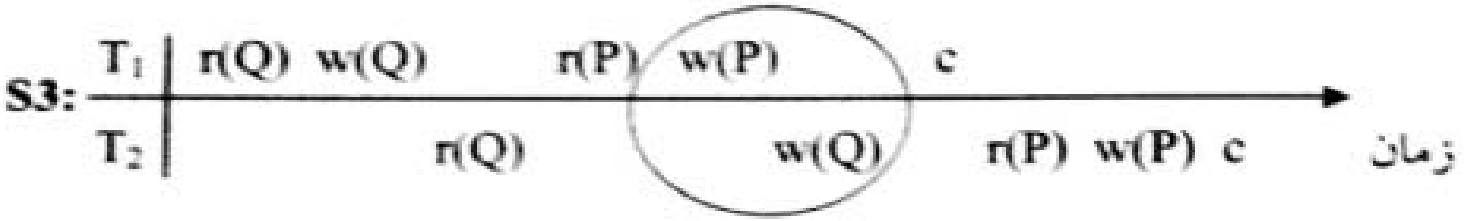


مثال هایی از زمانبندی ها

مثال: "معادل در برخورد" با زمانبندی S_2 زیر را بیابید.



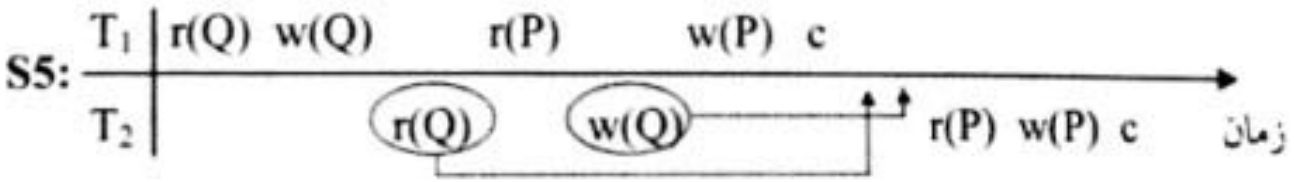
حل:



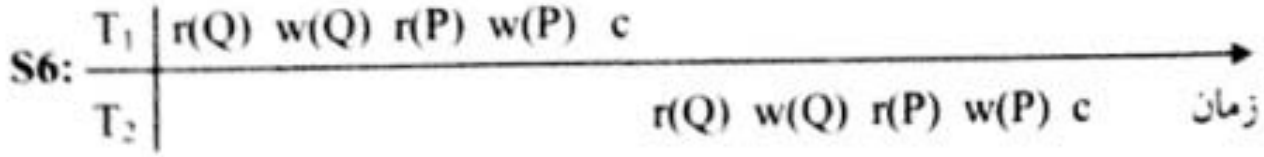
در این مثال فقط ترتیب زمانی $w_1(P)$ و $w_2(Q)$ عوض شده. هر چند هر دو دستور w() هستند ولی برخورد ندارند چون روی دو داده مختلف عمل می‌کنند.

مثال هایی از زمانبندی ها

مثال:

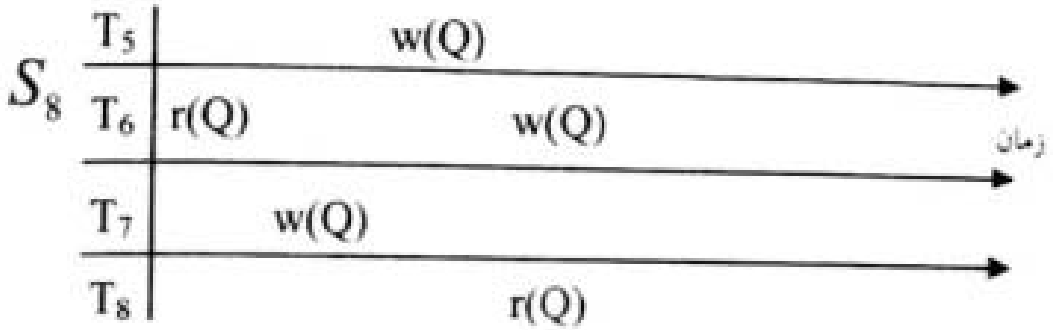


دستور $w_2(Q)$ را می‌توان از $w_1(P)$ و c_1 گذراند زیرا با آنها برخورد ندارد. همچنین می‌توان دستور $r_2(Q)$ را از $r_1(P)$، $w_1(P)$ و c_1 گذراند. حاصل این عملها، زمانبندی S_6 را به دست می‌دهدکه پی‌درپی و معادل S_5 است.

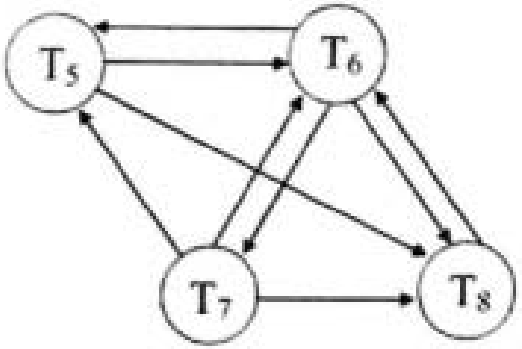


S_6 معادل اجرای پی‌درپی $T_1 < T_2$ می‌باشد پس می‌توان گفت S_5 و S_6 زمانبندهای پی‌درپی پذیر در برخورد هستند.

مثال هایی از زمانبندی ها



حل:



در ایـن زمانبنـدی سـاده چنـد حلقـه وجـود دارد ماننـد (T5 ,T7 ,T6 ,T8 ,T5) و (T6 ,T7 ,T6). می‌توان تشخیص داد که اگر تراکنش T6 ساقط شود همه حلقـه‌هـا از بین می‌روند و زمانبندی پی‌درپی پذیر $T_7 < T_5 < T_8$ به دست می‌آیـد.