پایگاه داده System concepts

Chapter 14: Transactions

نام استاد: آقای لاله پرور

مترجم: مینا رزمی

Chapter 14: transactions

- مفهوم تراکنش
- وضعیت تراکنش
- اجرای همروند تراکنش
- قابلیت سریالی شدن
- ترمیم پذیری
- پیاده سازی جدایی
- تعریف تراکنش در SQL
- آزمایش کردن قابلیت سریالی شدن

Transaction Concept

- تراکنش واحدی از اجرای برنامه می باشد که به آیتم‌های داده‌ای مختلف دسترسی دارد (آنها را می‌خواند) و در صورت امکان آنها را بروز می نماید. (واحد یعنی چیزی که حالت تجزیه‌ناپذیر دارد؛ یا انجام میشود یا انجام نمی شود.)

- دو مسئله اصلی که در زمینه تراکنش‌ها با آنها سروکار داریم عبارتند از:
  - وقوع انواع مختلفی از خرابی، مانند خرابی سخت افزاری و توقف عملیات سیستم
  - اجرای همروند چند تراکنش

Example of Fund Transfer

- **تراکنش برای انتقال $50 از حساب A به حساب B:**

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

- **نیازمندی تجزیه ناپذیری**

- اگر تراکنشی بعد از مرحله 3 و قبل از مرحله 6 دچار شکست شود پول گم خواهد شد و در نتیجه وضعیت پایگاه داده ناپایدار می‌شود
  - خرابی می تواند علت سخت افزاری یا نرم افزاری داشته باشد.

- سیستم باید تضمین کند که بروزرسانی‌های تراکنشی که قسمتی از آن اجرا شده در پایگاه داده منعکس نمی‌شود

در واقع هر چند تراکنش دنباله ای از عملیات می باشد اما باید طوری آن را اجرا نماییم که نتیجه به صورت تجزیه ناپذیر به نظر بیاید؛ یعنی یا اجرا شود یا اجرا نشود. در مثال بالا که بعد از عمل شماره 3 خرابی رخ داد، عملا طبق این نیازمندی Atomicity باید مقدار حساب A را نیزباید اصلاح نماییم ، زیرا عمل مربوط به حساب B، انجام نشده پس A را نیز نباید انجام دهیم (یا همه یا هیچ). به عبارت دیگر سیستم باید تضمین نماید که اجرای نیمه کاره نداریم؛ یا تراکنش باید کامل انجام شود یا اصلا اجرا نشود.)

- **نیازمندی دوام**

به محض اینکه به کاربر اطلاع داده شد که تراکنش کامل شده است (مثلاً انتقال وجه با موفقیت انجام شد)، بروزرسانی‌هایی که توسط تراکنش روی پایگاه داده انجام شده بایستی پایدار بمانند، حتی اگر شکست نرم افزاری و سخت افزاری وجود داشته باشد. به عبارت دیگر نیازمندی دوام این خصوصیت را بیان می کند که اگر ی تراکنش با موفقیت نجام شد، بایستی نتیجه آن حفظ شود.)

Example of Fund Transfer (cont.)

- **تراکنش برای انتقال $50 از حساب A به حساب B:**

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

نیازمندی سازگاری

- با اجرای تراکنش، مجموع B و A در مثال تغییری نمی‌کند.
- در مجموع نیازمندی سازگاری شامل موارد زیر است:

- محدودیت‌های (قواعد) جامعیت که به صورت صریح مشخص شده‌اند
  - مانند کلیدهای اصلی و کلیدهای خارجی
- قواعد جامعیت ضمنی (غیر صریح)
  - مثلاً مجموع موجودی کلیه حساب‌ها منهای مجموع مقادیر مربوط به وام‌ها باید برابر مقدار دارایی پولی شخص باشد.
- تراکنش باید پایگاه داده را در وضعیت سازگار ببیند.
- در حین اجرای تراکنش ممکن است پایگاه داده به طور موقت در وضعیت ناسازگار قرار بگیرد.
- هنگامی که اجرای تراکنش با موفقیت کامل می شود، پایگاه داده باید در وضعیت سازگار باشد.
- منطق نادرست یک تراکنش می تواند منجر به ناسازگاری شود.

Example of Fund Transfer (cont.)

- **نیازمندی جدایی**

اگر بین مرحله 6 و 3 تراکنش دیگری مانند T2 اجازه دسترسی به پایگاه داده‌ای که به صورت ناقص بروزرسانی شده را داشته باشد، این تراکنش پایگاه داده را در یک وضعیت ناسازگار خواهد دید (مجموع B و A کمتر از آن چیزی است که باید باشد.)

T1	T2
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	

- جدایی می تواند به صورت خام و ابتدایی با اجرای تراکنش ها به صورت سریال (پشت سر هم) تضمین شود.
- اجرای چند تراکنش به صورت یکجا دارای مزایای مهمی است که در ادامه خواهیم دید.

ACID Properties

، تراکنش واحدی از اجرای برنامه است که به آیتم‌های داده‌ای مختلف دسترسی پیدا می‌کند و در صورت امکان آنها بروزرسانی می نماید. سیستم پایگاه داده باید برای حفظ جامعیت داده‌ها موارد زیر را تضمین نماید:

- تجزیه ناپذیری: یا تمام عملیات تراکنش به طور صحیح در پایگاه داده منعکس می شود یا هیچ یک از آنها
- سازگاری: اجرای یک تراکنش به صورت جدا (ایزوله)، سازگاری پایگاه داده را حفظ می‌کند.

- جدایی: اگر چه چند تراکنش ممکن است به صورت همروند اجرا شوند، هر تراکنش باید بی خبر از اجرای همروند تراکنش های دیگر باشد. نتیجه‌های میانی تراکنش‌ها باید پنهان از مابقی تراکنش هایی باشد که همروند اجرا می‌شوند.
  - برای هر جفت تراکنش مانند T_i , T_j ، برای T_i به نظر می‌رسد که یا T_j قبل از شروع اجرای T_i به پایان رسیده یا اجرای T_j بعد از پایان اجرای T_i شروع شده است.
- دوام: پس از اینکه اجرای تراکنش با موفقیت انجام شد تغییراتی که در پایگاه داده ایجاد می شود، باید حفظ شود؛ حتی اگر در سیستم خرابی به وجود بیاید.

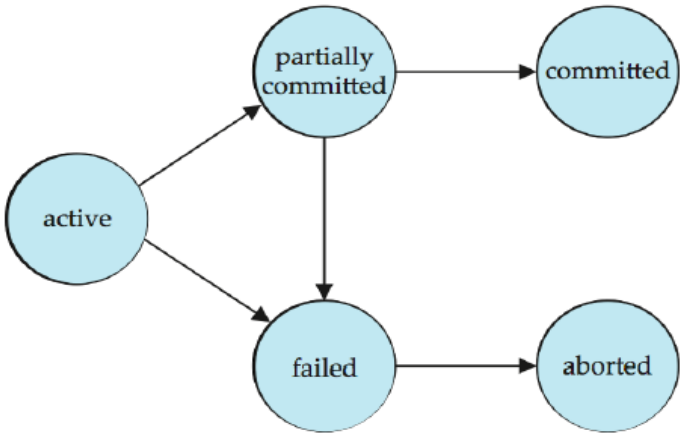
Transaction State

- Active (فعال): وضعیت اولیه؛ تا زمانی که تراکنش در حال اجراست در این وضعیت می ماند.

- Partially Committed (موفق شده به صورت جزئی): بعد از اجرای آخرین دستور
- Failed (دچار شکست شده): بعد از کشف اینکه اجرای نرمال نمی تواند بیش از این پیش برود
- Aborted (سقط شده): بعد از اینکه تراکنش عقبگرد (Rollback) کرد و پایگاه داده به حالتی که قبل از آغاز تراکنش داشت برگشت.
  - دو گزینه بعد از سقط شدن یک تراکنش وجود دارد:
    1- راه اندازی مجدد تراکنش که تنها در صورت عدم وجود خطای منطقی درونی می‌تواند انجام شود
    2- از بین بردن (کشتن) تراکنش.

- Committed (موفق شده): پس از تکمیل اجرا با موفقیت

Transaction State (cont.)



Concurrent Executions

- چند تراکنش می‌توانند در سیستم به صورت همروند اجرا شوند.

مزایای آن عبارت است از:

- افزایش بهره‌وری پردازنده و دیسک: که منجر به بهبود قابلیت گذردهی تراکنش می‌شود.
  - مثلاً یک تراکنش می تواند از cpu استفاده کند، در حالی که تراکنش دیگری در حال خواندن از یا نوشتن روی دیسک است.

- کاهش میانگین زمان پاسخگویی به تراکنش ها: تراکنش های کوتاه نیازی به صبر کردن برای اتمام کار تراکنش های طولانی ندارند.

- شِماهای کنترل همروندی: مکانیزم هایی برای دست یافتن به جدایی

- کنترل کردن تعاملات میان تراکنش های همروند به منظور پیشگیری از نابود کردن سازگاری پایگاه داده
- در فصل بعد بررسی خواهد شد

Schedules

- زمانبندی عبارت است از دنباله ای از دستورالعمل ها که یک ترتیب زمانی برای اجرای دستورات تراکنش-های همروند را مشخص می‌کند.
  - یک زمانبندی برای تراکنش ها باید شامل همه دستورالعمل های آن تراکنش ها باشد.
  - یک زمانبندی برای تراکنش ها باید ترتیبی را که بر اساس آن دستورالعمل ها در هر یک از تراکنش‌ها ظاهر می‌شوند را حفظ کند.
- تراکنشی که اجرایش با موفقیت به اتمام می رسد دارای دستورالعمل commit به عنوان آخرین دستور می باشد.(منظور اینکه آخرین دستور تراکنش commit می باشد.)
  - به صورت معمول فرض می‌شود که تراکنش دستورالعمل commit را به عنوان آخرین مرحله خود اجرا کند.
  - تراکنشی که اجرای موفقش با شکست روبرو می‌شود، یک دستور Abort را به عنوان آخرین دستور دارد.

Schedule 1

- اجازه دهید T1 ، $50 از حساب A به حساب B انتقال دهد
- زمانبندی سریال

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

- زمانبندی سریال

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	

Schedule 3

- این زمانبندی سریال نما باشد اما با زمانبندی 1 برابر می باشد.
- در زمانبندی 1،2،3 مجموع A+B مقدارشان حفظ شده است.

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 4

- زمانبندی 4 همروند می باشد و مقدار A+B را حفظ نمی کند.

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

serializability

- فرض اولیه: هر تراکنشی سازگاری پایگاه داده را حفظ می کند.
- در نتیجه اجرای سریال مجموعه‌ای از تراکنش ها سازگاری پایگاه داده را حفظ می کند.
- یک زمانبندی (احتمالاً همروند) قابل سریال شدن است اگر هم ارز و معادل با یک زمانبندی سریال باشد.

صورت‌های مختلف برای «معادل بودن زمانبندی‌ها» موجب ظهور مفاهیم زیر می شود:

1- conflict serializability (قابلیت سریالی شدن از نظر تعارض)

2- view serializability (قابلیت سریالی شدن از نظر نما)

Simplified view of transactions

- عملیات‌هایی به جز دستورالعمل های خواندن و نوشتن را نادیده می گیریم.
- فرض می کنیم که ممکن است تراکنش ها ما بین عمل خواندن و نوشتن یکسری محاسبات دلخواه بر روی داده های واقع در بافرهای محلی انجام دهند.
- زمانبندی های ساده شده، تنها شامل دستور العمل های خواندن و نوشتن هستند.

Conflict Instructions

- دستورالعمل های l_i ,l_j به ترتیب از تراکنش های T_j و T_I با هم متعارضند اگر و تنها اگر l_i ,l_j هر دو به آیتم Qدسترسی داشته باشند و حداقل یکی از این دستورالعمل ها Q را نوشته باشد.
- اگر l_i ,l_j در زمانبندی پشت سرهم بوده و با هم متعارض نباشند، حتی اگر جای l_i ,l_j در زمانبندی تعویض شود نتیجه آن ها یکسان باقی می ماند.

Conflict Serializability

- اگر زمانبندی s با جابجایی دستورالعمل‌های نامتعارض بتواند به زمانبندی Ś تبدیل شود، آن وقت می گوییم که S و Ś از نظر تعارض معادل هستند.
- اگر S از نظر تعارض معادل با یک زمانبندی سریال باشد، می گوییم زمانبندی S قابلیت سریالی شدن تعارضی دارد.

Conflict Serializability(cont.)

- زمانبندی 3 می تواند با جابجایی دستورات نامتعارض آن به زمانبندی 6 (که یک زمانبندی سریال شامل T1 و T2 است) تبدیل شود. بنابراین زمانبندی 3 قابلیت سریالی شدن تعارضی دارد.

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

Conflict Serializability(cont.)

- مثالی از زمانبندی ای که قابلیت سریالی شدن تعارضی ندارد.

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

- ما قادر به جابجا کردن دستورالعمل ها در زمانبندی بالا برای بدست آوردن زمانبندی سریال <T3 , T4> یا <T4, T3> نیستیم.

View Serializability

- فرض کنید S و Ś دو زمانبندی با مجموعه ای از تراکنش های یکسان باشند. S و Ś قابلیت سریالی شدن از نظر نما را دارند اگر این سه شرط برای هر آیتم مانند Q صدق نماید:
  1- اگر در زمانبندی S، تراکنش T_I مقدار اولیه Q را بخواند ، سپس در زمانبندی Ś همچنین تراکنش T_I باید مقدار اولیه Q را بخواند.
  2- اگر در زمانبندی S تراکنش T_I ، read(Q) را اجرا نماید، و آن مقدار توسط تراکنش T_J تولید شده باشد ، بنابراین در زمانبندی Ś همچنین تراکنش T_I باید مقدار Q را که توسط همان write(Q) عملیات تراکنش T_J تولید شده است ، بخواند .
  3- اگر هر تراکنشی که عملیات write(Q) نهایی را در زمانبندی S انجام می دهد ،در زمانبندی Ś هم باید این عمل انجام شود .

View Serializability(cont.)

- زمانبندی S ، اگر با یک زمانبندی سریال معادل از نظر نما باشد، در این صورت زمانبندی S، قابلیت سریالی شدن از نظر نما را دارد.
- هر زمانبندی که قابل سریالی شدن از نظر تعارض باشد، قابل سریالی شدن از نظر نما هم می باشد.
- هر زمانبندی با قابلیت سریالی شدن از نظر نما که قابل سریالی شدن از نظر تعارض نیست، دارای نوشتن کورکورانه (یعنی نوشتن آیتمی قبل از خواندن آن) می باشد.
- زمانبندی زیر قابل سریالی شدن از نظر سریالی شدن از نظر نما می باشد اما قابل سریالی شدن از نظر تعارض نمی باشد.

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

Other notions of serializability

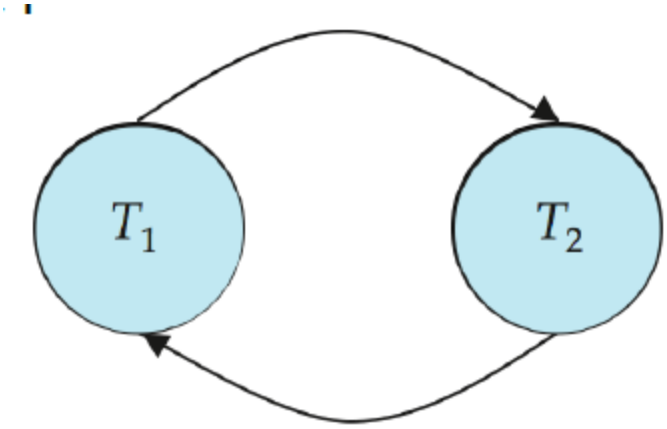
- زمانبندی زیر همان نتیجه ای را می دهد که زمانبندی سریال <T1, T5>، با این حال نه قابلیت سریالی شدن از نظر نما دارد و نه قابلیت سریالی شدن از نظر تعارض.
- تعیین کردن چنین برابری هایی نیازمند آنالیز کردن عملیاتی به جز read, write می باشد.

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)

Testing for Serializability

- یک زمانبندی دلخواه شامل تراکنش های T1,T2,....,Tn را در نظر بگیرید.
- **Precedence graph**: (گراف تقدم): گرافی جهت‌دار است که در آن راس ها تراکنش ها هستند.

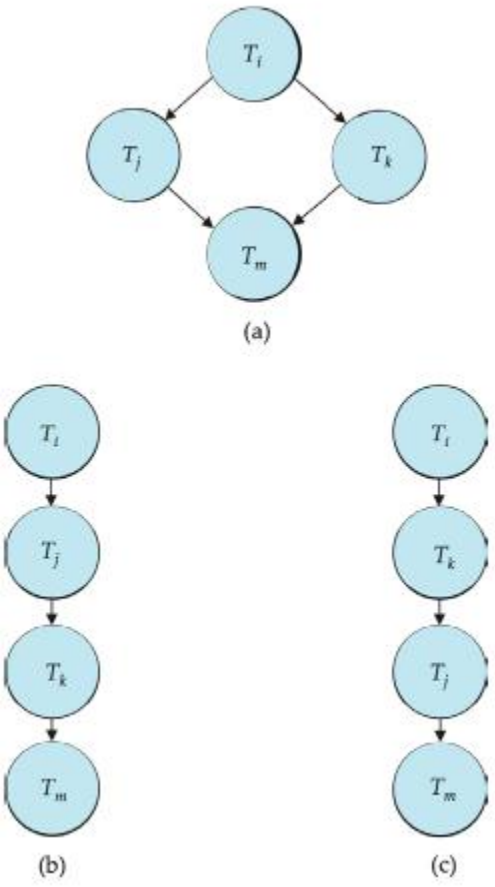
- یک کمان از T_I به T_J رسم می کنیم اگر دو تراکنش با هم متعارض باشند و T_I زودتر به ایتم داده‌ای که تعارض روی آن رخ داده دسترسی پیدا کند.



Testing for Conflict Serializability

- یک زمانبندی قابلیت سریالی شدن تعارضی دارد اگر و تنها اگر گراف تقدم آن دور نداشته باشد.
- الگوریتم‌هایی برای پیدا کردن دور وجود دارد که از مرتبه زمانی n^2 می باشند که در آنها n تعداد رئوس در گراف می باشد.
- اگر گراف تقدم دور نداشته باشد، ترتیب اجرای تراکنش‌های سریال می تواند توسط مرتب سازی Topological گراف بدست آید.
  - این یک مرتب سازی خطی سازگار با ترتیب جزئی گراف است.
  - برای مثال، ترتیب اجرای سریال برای زمانبندی A به صورت زیر است:

T5→ T1→ T3→ T2 →T4



Test for view Serializability

- تست گراف تقدم برای قابلیت سریالی شدن تعارضی نمی تواند به صورت مستقیم برای تست قابلیت سریالی شدن از نظر نما استفاده شود.
- مسئله بررسی اینکه یک زمانبندی «قابلت سریالی شدن از نظر نما» دارد یا خیر، در رده مسائل Np-complete قرار می‌گیرد.
- با این حا، الگوریتم های عملی که فقط شروط کافی برای قابلت سریالی شدن از نظر نما را بررسی می‌کنند را می‌توان مورد استفاده قرار داد.

Recoverable Schedules

- زمانبندی قابل ترمیم: اگر تراکنش T_J آیتم داده‌ای را که قبلا توسط تراکنش T_I نوشته شده بخواند، عمل commit کردن T_I قبل از عمل commit کردن T_J ظاهر شود.
- اگر T_9 فوراً بعد از خواندن، commit کند، زمانبندی T_{11} قابل ترمیم نیست.

(اگر commit داشته باشیم نمی توان به ابتدای برنامه عقبگرد کرد. اگر commit رخ دهد تراکنش غیر قابل ترمیم است. برای رفع چنین مشکلی باید سازوکاری را اعمال کرد که چون T_J به T_I وابسته است، اگر T_I دچار شکست شود T_Jهم باید عقبگرد کند ولی به علت وجود commit ، این اتفاق رخ نخواهد داد. پس باید commit T_I جلوتر از T_J باشد. به طور کلی باید زمانی مصرف کننده commit کند که تولیدکننده قبل از آن commit کرده باشد.

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

Cascading Rollbacks

- عقبگرد آبشاری: شکست یک تراکنش باعث عقبگرد در یکسری تراکنش می شود.

- در جدول صفحه 14.29 اگر T_{10} ، دچار شکست شود T_{11} و T_{12} نیز باید عقبگرد کنند.
(اگر T_{12} به T_{11} وابسته باشد و T_{11} به T_{10} وابسته باشد ، اگر T_{10} عقبگرد داشته باشد باید همه تراکنش های وابسته به آن هم عقبگرد داشته باشند.)

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

Cascadeless Schedule

- زمانبندی بدون عقبگرد آبشاری: زمانبندی است که در آن عقبگرد آبشاری نمی‌تواند رخ دهد. برای هر جفت تراکنش T_J و T_I که T_J آیتم پیشتر نوشته شده توسط T_I را می‌خواند، عمل commit شدن T_I قبل از عمل خواندن T_J ظاهر می شود.

- هر زمانبندی غیر آبشاری قابل ترمیم هم می باشد.
- این خوشایند است که زمانبندی ها را به آنهایی که غیرآبشاری هستند محدود کنیم.

Concurrency Control

- پایگاه داده باید مکانیزمی را فراهم آورد که تضمین کند کلیه زمانبندی های ممکن:
  - قابلیت سریالی شدن تعارضی یا از نظر نما داشته باشند.
  - قابل ترمیم و ترجیحاً بدون عقبگرد آبشاری باشند.
- این سیاست که در هر زمان تنها یک تراکنش اجرا شود، زمانبندی های سریال ایجاد می کند اما درجه ضعیفی از همروندی را فراهم می آورد.

- تست کردن زمانبندی برای قابلیت سریالی شدن بعد از اجرای آن کمی دیر است.

- هدف: توسعه پروتکل‌های کنترل همروندی که قابلیت سریالی شدن را تضمین می نمایند.

Concurrency Control(cont.)

- به منظور سازگاری پایگاه داده، زمانبندی‌ها باید قابل سریالی شدن، قابل ترمیم و ترجیحاً بدون عقبگرد آبشاری باشند.

- این سیاست که در هر زمان تنها یک تراکنش اجرا شود، زمانبندی های سریال ایجاد می کند اما درجه ضعیفی از همروندی را فراهم می آورد.
- شِماهای کنترل همروندی، میان میزان همروندی مجاز و مقدار سرباری که ایجاد می‌کنند، تعادل برقرار می-کنند.
- بعضی از شِماها تنها تولید زمانبندی های دارای قابلیت سریالی شدن تعارضی را مجاز می‌دانند؛ در حالی که بقیه، زمانبندی های دارای قابلیت سریالی شدن از نظر نما (که قابلیت سریالی شدن تعارضی ندارند) را اجازه می دهند.

Concurrency Control vs. serializability Tests

- پروتکل‌های کنترل همروندی، امکان استفاده از زمانبندی‌های همروند را می دهند، اما تضمین می کنند که این زمانبندی ها قابلیت سریالی شدن و ترمیم را دارند و فاقد عقبگرد آبشاری هستند.
- به طور کلی پروتکل‌های کنترل همروندی، گراف تقدم را در حین ایجاد آن بررسی نمی کنند.
  - در عوض، این پروتکل‌ها یک نظم و ترتیبی را اِعمال می‌کنند که از تولید زمانبندی‌های فاقد قابلیت سریالی شدن اجتناب شود.
- پروتکل‌های کنترل همروندی مختلف، میان میزان همروندی مجاز و مقدار سرباری که ایجاد می‌کنند، میزان متفاوتی از تعادل برقرار می‌کنند (یعنی برخی همروندی بالاتری ایجاد می‌کنند و برخی سربار کمتری).
  میزان overheadای که آنها متحمل می شوند را فراهم می کند.
- تست قابلیت سریالی شدن به ما کمک می کند تا متوجه شویم چرا یک پروتکل کنترل همروندی صحیح است.

Weak levels of Consistency

- برخی از برنامه‌های کاربردی مایلند با سطوح ضعیفی از سازگاری کار کنند و امکان استفاده از زمانبندی‌های فاقد قابلیت سریالی شدن را داشته باشند
  - مثل یک تراکنش فقط خواندنی که می خواهد مقداری تقریبی از کل موجودی تمام حساب‌ها را به دست آورد.
  - مثل داده‌های آماری پایگاه داده جهت بهینه‌سازی پرس و جوها که می توانند به صورت تقریبی محاسبه شوند.
  - چنین تراکنش هایی نیازی به داشتن قابلیت سریالی شدن نسبت به دیگر تراکنش‌های موجود در سیستم ندارند.
- تعادل بین دقت و کارایی (کاهش دقت به منظور افزایش کارایی)

Levels of Consistency in SQL

- قابل سریالی شدن – سطح پیش‌فرض
- خواندن تکرارپذیر: فقط رکورد هایی که commit شده اند می توانند خوانده شوند. خواندن های تکراری یک رکورد یکسان باید مقدار یکسانی را برگرداند. با این وجود، ممکن است تراکنش قابلیت سریالی شدن نداشته باشد. ممکن است تراکنش برخی از رکورد هایی که توسط یک تراکنش درج شده را پیدا کند اما بقیه را پیدا نکند
- خواندن commit شده: فقط رکوردهایی که commit شده اند می‌توانند خوانده شوند، اما خواندن های متوالی یک رکورد ممکن است مقادیر مختلفی (هر چند commit شده) را برگرداند.
- خواندن commit نشده: حتی رکوردهای commit نشده می‌توانند خوانده شوند.
- سطوح سازگاری پایین‌تر، برای جمع آوری اطلاعات تقریبی درباره پایگاه داده مفید هستند.
- هشدار: بعضی از سیستم‌های پایگاه داده، به صورت پیش‌فرض تضمین نمی‌کنند که زمانبندی‌ها قابلیت سریالی شدن داشته باشند.
مثلاً Oracle و PostgreSQL به صورت پیش‌فرض از سطحی از سازگاری که «جدایی تصویر لحظه‌ای» نام دارد پشتیبانی می‌کنند.

Transaction Definition in SQL

- زبان دستکاری داده‌ها باید دارای ساختاری برای تعیین کردن مجموعه اعمالی که با هم یک تراکنش را تشکیل می‌دهند، داشته باشد.

- در SQL یک تراکنش به طور ضمنی (غیر صریح) آغاز می شود.
- در SQL تراکنش توسط موارد زیر پایان می یابد:

  - Commit work تراکنش جاری را commit می‌کند و یک تراکنش جدید را آغاز می‌کند
  - Rollback work موجب می شود تا تراکنش جاری Abort شود

- تقریباً در کلیه سیستم های پایگاه داده، به صورت پیش فرض، هر دستور SQL که با موفقیت اجرا شود، به صورت ضمنی commit می‌شود.

  - قابلیت commit ضمنی را می‌توان با یک دستور پایگاه داده غیر فعال کرد.
    - مثلاً در JDBC به صورت connection.setAutoCommit(false) غیر فعال می‌شود.