دانشگاه آزاد اسلامی واحد نجف آباد
دانشکده کامپیوتر
گروه کارشناسی ارشد نرم افزار

درس پایگاه داده پیشرفته
(بخش اول)

مدرس :
دکتر محمد نادری دهکردی
E-mail: naderi@iaun.ac.ir
Weblog: adbms.blogfa.com

تشریح مفاهیم درس

- **تعداد واحد:** ۳

- **درس پیشنیاز و اطلاعات پیش زمینه مورد نیاز:** اصول طراحی پایگاه داده (مقطع کارشناسی)

- **رویکرد یادگیری:**
  - مطالب درسی در کلاس تدریس می شود.
  - به موازات آن دانشجو مقالات پژوهشی مرتبط را مورد بررسی قرار می دهد،
  - بر روی مسأله تحقیق، بحث و تجزیه تحلیل نموده
  - و گزارشات را به هر دو صورت شفاهی و کتبی ارائه خواهد کرد.
  - دانشجو موظف است حداقل یک گزارش کتبی فنی تحویل دهد که می تواند
  - یک مقاله مروری (Survey) و یا مقاله تحلیلی (Analysis) از تحقیقات موجود باشد.

اهداف اصلی در این درس

- مجهز نمودن دانشجو به دانش آخرین کارهای انجام شده در زمینه تحقیقاتی پایگاه داده
- تعلیم دانشجویان جهت توانایی آنالیز و نقد مقالات تحقیقاتی
- فراهم آوردن زمینه یادگیری مهارت مقاله نویسی فنی

- کلمات کلیدی درس:

- Internet/Web Database
- Mobile Database
- Concurrency Control
- Spatial/Temporal Database
- Multimedia Database
- Query Evaluation and Optimization
- Buffer Management
- Advanced Indexing Techniques
- Data Mining and Decision Support
- Data Privacy and Security
- Parallel and Distributed Database Systems
- Advanced Data Models

مراجع مطالعات برای پژوهش

- ACM Computing Surveys
- Communications of the ACM
- IEEE Computer
- Journal of the ACM
- Relevant ACM Transactions: Transactions on Database Systems
- Relevant IEEE Transactions: Transactions on Knowledge and Data Engineering
- Other relevant journals and magazines: Journal of VLDB, Journal of Distributed and Parallel Databases,
- Data and Knowledge Engineering
- Seminal papers in relevant conferences: SIGMOD, VLDB, PODS, ICDE, ICDT, EDBT, CIKM

مراجع کلاسیک درس

- An introduction to Database Systems Volume II, C. J. Date, IBM corporation.
- Distributed Databases, Ceri & Pelagatti.
- بانک اطلاعات علمی-کاربردی، جلد دوم: مفاهیم پیشرفته، دکتر مصطفی حق جو و علی اصغر صفائی.

فهرست مطالب

- معماری های مختلف سیستم پایگاه داده
  - معماری متمرکز
  - معماری نامتمرکز
  - معماری سیستم پایگاهی مشتری-خدمتگزار
  - معماری سیستم پایگاهی توزیع شده
  - معماری با پردازش موازی
  - معماری با حافظه مشترک
  - معماری با دیسکهای مشترک
  - معماری بی اجزاء مشترک
  - معماری سلسله مراتبی
  - معماری چند پایگاهی
  - سیستم پایگاهی همراه

فهرست مطالب

- بانک اطلاعات مبتنی بر شیئ و XML
  - بانک اطلاعات شیئ گرا
  - بانک اطلاعات شیئ-رابطه ای
  - بانک اطلاعات XML
- مدیریت تراکنش
  - مفاهیم
  - پروتکل های کنترل همروندی
  - مدیریت ترمیم
  - مبانی نظری مدیریت تراکنش

فهرست مطالب

- بانک اطلاعات نامتمرکز (توزیع شده)
- سیستم های نوین بانک اطلاعات و مدل های پیشرفته تراکنش
  - سیستم های نوین بانک اطلاعات
  - مدلهای پیشرفته تراکنش
- امنیت در بانک اطلاعات

انواع معماری سیستم های پایگاه داده

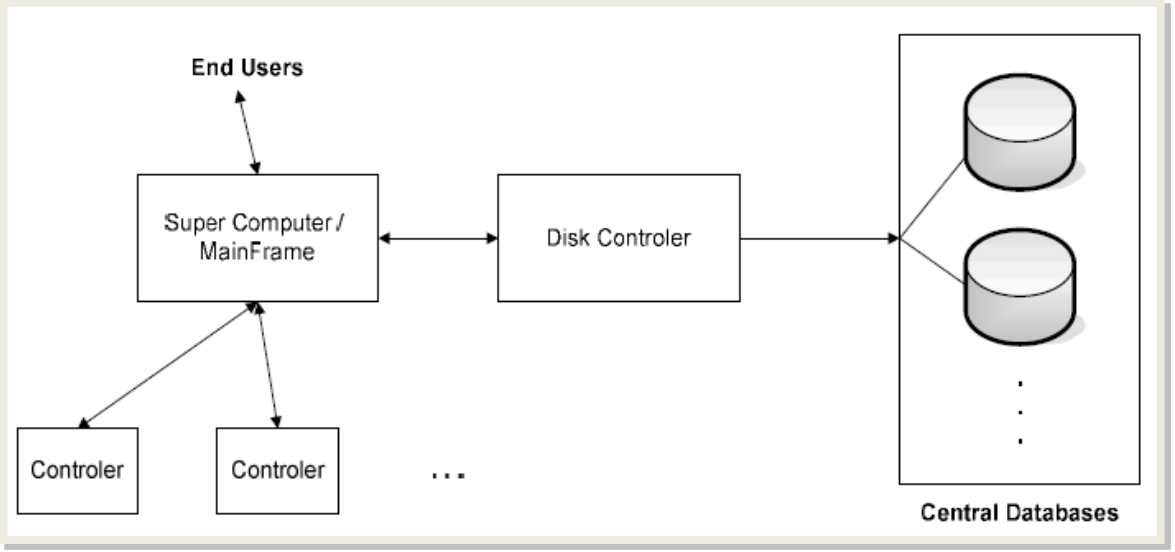
عوامل مؤثر در معماری پایگاه داده

- سخت افزار مدیریت پایگاه داده
- نرم افزار مدیریت پایگاه داده
- موقعیت جغرافیایی کاربران
- ماهیت پردازشها و تراکنش ها
- تعداد تراکنش ها
- حجم داده های ذخیره شدنی
- موقعیت مکانی داده ها و ارتباطات بین آنها
- ماهیت کاربردهای مورد نظر

انواع معماری (کلان)

- معماری متمرکز
- معماری نامتمرکز

معماری متمرکز



- یک پایگاه داده روی یک سیستم قرار دارد بدون ارتباط با سیستم های دیگر.
- روی کامپیوترهای شخصی تک کاربر، کاربردهای کوچک و با امکانات محدود است.
- روی کامپیوترهای متوسط و بزرگ می تواند به تعداد زیادی پایانه متصل بوده و دارای کارایی مناسبی باشد.

انواع معماری نامتمرکز

- معماری سیستم پایگاهی مشتری-خدمتگزار
- معماری سیستم پایگاهی توزیع شده
- معماری با پردازش موازی
- معماری چند پایگاهی
- سیستم پایگاهی همراه

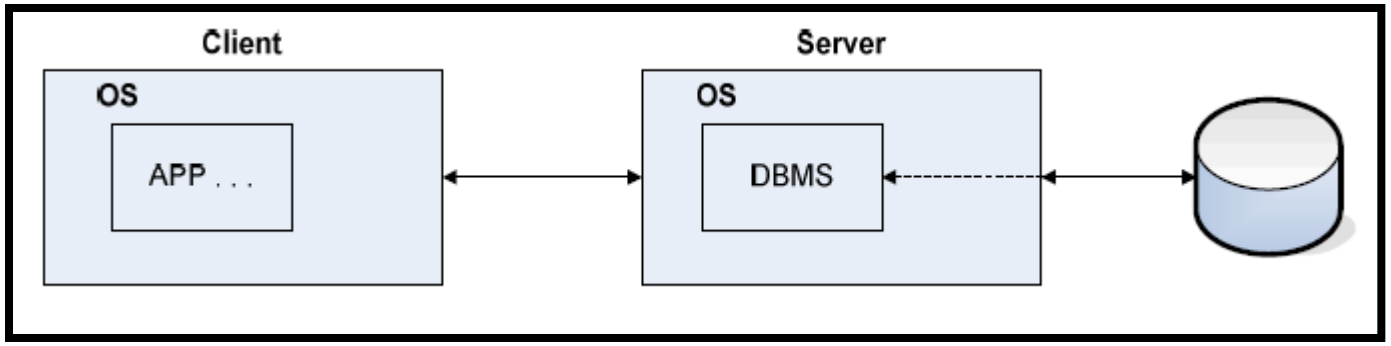
معماری سیستم های پایگاهی مشتری–خدمتگزار

- معنای عام:

هر معماری که در آن قسمتی از پردازش را یک برنامه، سیستم یا ماشین انجام دهد و انجام قسمت دیگر از برنامه را از برنامه، سیستم یا ماشین دیگر بخواهد.

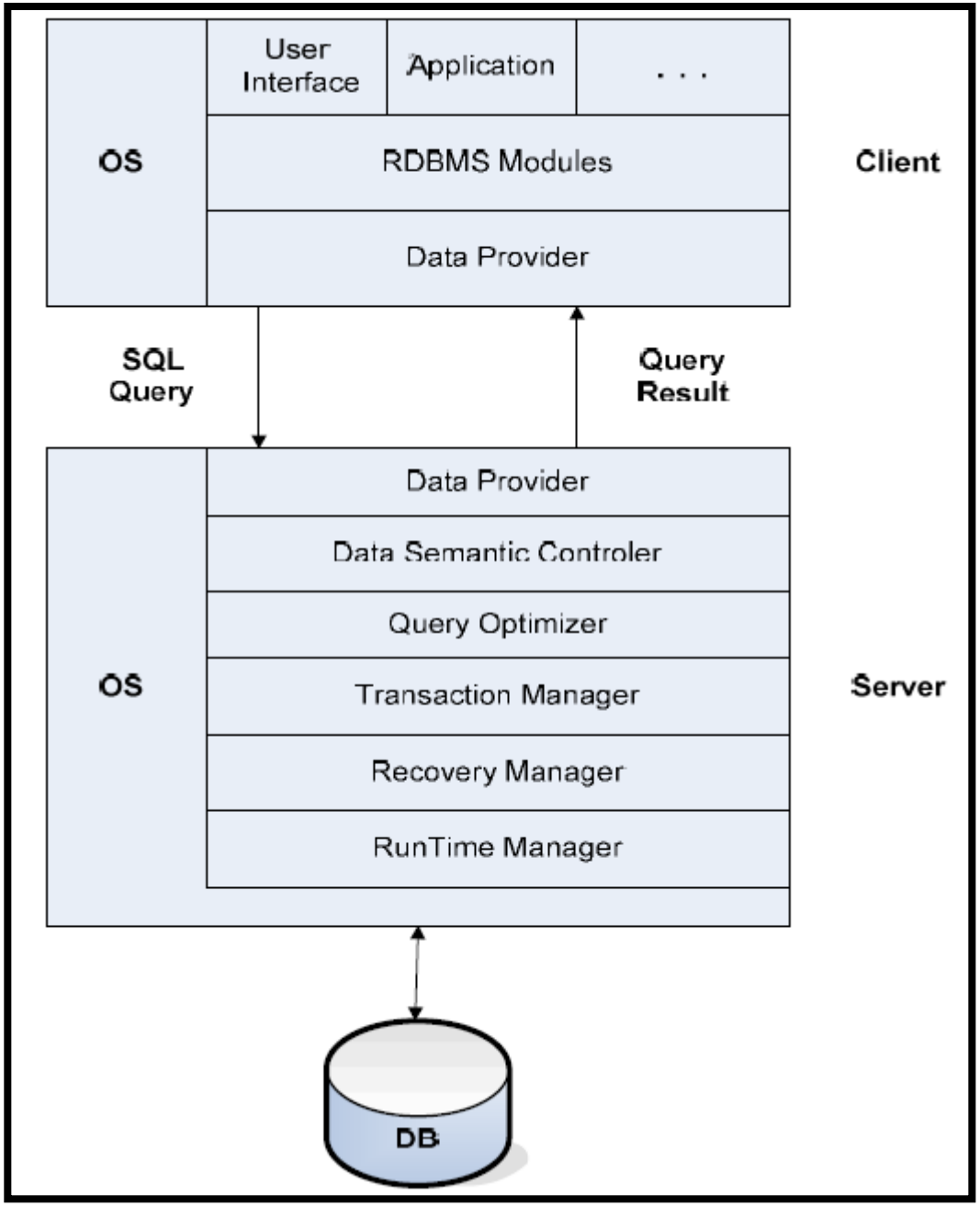
- وظیفه ای که باید "سیستم" انجام دهد به دو دسته تقسیم می شود:
  - دسته ای که انجام آن بر عهده خدمتگزار است.
  - دسته ای که انجام آن بر عهده مشتری است

معماری پایگاهی مشتری–خدمتگزار



- کاربر با وجود واسط هایی نظیر زبان داده فرعی، واسط گرافیکی و واسط فرمی عمل می کند.

اجزاء معماری پایگاهی مشتری–خدمتگزار

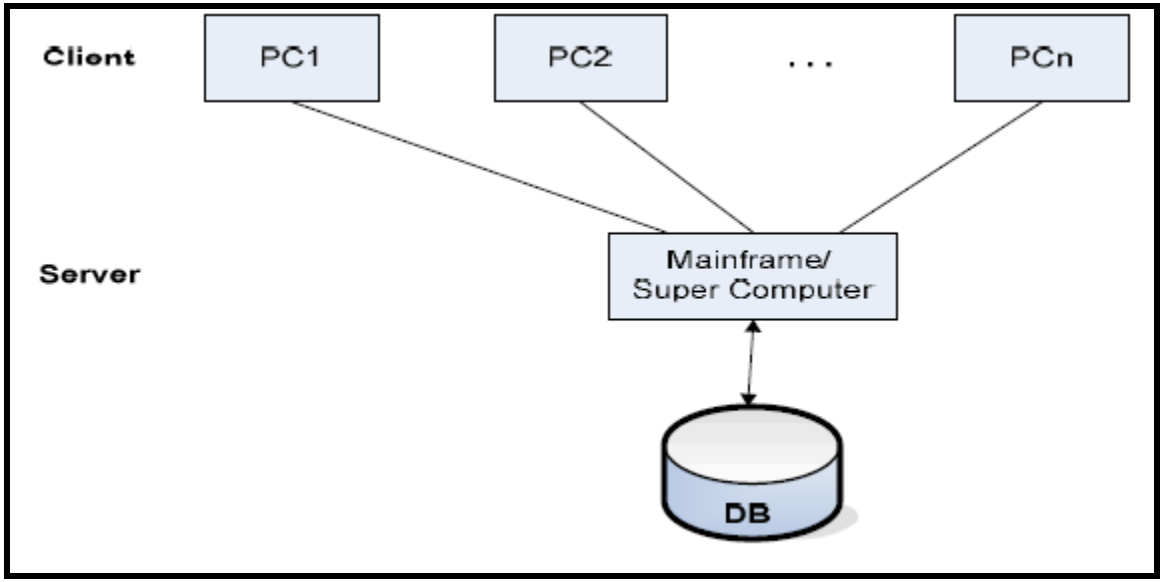


- اجزاء با در نظر گرفتن RDBMS .
- ابزارهایی مثل ODBC و JDBC برای تسهیل تماس مشتری و خدمتگزار
- امکان فراخوانی راه دور RPC

طرح های معماری مشتری–خدمتگزار

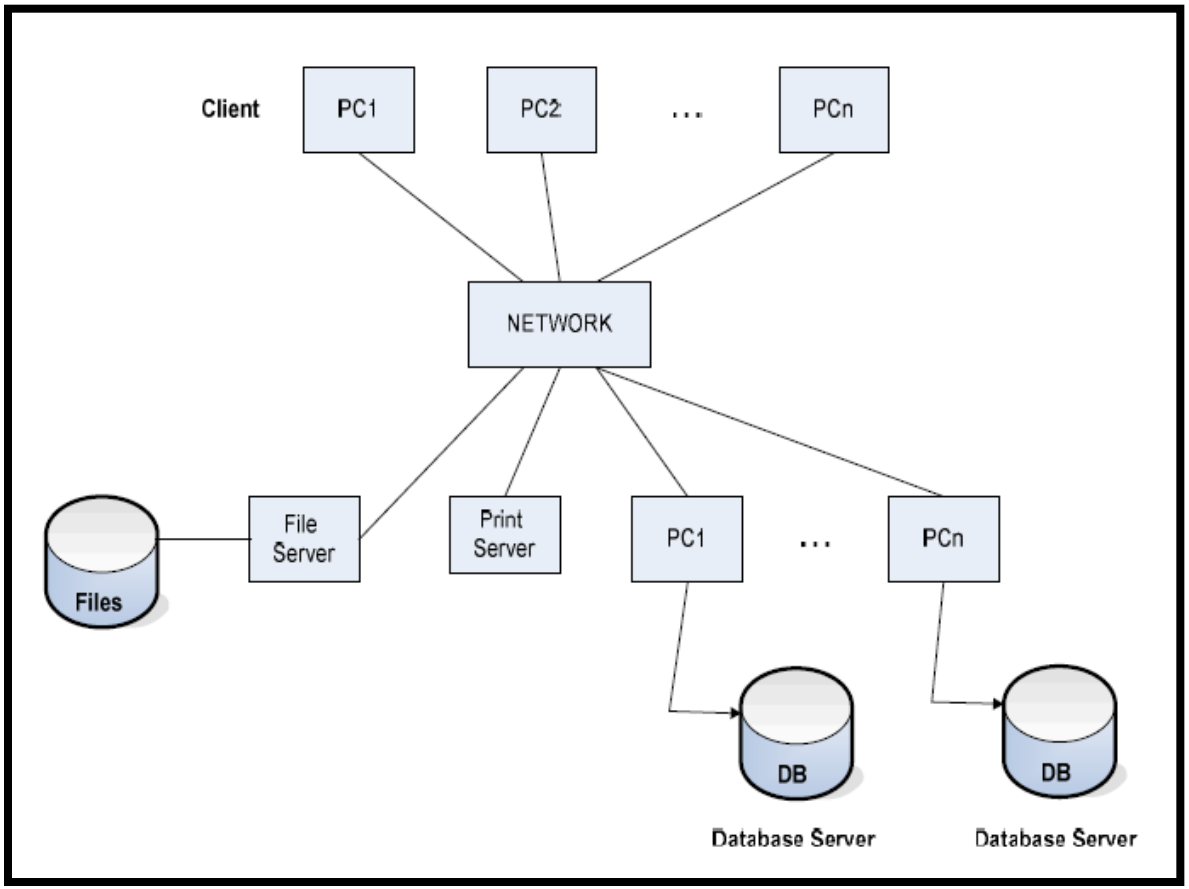
- از نظر تعداد مشتری
  - یک مشتری - یک خدمتگزار
  - یک مشتری - چند خدمتگزار
  - چند مشتری - یک خدمتگزار
  - چند مشتری - چند خدمتگزار
- از نظر پیکربندی کامپیوتری
  - معماری حول کامپیوترهای بزرگ
  - معماری حول شبکه

معماری حول کامپیوترهای بزرگ



- ماشین خدمتگزار یک کامپیوتر بزرگ است و پایگاه داده روی همین ماشین ایجاد و مدیریت می شود.
- تعدادی کامپیوتر شخصی از خدمات پایگاهی این کامپیوتر بزرگ استفاده می کنند.

معماری حول شبکه

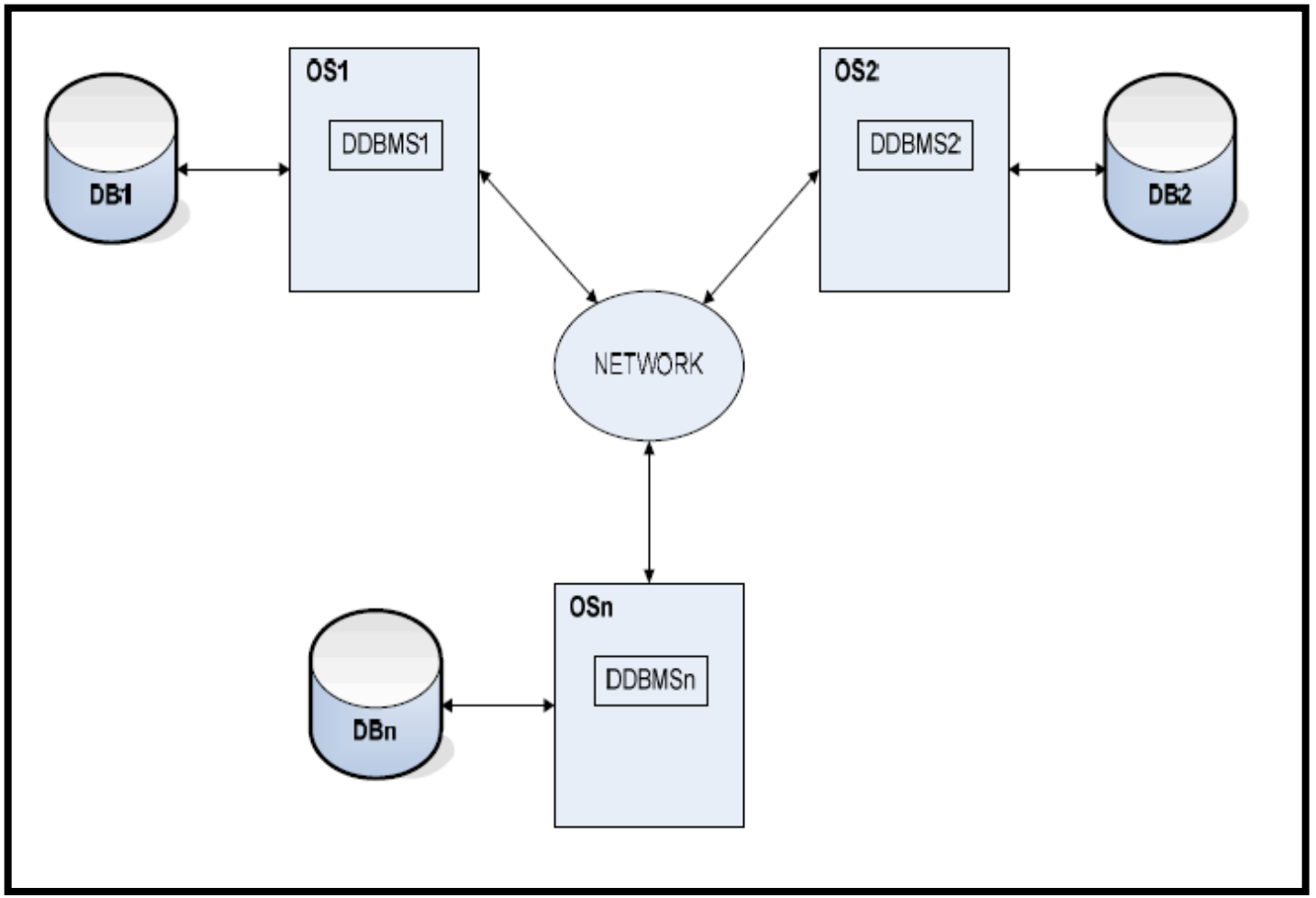


- تعدادی از کامپیوترهای بعنوان مشتری و تعدادی خدمتگزار هستند و با شبکه به یکدیگر مرتبط هستند.

مزایای معماری مشتری–خدمتگزار

- تقسیم پردازش
- کاهش ترافیک شبکه (در معماری حول شبکه)
- استقلال ایستگاههای کاری
- اشتراک داده ها

معماری سیستم پایگاهی توزیع شده



ویژگی های معماری سیستم پایگاهی توزیع شده

- مجموعه ای است از داده های منطقاً مرتبط و اشتراکی.
- بعضی از بخشها ممکن است بصورت تکراری (در چند نسخه) در کامپیوترها ذخیره شده باشند.
- کامپیوترها از طریق یک شبکه بهم مرتبط اند.
- داده های ذخیره شده در هر کامپیوتر تحت کنترل یک DBMS است.
- DBMS در هر کامپیوتر می تواند برنامه های کاربردی محلی را بصورت اتوماتیک اجراء کند.
- هر DBMS حداقل در اجراء یک برنامه کاربردی سرتاسری مشارکت دارد.

اصول حاکم بر معماری سیستم پایگاهی توزیع شده

- خود مختاری محلی (داخلی).
- تداوم عملیات.
- عدم وابستگی کامپیوترها به کامپیوتر اصلی.
- عدم وابستگی برنامه ها به مکان ذخیره سازی داده ها.
- عدم وابستگی برنامه ها به نحوه قرارگیری داده ها در کامپیوترها.
- پردازش پرس و جو ها به شیوه توزیع شده.
- عدم وابستگی برنامه ها به سخت افزار.
- عدم وابستگی برنامه ها به سیستم عامل.
- عدم وابستگی برنامه ها به سیستم مدیریت پایگاه داده ها.
- عدم وابستگی برنامه ها به شبکه.

مزایای معماری سیستم پایگاهی توزیع شده

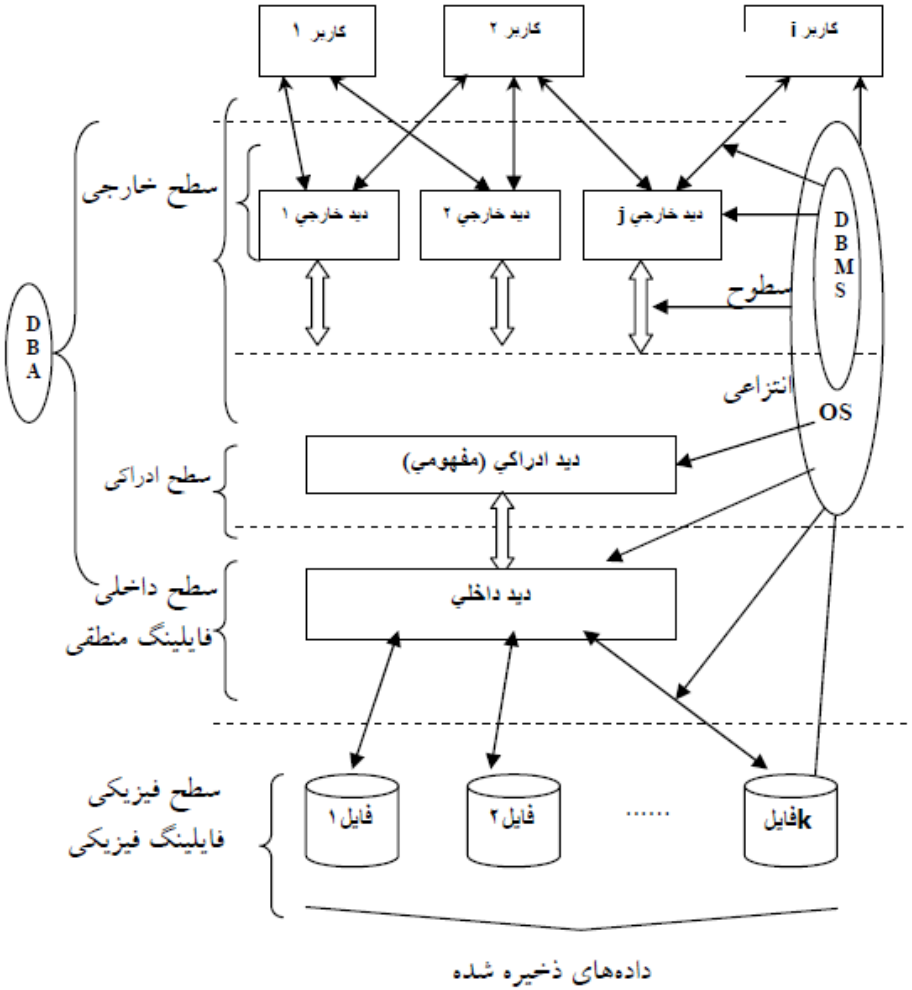
- سازگاری و هماهنگی با ماهیت سازمان های نوین.
- کارایی بیشتر در پردازش داده ها به ویژه در پایگاه داده های بزرگ.
- دستیابی بهتر به داده ها.
- اشتراک داده ها.
- افزایش پردازش موازی.
- کاهش هزینه ارتباطات.
- تسهیل گسترش سیستم.
- استفاده از پایگاه داده های از قبل موجود.

معایب معماری سیستم پایگاهی توزیع شده

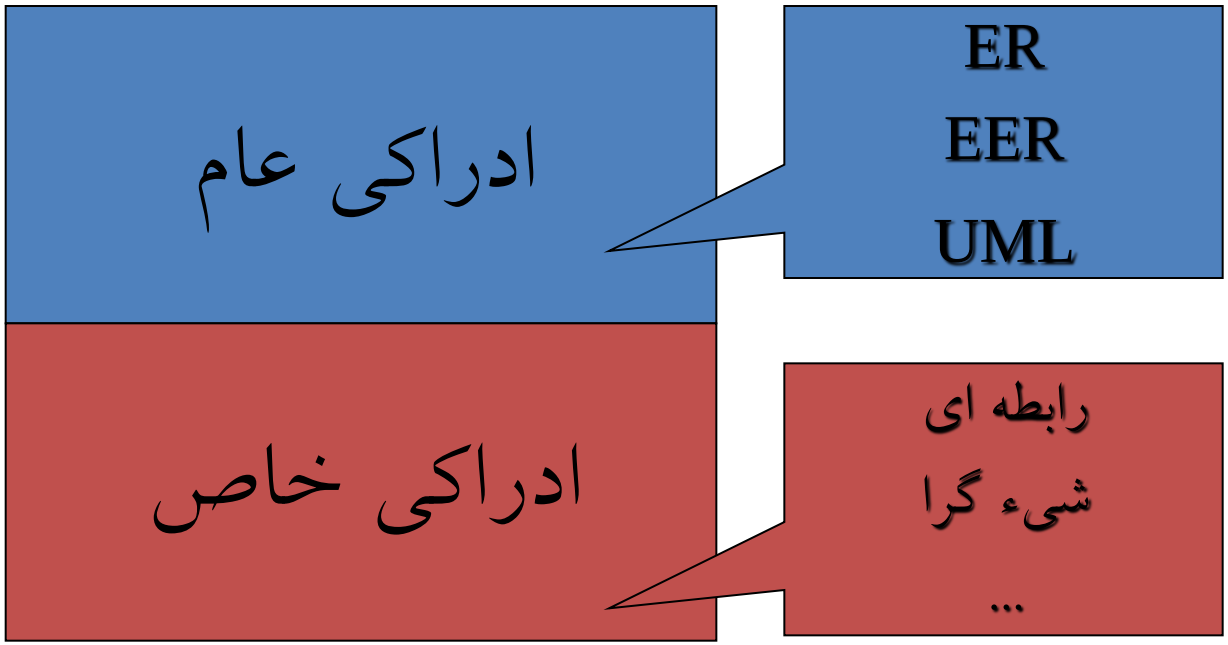
- پیچیدگی طراحی سیستم.
- پیچیدگی پیاده سازی.
- کاهش کارایی در برخی موارد.
- هزینه بیشتر.
- مصرف حافظه بیشتر.

بانکهای اطلاعات مبتنی بر شیء و XML

معماری چهار لایه ای بانک اطلاعات



لایه ادراکی



مهمترین مزایای مدل رابطه ای

- سادگی و قابل فهم بودن برای عموم
- پشتوانه تئوریک قدرتمند
  - جبر رابطه ای
  - حساب رابطه ای تاپلی
  - جساب رابطه ای دامنه ای
- ابزارهای نرم افزاری قدرتمند نظیر SQL
- پشتیبانی خوب از:
  - امنیت
  - جامعیت
  - نرمال سازی
  - بهینه سازی پرس و جو

مهمترین معایب مدل رابطه ای

- فاقد کارایی مناسب در کاربردهای جدید مثل CAD و CAM و ...
- عدم پشتیبانی از انواع جدید داده نظیر صوت و تصویر
- عدم پشتیبانی از:
  - رابطه های تو در تو
  - پرس و جوهای بازگشتی
  - دیدهای قابل بروز رسانی
- عدم کارایی در مدیریت تراکنش های طولانی مدت
- ناتوانی زبان پرس و جوی متداول
  - رویه ای بودن
  - عدم همخوانی با زبان میزبان
- داده های انفعالی: جدا بودن داده ها از رفتارها و مشخص نشدن ارتباط میان آنها

Object Management Group

- عمده فعالیت این گروه در زمینه مدیریت سیستم های شیء گرا می باشد.
- استانداردهای این گروه بعنوان مهمترین منابع موجود مورد استفاده قرار می گیرد.
- دارای کارگروه ویژه ODBTWG برای:
  - توسعه استانداردهای مختلف برای پایگاه داده شیء گرا
  - فناوریهای پایگاه داده شیء گرا نظیر (Replication)
  - مدیریت پایگاه داده نظیر (Spatial Indexing)
  - فرمت های مختلف نگهداری اطلاعات نظیر (XML)
- سیستم های بلادرنگ
- نرم افزارهای متن باز نظیر db4o

مفاهیم مدل داده شیء گرا

کلاس

قالب اطلاعات که ویژگی های مورد نظر برای مجموعه همسان از اشیاء دنیای واقعی را بیان می کند

شیء

مجموعه مقادیر داده هایی که یک پدیده دنیای واقعی را معرفی می کنند و نمونه ای از کلاس می باشند.

تناظر مفاهیم مدل داده شیء گرا و رابطه ای

مدل داده ای شیء گرا	مدل داده ای رابطه ای
کلاس (Class)	موجودیت (Schema)
شیء (Object)	نمونه (Instance)

کلاس

- مجموعه متغیرها (instant variable) که قالب داده های مربوط به کلاس را نگهداری می کنند.
- مجموعه ای از متدها (method) که هر یک کدی است که برای پیاده سازی یک رفتار نوشته شده است.

مثال

```
class student
{
     /* variables */
       string sname;
       string city;
        …

    /*method*/
       int avg();
       string get_name();
       int set_address(string new_address);
}
```

```
int set_address(string new_address);
{
        address=new_address;
}
```

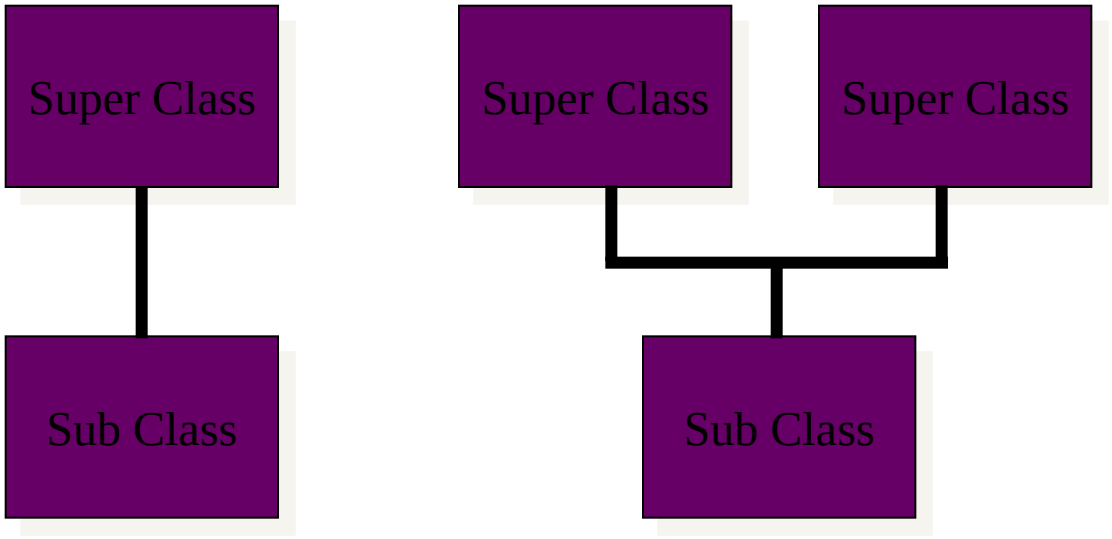
شناسه شیء (O-ID)

بمنظور برقراری تناظر یک به یک بین هر پدیده در دنیای واقعی با هر شیء ذخیره شده در بانک اطلاعات می باشد. این شناسه با مرور زمان و حتی با تغییر برخی یا همه مقادیر صفات آن پدیده تغییر نخواهد کرد. شناسه می تواند انتخابهای زیر باشد:

- نام: که توسط کاربر تعریف می شود.
- Built-in: شناسه موجود در زبان برنامه نویسی یا مدل داده ای.
- System generated: مقداری که توسط سیستم بانک اطلاعات تولید می شود.

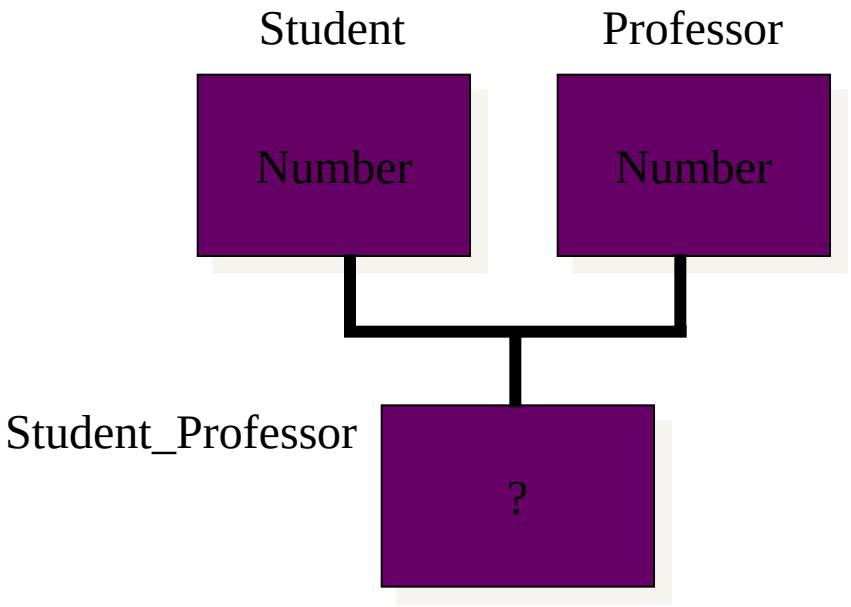
وراثت (Inheritance)

- این مفهوم بواسطه برقراری اشتراک داده ها و رفتارهاست.
- وراثت به معنای اشتقاق کلاسها از یکدیگر و استفاده از کلاسها در تعریف کلاس دیگر است.



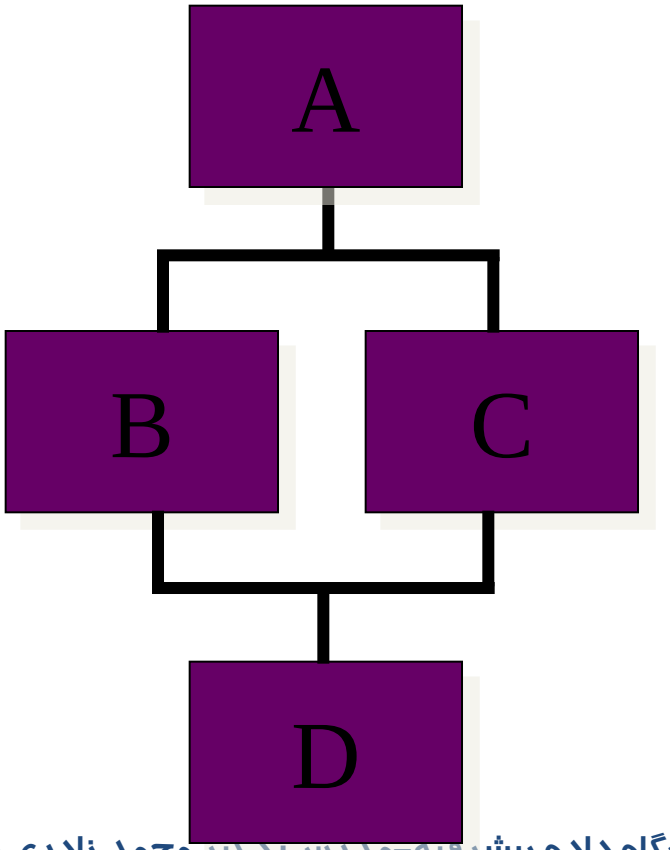
برخورد نام ها (Name Clash)

- در سلسله مراتب کلاسها ممکن است یک نام ممکن است با معانی مختلفی آمده باشد.



ارث بری تکراری
(Repeated Inheritance)

- این پدیده زمانی رخ می دهد که اجداد کلاس در درخت وراثت از دو طریق به یک نقطه ختم می شوند.



بسته بندی (Encapsulation)

کنار هم قرار دادن داده ها و رفتارها در یک بسته (کپسول) را بسته بندی می گوییم.

- باعث "استقلال پدیده ها" می گردد یعنی هر چه به یک پدیده مربوط است همراه اوست.
- باعث "پنهان سازی اطلاعات" نیز می گردد.

مفاهیم بسته بندی

- چهره درونی کلاس-شیء (Internal): شامل تمام داده ها و برخی از رفتارهای کلاس یا شیء است. دسترسی به این بخش فقط و فقط توسط خود کلاس یا شیء امکان پذیر است. اشیاء از ساختار درونی یکدیگر بی خبرند و به جزئیات هم کاری ندارند.
- چهره رابط کلاس-شیء (Inteface): شامل تعدادی از رفتارهاست. تنها وسیله ارتباطی اشیاء چهره رابط آنهاست. اشیاء درون یک سیستم با فراخوانی چهره رابط یکدیگر (پیام رسانی) کارها را انجام می دهند.

- یک قانون مهم: "چهره رابط باید کمینه باشد"
- **پنهان سازی اطلاعات**
- **نگهداری سیستم و عدم سرایت تغییرات به بخشهای دیگر.**
  - **این چهره باید با حالتی پایا و یا آینده نگری لازمه طراحی شود تا نیاز به تغییر نداشته باشد.**

چند ریختی (Polymorphism)

- در زبانهای برنامه نویسی استفاده از اسامی تکراری بصورت محدود امکان پذیر است.
- در دنیای شیءگرایی اسم تکراری به فراوانی رخ می دهد.
- چند ریختی به اشیاء مختلف این امکان را می دهد که به یک پیام واحد پاسخ های متفاوتی بدهند.
- رفتارهایی که شامل چند ریختی می شوند در چهره رابط مشابهند ولی در پیاده سازی متفاوت؛ یعنی نام یکسان و پارامترهای متفاوت دارند.

پیام رسانی (Message Passing)

- بمعنای فراخوانی رفتارهای اشیاء است. X.Y(Z)
- اشیاء با دریافت پیام عکس العمل مناسب نشان می دهند.
- شرایط پیام:
  - پیش شرط ها (Pre-Conditions): شرایطی که پارامتر ورودی باید داشته باشد و در صورت عدم برآورده شدن آن شیء اعلام خطا خواهد کرد.
  - پس شرط ها(Post-Conditions): شرایطی که پارامترهای خروجی باید داشته باشند.
  - اقداماتی که شیء در صورت بروز خطا باید انجام دهد. مثلاً اگر شیئی بعنوان مدیر تراکنش تعریف شود در صورت بروز خرابی یا اتفاق غیر منتظره باید اقدام به بازگرد نماید.
- یکی از مشکلات عمده پیام رسانی در سیستم های بزرگ به یاد آوردن رفتار اشیاء گوناگون است. بخصوص برای کاربران بانک اطلاعات که از اشیاء و کلاسهای متعدد استفاده می نمایند.

انواع پیام

- پیام هماهنگ (Synchronous): فرستنده و گیرنده درگیر هستند تا کار به اتمام برسد. فرستنده منتظر پاسخ است و کار دیگری انجام نمی دهد.
- پیام ناهماهنگ (Asynchronous): فرستنده بدون توجه به وضعیت گیرنده پیام را ارسال می کند و خودش به کار دیگری می پردازد. ممکن است از پاسخ بصورت مستقیم یا غیر مستقیم استفاده کند.
- پیام آینده (Future): فرستنده پیام را ارسال می کند و در زمان مقرر یا در فواصل زمانی مشخص منتظر پاسخ است.
- پیام فرصتی (Timeout): فرستنده مدت مشخصی منتظر پاسخ می ماند که یا آماده دریافت پیام شود یا پاسخ دهد. پس از اتمام مدت، ارتباط قطع شده و یا پیام مجدداً تکرار می شود.
- پیام فوری (Balking): فرصت پیام نزدیک به صفر است. یعنی اگر گیرنده بلافاصله آماده دریافت پاسخ نباشد، فرستنده ارتباط را قطع می کند.

پیوند پویا (Dynamic/Late Binding)

- لازم است نوع پیاده سازی برخی از کلاسها تا لحظه اجراء نامعلوم باشد.
- برنامه در زمان اجراء باید نوع پیاده سازی این کلاسها را تعیین کند.
- باعث افزایش انعطاف پذیری در پیاده سازی اشیاء می شود.
- باعث کاهش سرعت اجراء برنامه ها نیز می گردد.

انواع کلاس

- کلاس پایه (Base Class): توسط برنامه مورد نظر قبلاً تعریف شده اند.
- کلاس مجرد (Abstract Class): هدف از تعریف این کلاسها، استفاده از آنها در ساخت کلاسهای دیگر است و نه تولید شیء. نام دیگر این دسته از کلاسها بدون نمونه non-(instance) است.
- فوق کلاس (Meta Class): به کلاسهایی گفته می شود که کلاسهای دیگری ایجاد می کنند، همچنانکه کلاسهای معمولی اشیاء را ایجاد می کنند.

معیارهای سنجش کیفیت کلاس

۱. همبستگی (Coupling): کلاس ها نباید به صورت نامعقول به هم وابسته باشند.
۲. پیوستگی (Cohesion): پیوستگی ذاتی داده ها و رفتارهای یک کلاس با هم.
۳. کامل بودن (Completeness): چهره واسط باید همه خصوصیات آنرا در رابطه با کاربرد مورد نظر در بر بگیرد و پیش بینی ها لازم شده باشد.
۴. کفایت (Sufficiency) و بی پیرایگی (Primitiveness): جامع و مانع بودن.

داده مانا (Persistent Data)

- در بانک اطلاعات داده ها اصالت دارند و برنامه ها در حال تغییر. داده ها باید مورد حفاظت قرار بگیرند.

۱. مانایی زمانی
   - مانایی اشیاء
   - مانایی کلاسها

۲. مانایی مکانی

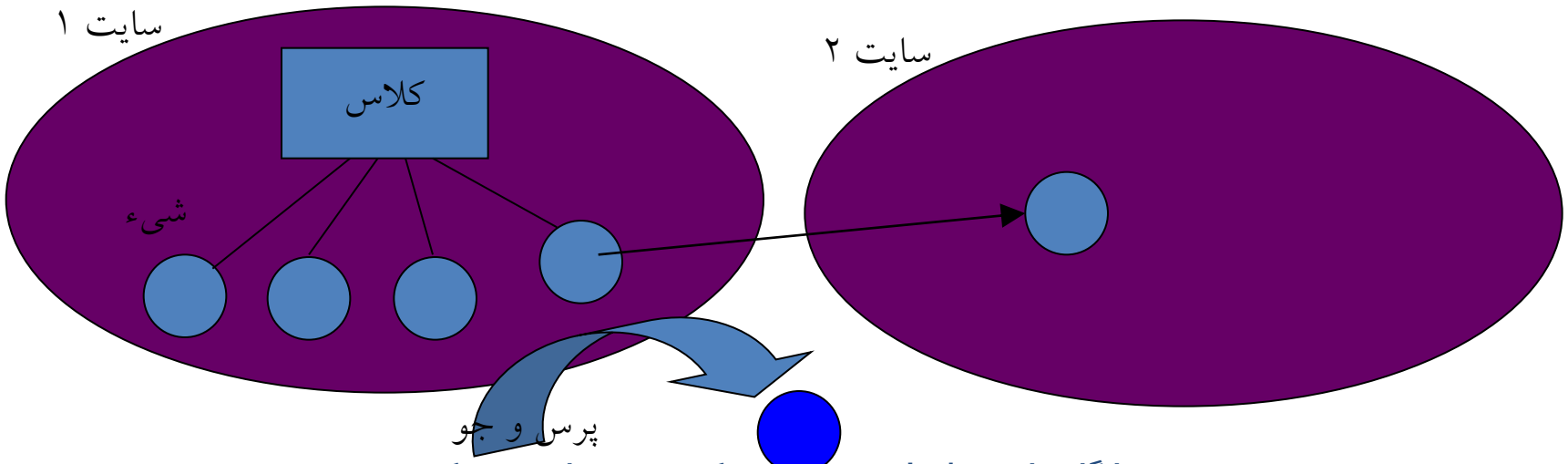
طراحی بانک اطلاعات شیء گرا

- هر مدل داده ای در واقع انتزاعی است از دنیای خارج.
- این انتزاع شامل داده ها و ارتباطات میان آنهاست.
- در مدل بانک اطلاعات شیء گرا انتزاع های زیر وجود دارد:
  - **داده ها** تنها بصورت شیء ذخیره می شوند.
  - **ارتباطات میان داده ها**، مواردی مثل نوع داده، صفت ها، محدودیت ها و مسائل زمان و مکان را در بر می گیرد.

مهمترین انتزاع های پیشنهادی در بانک اطلاعات شیء گرا

۱- کلاس بندی (Classification)

- به مفهوم کلاس بر می گردد و رابطه تنگاتنگی با تعریف نوع داده دارد.
- با عنوان instance-of شناخته شده است.



مهمترین انتزاع های پیشنهادی در بانک اطلاعات شیء گرا

۲- گروه بندی (Aggregation و یا Grouping)

- به معنای تشکیل مجموعه ای از اشیاء است.
- به بیان دیگر ایجاد یک شیء بزرگتر از روی مجموعه ای از چند شیء.
- این انتزاع کارایی بالایی دارد می توان از آن برای تولید ساختارهایی نظیر مجموعه ها و آرایه ها استفاده نمود.
- این انتزاع با عنوان belongs-to و در حالت خاص به نام is-part-of شناخته شده است.
- مثال: "شیء آدرس"

address=(country, city, street, no, zip)

مهمترین انتزاع های پیشنهادی در بانک اطلاعات شیء گرا

۳- ارث بری (generalization/specialization)

- همان مفهوم ارث بری است که قبلاً اشاره شد.
- با عنوان is-a شناخته شده است.

مهمترین انتزاع های پیشنهادی در بانک اطلاعات شیء گرا

٤- ارتباط (association)

- نشان دهنده ارتباط میان اشیاء است.
- برخلاف نظر برخی از افراد، این ارتباط خود می تواند یک شیء باشد. (نظیر مدل رابطه ای)
- این رابطه با عنوان member-of شناخته شده است.

روشهای مختلف طراحی بانک اطلاعات شیء گرا

۱- استفاده از ساختارهای زبان های شیء گرا موجود.

۲- هدایت EER بسوی مدل شیء گرایی و مدلهایی نظیر ODL و UML.

موارد مهم تعریف کلاس در ODL

۱. کلمه interface بیانگر آن است که آنچه تعریف می شود فقط چهره رابط است.
۲. نام رابط تعریف شده.
۳. اجزاء کلاس نظیر صفت ها (attributes)، ارتباط ها (relationships) و روال ها یا متد ها (methods) درون آکولاد تعریف می شوند.

مثال : تعریف کلاس دانشجو

```
interface stud{
    attribute integer s#;
    attribute string sname;
    attribute string city;
    attribute float avg;
    attribute integer clg#;
/* method */
pass (in crs#, out score)
/* relationship */
relationship set <sec> enroll
    inverse stud::enroll;
relationship loan get-loan
    inverse loan::give-loan;
};
```

```
interface loan{
relationship stud give-loan
    inverse stud::get-loan;
};
```

مثال : تعریف کلاس دانشجوی کارشناسی ارشد

```
interface gradstud:stud {
    attribute struct thesis {date start, finish; string title} mythesis;
pass (in crs#, out score);
relationship set <sec> courses
    inverse sec::gets;
};
```

مثال : تعریف کلاس دانشجوی کارشناسی ارشد (کامل تر)

```
interface gradstud:stud {
    attribute struct thesis {date start, finish; string title} mythesis;
relationship set <sec> courses
    inverse sec::gets;
pass (in crs#, out score);
instructors (out set<prof>)
    raises(no-insturctor);
classmates (in c#, out set <stud>)
    raises(no-class-mate);
};
```

XML

فهرست مطالب

- ✓معرفی XML
- ✓آشنایی با قواعد نوشتاری یک سند XML
- ✓تهیه سند اعتبار: DTD و شمایXML
- ✓ نمایش سند XML در وب: XSLT و CSS

معرفی XML

XML چیست؟

- ✓XMLمخففeXtensible Markup Language می باشد.
- ✓XML برای توصیف و مدیریت بهتر اطلاعات به وجود آمده است و در واقع به مانند زبان های برنامه نویسی، قادر به انجام کاری نیست.
- ✓ XML در ساختار خود بسیار به زبان HTML شبیه است ولی با قدرت بسیار بیشتر برای توصیف اطلاعات.

XML در مقایسه با HTML

HTML

- ✓ثابت بودن عبارات تگ ها.
- ✓آزاد بودن در رعایت قوانین نوشتاری
- ✓هدف اصلی HTML برای نمایش اطلاعات به فرمت مناسب است.

XML

- ✓ تعریف تگ ها در اختیار کاربر است.
- ✓ سخت گیری در پیاده سازی دقیق قوانین.
- ✓ هدف اصلی XML در راستای توصیف داده ها و ایجاد متن های فرا داده ایست و به تنهایی قادر به نمایش اطلاعات به مانند HTML نیست.

XML	HTML
<?xml version="1.0""?> <paper> <title> Issues with Designing Large Scale Libraries Based on NCSTRL </title> <authors> K. Maly, M. Zubair, H. Anan, D. Tan, and Y. Zchang </authors> <abstract> NCSTRL is an unified …..</abstract> </paper>	<html> <head> Issues with Designing Large Scale Libraries Based on NCSTRL </head> <body> K. Maly, M. Zubair, H. Anan, D. Tan, and Y. Zchang Department of Computer Science, Old Dominion University ….. </body> </html>

چرا XMLدارای اهمیت فراوان است؟

- ✓ **آسان کردن دستیابی و جستجو در اطلاعات**

استفاده از تگ هایی برای توصیف داده ها در متن باعث می شود تا دراین حالت موتور های جستجو با استفاده از این تگ ها بتوانند به راحتی اطلاعات خواسته شده از هر متن را جستجو کنند.(برای اطلاعات بیشتر رجوع شود به مبحث semantic web search engines)

- ✓ **آسان شدن تبادل اطلاعات بین سیستم های ناهمگون.**

در دنیای واقعی سیستم های کامپیوتری و پایگاه داده ها شامل داده ها یی هستند که در فرمت های متفاوتی ذخیره و دسته بندی شده اند .یکی از کارهای پر دردسر تبادل داده ها بین این سیستم ها در محیط اینترنت است.

تبدیل اطلاعات به یک فرمت مشخص به مانند xml می تواند تا حد بسیار زیادی این مشکل را بر طرف کند.(برای اطلاعات بیشتر رجوع شود یه مبحثEDI در سیستم های تجارت الکترونیک)

ابزار نوشتن XML

✓XML را می توان همانند HTML با هر ویرایشگر متنی بر روی هر سیستم عامل کامپیوتری نوشت(مانند Note Pad در ویندوز)تنها باید آن را با پسوند ***xml .*** ذخیره نمود.

✓همچنین ویرایشگر های متنی ویژه ای به بازار آمده است که می توانند اسناد XML را هنگام نوشتن تست کنند و از صحت رعایت قوانین نوشتاری آن مطمئن شوند.(به عنوان مثال می توانید Stylus Studio را از این سایت دانلود کنید (http://www.stylusstudio.com/xml_download.html

✓برنامه هایی وجود دارند که می توانند اسناد سایر برنامه ها (مانند بانک های اطلاعاتی ، صفحات طراحی یک سیستم و ...)را به XML تبدیل کنند.برای مثال در خود برنامه SQLserver 2000 دستوراتی وجود دارد تا داده های داخل پایگاه داده را به یک سند XML تبدیل کند و بسیاری از موارد مشابه

اجزاء اساسی برای تولید و نمایش یک سند XML تحت وب

✓ **سند XML**

جز اصلی و اساسی است که با استفاده از تگ های خصوصی و مورد علاقه هر شخص تولید می شود.

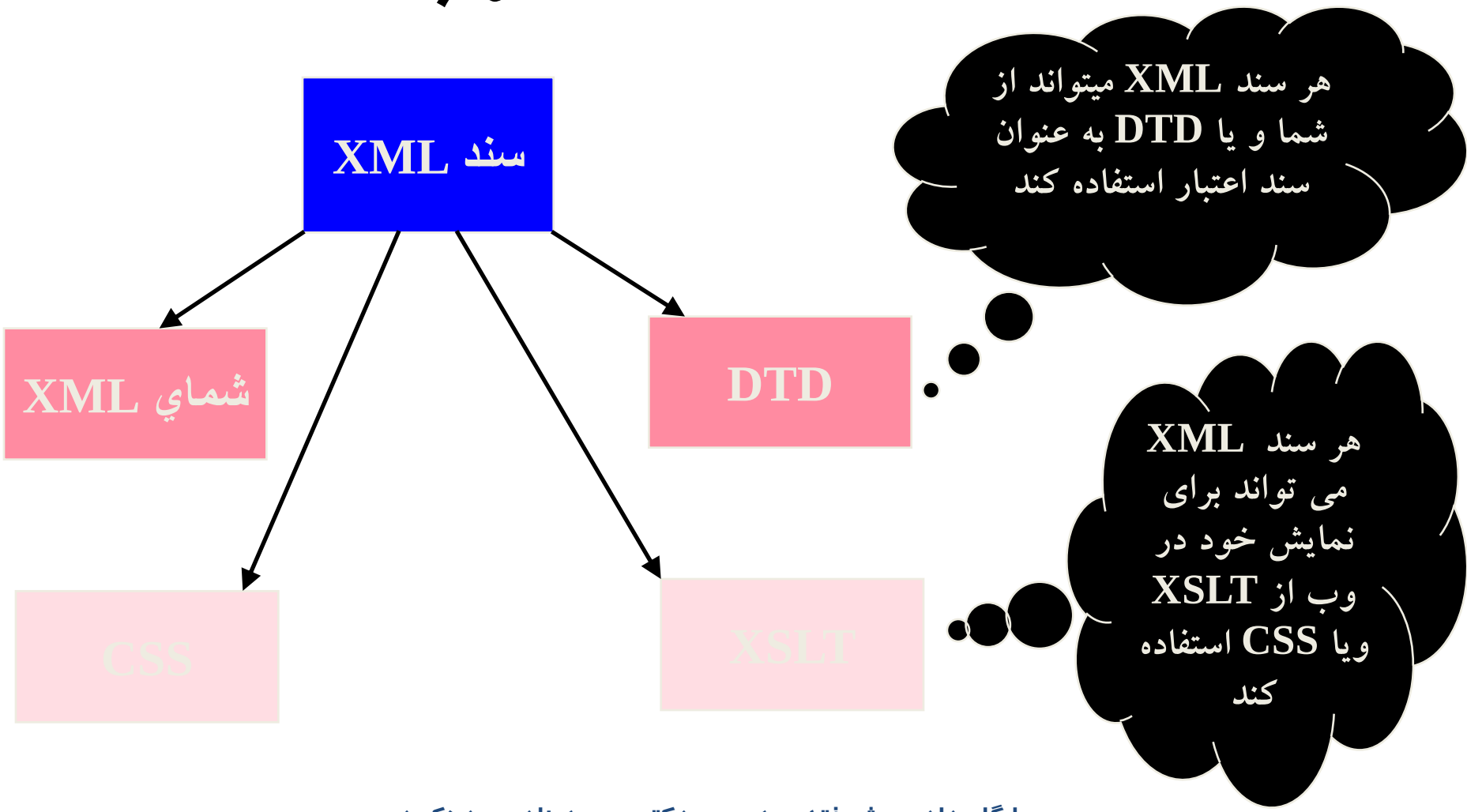
✓ **DTD و شمای XML**

پس از تولید هر سند نکته اساسی برای استفاده کنندگان این سند این است که آیا سند تولید شده معتبر است . برای مثال قوانین تو درتویی تگ ها ، مقادیر صفت ها و... درست رعایت شده یا نه. به این منظور از ابزارهایی تحت عنوان DTD ها و شمای XML استفاده می کنند.

✓ **XSLT و CSS**

سند XML به خودی خود در وب قابل نمایش نیست برای تبدیل سند XML به فرمت قابل نمایش در وب از XSLT و یا CSS استفاده می شود.

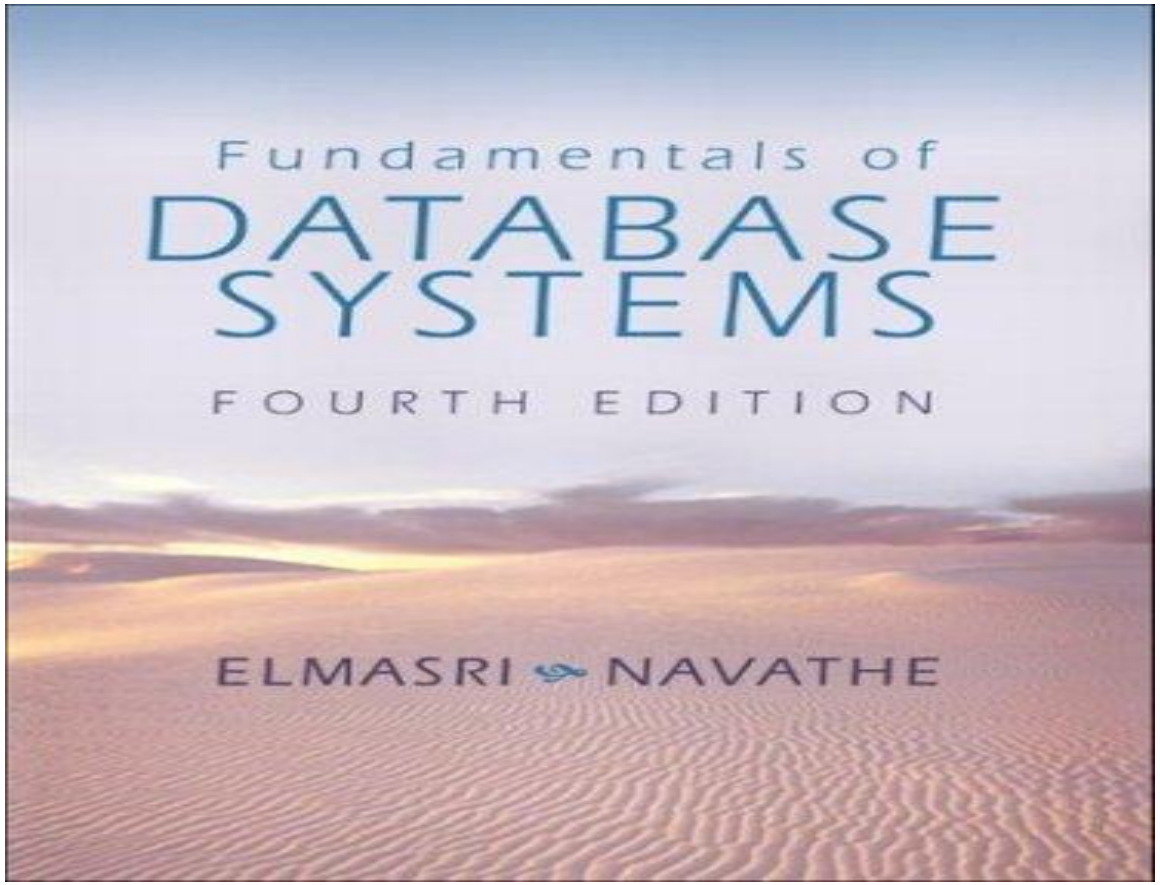
اجزاء اساسی برای تولید و نمایش یک سند XML تحت وب



آشنایی با قواعد نوشتاری یک سند XML

تولید سند XML
یک مثال ساده

می خواهیم اطلاعات این کتاب را به صورت یک سند XML نمایش دهیم.



قبل از هر چیز در ابتدای فایل XML باید اعلان XML version مورد استفاده را کرد برای این منظور:

```
<?xml version="1.0" ?>
```

- ✓ در ابتدای سند و قبل از هر چیز بنویسید <?xml
- ✓ سپس بنویسید "version=1.0" (یا هر version دیگری که مورد استفاده است)
- ✓ در انتها ?>

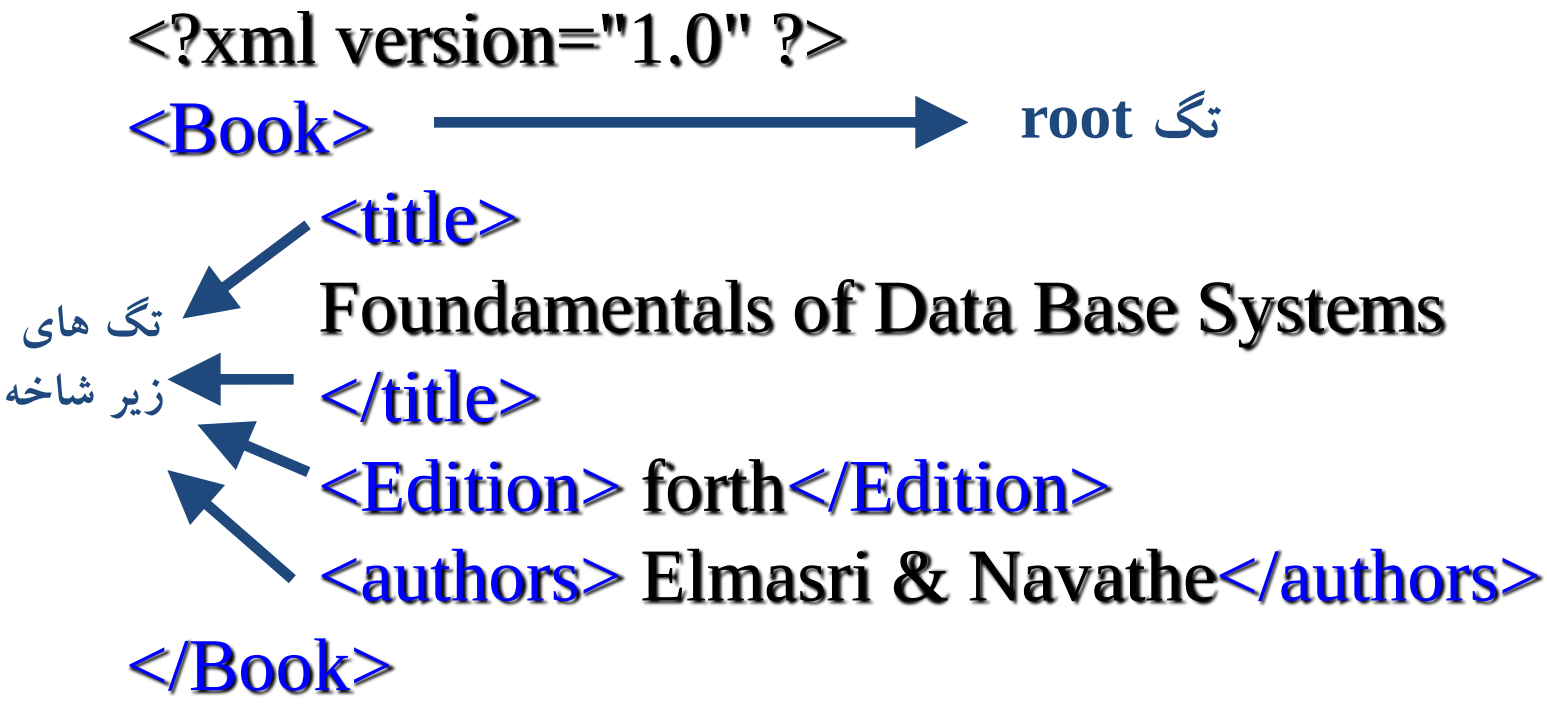
پس از آن موضوعی را که در مورد آن می خواهیم بحث کنیم که در اینجا یک کتاب است به صورت تگ اصلی نمایش می دهیم .

```
<?xml version="1.0" ?>
<Book>

</Book>
```

پس از آن هر بخش از اطلاعات کتاب را به صورت زیر شاخه در داخل تگ اصلی اضافه می کنیم و در جلوی آن اطلاعات مربوط به آن را می نویسیم.

```
<?xml version="1.0" ?>
<Book>
        <title>
        Foundamentals of Data Base Systems
        </title>
        <Edition> forth</Edition>
        <authors> Elmasri & Navathe</authors>
</Book>
```



چه قوانینی را در نوشتن یک فایل XML باید رعایت کرد؟

- ✓در ابتدای هر سند و قبل از هر چیزی <?XML version="1.0"?>
- ✓هر تگی که بازمی شود باید حتما بسته شود.
- ✓هر سند باید شامل یک عنصر پایه(root) باشد.
- ✓بزرگی و کوچکی حروف در XML حائز اهمیت است.
- ✓در رعایت سلسله مراتب بازو بسته نمودن تگ ها باید دقت شود.
- ✓برای اضافه کردن Comment به برنامه از علامت زیر استفاده می شود.

<!-- This is a comment -->

استفاده از صفت

- در یک سند XML هر یک از تگ های زیر شاخه را می توان به صورت یک صفت نیز نمایش داد.

- صفت را به تگ root اضافه می کنیم و داده قبلیش را نیزبه عنوان مقدار صفت به آن می افزاییم.

- مقدار هر صفت باید حتما در بین دو علامت " " و یا ' ' قرار داشته باشد.

یک مثال دیگر

برای مثال می خواهیم
این پیغام را مدل کنیم

Note
To: Tove
From: Jani
Date :12/11/2002

```
<?xml version="1.0"?>
<note>
<date> 12/11/2002</date>
<to> Tove </to>
<from> Jani</from>
</note>
```

میتوان این پیغام را به دو
شیوه مدلسازی کرد یکی با
استفاده از صفت و دیگری
بدون استفاده از آن

```
<?xml version="1.0"?>
<note date="12/11/2002">
<to> Tove </to>
<from> Jani</from>
</note>
```

چه تفاوتی در استفاده از صفت به جای یک تگ زیر شاخه وجود دارد؟

- ✓یک صفت نمی تواند چند مقدار داشته باشد ولی یک تگ می تواند چندین بار تکرار شود و هر بار یک مقدار متفاوت را اتخاذ کند.
- ✓یک صفت نمی تواند ساختار سلسله مراتبی را به خوبی بازگو کند.
- ✓برای تجزیه گر های مستندات XML کار با صفت مشکل تر از کار با یک زیر شاخه است (از لحاظ پیچیدگی زمان اجرای یک الگوریتم).

تهیه سند اعتبار:

DTD
شمای XML

DTD (Document Type Definition) چیست؟

- برای تعیین صحت اسناد تعریف می شود.
- گاهی به منظور یکپارچه سازی در استفاده از تگ ها با معانی مشخص و از پیش تعیین شده و تعیین استاندارد مورد استفاده قرار می گیرد.
- به این منظور در یک سند DTD باید عنوان تگ های استفاده شده و همچنین تو در تویی آنها ، صفتهای استفاده شده و نوع مقادیری که می پذیرند مشخص شود.
- هر سیستم قبل از استفاده از هر سند XML آن را با DTD آن سند مطابقت می دهد.چنانچه سند xml با سند DTD هم خوانی نداشت ، معتبر (well-formed) شناخته نمی شود و بی ارزش است.

انواع DTD

✓DTD داخلی

برای اسناد XML شخصی و تکی را حت تر آن است که DTD را در داخل خود سند ایجاد کنید.

✓DTD خارجی

- ✓چنانچه چند سری سند XML به هم مرتبط وجود دارد می توان از یک DTD برای تمام آنها استفاده کرد (به جای نوشتن DTD در هر سند). به این ترتیب یک فایل خارجی شامل DTD مورد نظر ساخته می شود و سپس اسناد XML مورد نظر توسط URL این فایل به آن مرتبط می شوند.

DTD داخلی

```
<?xml version="1.0"?>
<!DOCTYPE note [


]>
<note>
<to> Tove</to>
 <from> Jani</from>
<date> 12/11/2002 </date>
</note>
```

برای مثالی که قبلا سند XML آن را نوشته بودیم حال می خواهیم یک DTD داخلی ایجاد کنیم برای این منظور:

- ✓بعد از اعلان نسخه XML تایپ می کنیم <!DocType note
- ✓مقداری فضای خالی برای نوشتن محتوای سند DTD قرار می دهیم
- ✓در انتها تایپ می کنیم]>

DTD داخلی

```
<?xml version="1.0"?>
<!DOCTYPE note [
 <!ELEMENT note (to,from,date)>
]>
<note>
<to> Tove</to>
 <from> Jani</from>
<date> 12/11/2002 </date>
</note>
```

حال باید به ترتیب تگ های آورده شده در سند را معرفی کنیم ، ابتدا تگ root را معرفی می کنیم:

- ابتدا می نویسیم <!ELEMENt
- بعد نام تگ root را می نویسیم
- سپس در جلوی آن نام تگ هایی که به صورت زیر شاخه در عنصر root تعریف می شوند را می نویسیم.(to,from,date)
- در انتها می نویسیم >

DTD داخلی

```
<?xml version="1.0"?>
<!DOCTYPE note [
 <!ELEMENT note (to,from,date)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
<!ELEMENT date (#PCDATA)>
]>
<note>
<to> Tove</to>
 <from> Jani</from>
<date> 12/11/2002 </date>
</note>
```

پس از آن به ترتیب هر یک از تگ ها را تعریف می کنیم.عبارت #PCDATA در واقع مشخص می کند که داده این تگ یک عبارت متنی است و در ضمن نمی توان این تگ را چندین بار تعریف کرد و هر بار یک داده ای به آن اختصاص داد.البته توصیفگرهای دیگری نیز وجود دارند مانند:

#CDATA
#IMPLIED
#REQUIRED
…..

DTD داخلی(برای توصیف صفت)

```
<?xml version="1.0"?>
<!DOCTYPE note [
 <!ELEMENT note (to,from,date)>
<!ATTLIST note date (#PCDATA)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
]>

<note date="12/11/2002 ">
<to> Tove</to>
 <from> Jani</from>
</note>
```

چنانچه تگ اصلی دارای صفتی باشد آن را به صورت زیر در DTD اضافه می کنیم:

- ✓بعد از معرفی تگ ریشه می نویسیم <!ATTLIST
- ✓بعد از آن نام تگ ریشه و صفت آن را می نویسیم
- ✓پس از آن با توجه به مقدار صفت نوع آن را با یکی از توصیفگر ها نمایش می دهیم.
- ✓در انتها تایپ می کنیم >

DTD خارجی

✓ابتدا این فایل را با پسوند ***.dtd*** ذخیره می کنیم.

```
<!ELEMENT note (to,from,date)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
<!ELEMENT date (#PCDATA)>
```

✓تمام قوانین گفته شده در نوشتن DTD داخلی نیز در اینجا رعایت می شود.

✓ سپس در سند XML بعد از اعلان می نویسیم:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
 <from>Jani</from>
<date> 12/11/2002 </date>
</note>
```

✓<!DOCTYPE
✓سپس نام ریشه را می نویسیم.
✓پس از آن می نویسیم SYSTEM
✓و پس از آن آدرس جایی که فایل dtd خود را ذخیره کرده ایم می نویسیم.

DTD خارجی

```
<?xml version="1.0"?>
<!DOCTYPE note PUBLIC “-//liz Castro//DTD End_species//EN//” “http://www.cookwood.com/xml/examples/dtd_creating/end_species.dtd">
<note>
<to>Tove</to>
 <from>Jani</from>
<date> 12/11/2002 </date>
</note>
```

- ✓عبارت SYSTEM که در مثال قبل استفاده شد به این معنا است که ما از یک فایل dtd شخصی استفاده می کنیم که شناخته شده نیست چنانچه بخواهیم از یک dtd عمومی و شناخته شده استفاده کنیم به این صورت عمل می کنیم:

- ✓به جای SYSTEM در مثال قبل می نویسیم PUBLIC
- ✓بعد از آن DTD-name مشخصه عمومی را می نویسیم
- ✓بعد از آن آدرس آن مشخصه عمومی را می نویسیم.

شمای XML

- ✓شمای XML دقیقا به همان منظور تایید صحت سند XML (مانند DTD) تهیه شده است.
- ✓قدرت نمایش و توصیف اطلاعات در شما از DTD بیشتر است.
- ✓نحوه نوشتار یک شمای XML به مانند یک سند XML است و از همان قوانین استفاده می کند.
- ✓با توجه به خصوصیات ذکر شده بیشتر ترجیح داده می شود تا برای توصیف اعتباریک سند از شما استفاده شود تا DTD.

شمای XML

چنانچه بخواهیم برای مثال ساده قبل یک شما بنویسیم به این ترتیب عمل می کنیم:

- ابتدا باید یک فایل **text-only** با پسوند ***.xsd*** ایجاد کنیم .
- پس از آن مانند یک سند **XML** در خط اول آن اعلان **XML** را می نویسیم.
- پس از آن دستوراتی را اضافه می کنیم (به مانند مثال روبرو) که نشان می دهد در توصیف اصطلاحات این زبان از چه فضای نامی استفاده می کنیم.

```
<?xml version="1.0"?>

<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
...
...

</xs:schema>
```

شمای XML

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
<xs:complexType>
....
....
</xs:complexType>
</xs:element>

</xs:schema>
```

- تگی که دارای تگ زیر شاخه باشد به آن **complex** میگوییم و چنانچه تگ زیر شاخه نداشته باشد به آن **simple** می گوییم
- حال برای نمایش شمای مثال پیغام، در ادامه معرفی فضای نام ،تگ اصلی یا همان ریشه را معرفی می کنیم
- ریشه از نوع تگ **complex** است.

شمای XML

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
 <xs:complexType>

<xs:sequence>
....
....
</xs:sequence>

</xs:complexType>
</xs:element>
</xs:schema>
```

✓برای معرفی تگ های زیر شاخه ریشه ، آنها را در تگ sequence تعریف می کنیم.

شمای XML

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com" elementFormDefault="qualified">
<xs:element name="note">
 <xs:complexType>
<xs:sequence>

<xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name=”date" type="xs:date"/>

</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

✓پس از آن تک تک تگ های زیر شاخه را معرفی می کنیم و نام آنها و نوع داده ای که می پذیرند را معرفی می کنیم.

توصیف صفت در شمای XML

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
 <xs:complexType>
<xs:sequence>
<xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>

 <xs:attribute name=”date" type="xs:date"/>

</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

فرض کنید در مثال قبل **date** به جای یک تگ زیر شاخه به صورت یک صفت تعریف شده باشد در این صورت آن را به صورت مقابل نمایش می دهیم.

شمای XML

✓انواع گونه داده ای را که می توان توصیف کرد عبارت است از:

- xs:string ✓
- xs:decimal ✓
- xs:integer ✓
- xs:boolean ✓
- xs:date ✓
- xs:time ✓

شمای XML

✓حال باید در سند XML اصلی خود مشخصات این فایل شما را معرفی کنیم.

✓چنانچه فایل شما را با نام note.xsd ذخیره کرده باشیم در داخل تگ ریشه عبارات زیر را اضافه می کنیم تا هم فایل شما را معرفی کنیم و هم فضای نامی را که شمای ما از آن استفاده می کند.

```
<?xml version="1.0"?>
<note
 xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">

<to>Tove</to>
<from>Jani</from>
<date>12/11/2002 </date>
 </note>
```

نمایش سند XML در وب:

XSLT

CSS

XSLT (EXtensible Stylesheet Language Transformations)

- XSLT سندی است که به فرمت اسناد XML نوشته می شود و وظیفه اصلی آن تبدیل سند XML به HTML و یاXHTML برای نمایش در browser است.

- XSLT با استفاده از XPath(به منظور استخراج هر تگ و مقدارآن و صفتهای مربوط به آن از سند XML استفاده می شود) سند XML را تجزیه می کند و با توجه به دستوراتی که در خود سند XSLT تعریف شده به فرمت مناسب در قالب HTML تبدیل می کند.

در اینجا یک سند XML
نشان داده شده است که
می خواهیم آن را در وب
نمایش دهیم

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Edited with XML Spy v4.4 -->
- <breakfast_menu>
  - <food>
      <name>Belgian Waffles</name>
      <price>$5.95</price>
      <description>two of our famous Belgian Waffles with plenty of real maple syrup</description>
      <calories>650</calories>
    </food>
  - <food>
      <name>Strawberry Belgian Waffles</name>
      <price>$7.95</price>
      <description>light Belgian waffles covered with strawberries and whipped
        cream</description>
      <calories>900</calories>
    </food>
  - <food>
      <name>Berry-Berry Belgian Waffles</name>
      <price>$8.95</price>
      <description>light Belgian waffles covered with an assortment of fresh berries and whipped
        cream</description>
      <calories>900</calories>
    </food>
  - <food>
```

```
<?xml version="١,٠" encoding="ISO-٨٨٥٩-١" ?>
<!--  Edited with XML Spy v٢٤,٢   -->
- <html xsl:version="١,٠" xmlns:xsl="http://www.w٣.org/١٩٩٩/XSL/Transform"
  xmlns="http://www.w٣.org/١٩٩٩/xhtml">
  - <body style="font-family:Arial,helvetica,sans-serif;font-size:١٢pt; background-color:#EEEEEE">
    - <xsl:for-each select="breakfast_menu/food">
      - <div style="background-color:teal;color:white;padding:٤px">
        - <span style="font-weight:bold;color:white">
            <xsl:value-of select="name" />
          </span>
          -
          <xsl:value-of select="price" />
        </div>
      - <div style="margin-left:٢٠px;margin-bottom:١em;font-size:١٠pt">
          <xsl:value-of select="description" />
        - <span style="font-style:italic">
            (
            <xsl:value-of select="calories" />
            calories per serving)
          </span>
        </div>
      </xsl:for-each>
    </body>
</html>
```

در اینجا می توانید ببینید
که مقادیر تگ های
name و **Price**
ازسند **xml** خوانده می
شود و در یک جدول
نمایش داده می شود

یک سند XSLT

XSLT

- پس از تعریف سندXSLT ،آن را با پسوند ***.xsl*** ذخیره می کنیم.

- پس از آن باید در سند XML آن را معرفی کنیم. برای این منظور بعد از دستور اعلان version اضافه می کنیم

?>“<?xml-stylesheet type="text/xsl" href=“xslt_file_name.xsl

- سپس فایل XML را با پسوند ***.xml*** ذخیره می کنیم .

- هنگامیکه بر روی این فایل کلیک کنید این فایل در browser نمایش داده می شود.


D:\Drive E\camputer\term۷\application engineering\xml\New Folder\simplexsl.xml - Microsoft Internet Explor...
www.jozve.org
File Edit View Favorites Tools Help
Back Search Favorites Media
Address D:\Drive E\camputer\term۷\application engineering\xml\New Folder\simplexsl.xml
Go Links
Belgian Waffles - $۵,۹۵
two of our famous Belgian Waffles with plenty of real maple syrup (۶۵۰ calories per serving)
Strawberry Belgian Waffles - $۷,۹۵
light Belgian waffles covered with strawberries and whipped cream (۹۰۰ calories per serving)
Berry-Berry Belgian Waffles - $۸,۹۵
light Belgian waffles covered with an assortment of fresh berries and whipped cream (۹۰۰ calories per serving)
French Toast - $۴,۵۰
thick slices made from our homemade sourdough bread (۶۰۰ calories per serving)
Homestyle Breakfast - $۶,۹۵
two eggs, bacon or sausage, toast, and our ever-popular hash browns (۹۵۰ calories per serving)
نمایش سند XML با
استفاده از XSLT در
browser

XSLT

البته تمامی browser ها قابلیت خواندن فایل XSLT ونمایش سند XML را ندارند.browser هایی که قابلیت نمایش را دارند عبارتند از:

- Firefox 1.0.2✓
- Mozilla 1.7.8✓
- Netscape 8✓
- Opera 8✓
- Internet Explorer 6✓
- Internet Explorer 5✓(البته در این مورد قوانین استفاده از xsltکاملا بر قوانین w3cمنطبق نیست)

CSS (Cascading Style Sheets)

- یکی دیگر از ابزار نمایش سند XML در browser ها CSS است.
- Browser ها هنگامی که یک سند HTML را می خوانند با توجه به فرمت تگ هایی که در آن استفاده شده نحوه نمایش آن را تعیین می کنند
- CSS در واقع نقش تگ های HTML را برای اسناد XML بازی می کند و نحوه نمایش هر یک از داده های سند XML را معین می کند . browser ها با استفاده از سند CSS (به عنوان تعیین چگونگی فرمت نمایش) و سند XML (به عنوان داده هایی که باید نمایش داده شود) اقدام به نمایش آن میکند.

```
<?xml version="١,٠" encoding="ISO-٨٨٥٩-١" ?>
<!--  Edited with XML Spy v٢٠٠٤ rp٢ U (http://www.xmlspy.com)  -->
- <CATALOG>
  - <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>١٠,٩٠</PRICE>
      <YEAR>١٩٨٥</YEAR>
    </CD>
  - <CD>
      <TITLE>Hide your heart</TITLE>
      <ARTIST>Bonnie Tyler</ARTIST>
      <COUNTRY>UK</COUNTRY>
      <COMPANY>CBS Records</COMPANY>
      <PRICE>٩,٩٠</PRICE>
      <YEAR>١٩٨٨</YEAR>
    </CD>
  - <CD>
      <TITLE>Greatest Hits</TITLE>
      <ARTIST>Dolly Parton</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>RCA</COMPANY>
```

در اینجا یک سند XML نشان داده شده است که می خواهیم آن را در وب نمایش دهیم

یک سند CSS

```
CATALOG
{
background-color: #ffffff;
width: ١٠٠%;
}
CD
{
display: block;
margin-bottom: ٣٠pt;
margin-left: ٠;
}
TITLE
{
color: #FF٠٠٠٠;
font-size: ٢٠pt;
}
ARTIST
{
color: #٠٠٠٠FF;
font-size: ٢٠pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #٠٠٠٠٠٠;
margin-left: ٢٠pt;
}
```

در اینجا نشان داده شده
که هر یک از تگ های
سند XML به چه
صورتی نمایش داده
شوند

CSS

- پس از تعریف سند CSS ،آن را با پسوند ***.css*** ذخیره می کنیم.

- پس از آن باید در سند XML آن را معرفی کنیم. برای این منظور بعد از دستور اعلان version اضافه می کنیم

```
<?xml-stylesheet type="text/css" href=”css_file_name.css"?>
```

- سپس فایلXML را با پسوند ***.xml*** ذخیره می کنیم .

- هنگامیکه بر روی این فایل کلیک کنید این فایل در browser نمایش داده می شود.

D:\Drive E\camputer\termV\application engineering\xml\New Folder\cd_catalog_with_css.x

File Edit View Favorites Tools Help

Back Search Favorites Media

Address D:\Drive E\camputer\termV\application engineering\xml\New Folder\cd_catalog_with_css.xml

Empire Burlesque Bob Dylan
USA
Columbia
۱۰,۹۰
۱۹۸۵

Hide your heart Bonnie Tyler
UK
CBS Records
۹,۹۰
۱۹۸۸

Greatest Hits Dolly Parton
USA
RCA
۹,۹۰
۱۹۸۲

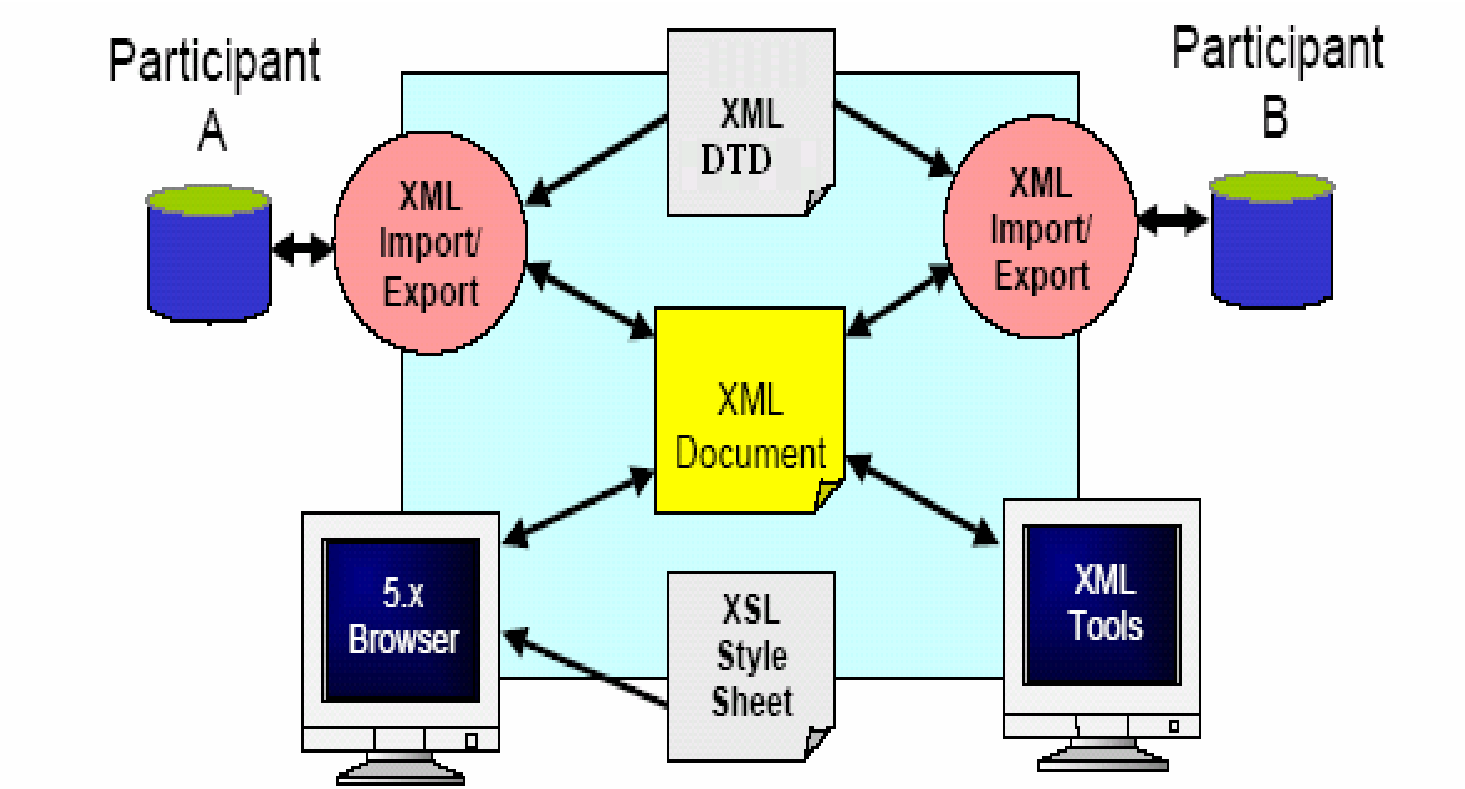
نمایش سند XML با استفاده از CSS در browser

CSS

البته تمامی browser ها قابلیت خواندن فایل CSS ونمایش سند XML را ندارند.browser هایی که قابلیت نمایش را دارند عبارتند از:

- ✓ Firefox 1.0.2
- ✓ Mozilla 1.7.8
- ✓ Opera 8
- ✓ Internet Explorer(البته این مورد نمی تواند تمامی دستورات CSS را اجرا کند)

نحوه ارتباط اجزا اساسی برای تهیه و نمایش یک سند XML



مدیریت تراکنش
(مفاهیم)

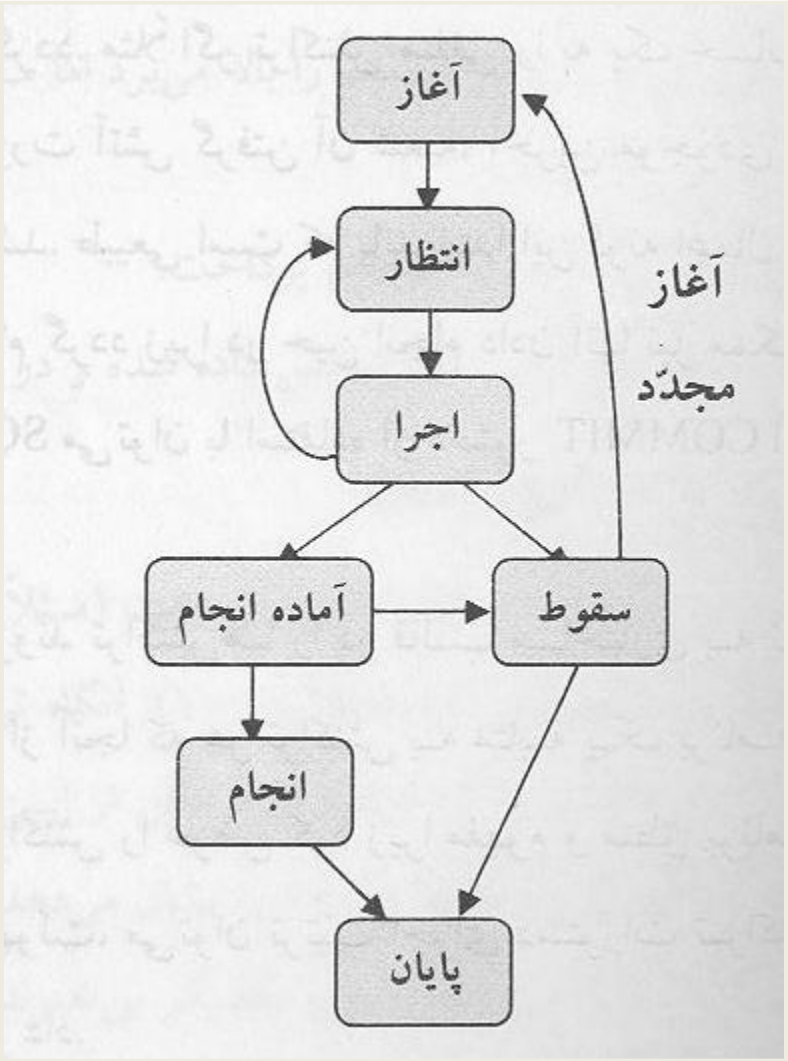
مفهوم تراکنش

- مجموعه ای از عملگرهای بانک اطلاعات که از دید کاربر یک واحد منطقی کار را تشکیل می دهد را تراکنش می گوییم.

```
begin T1:
        read(A);
        A:=A-50;
        write(A);
        read(B);
        B:=B+50;
        write(B);
end T1
```

- برای اینکه تراکنش جامعیت بانک اطلاعاتی را حفظ نماید، باید چهار خاصیت ACID را رعایت نماید.

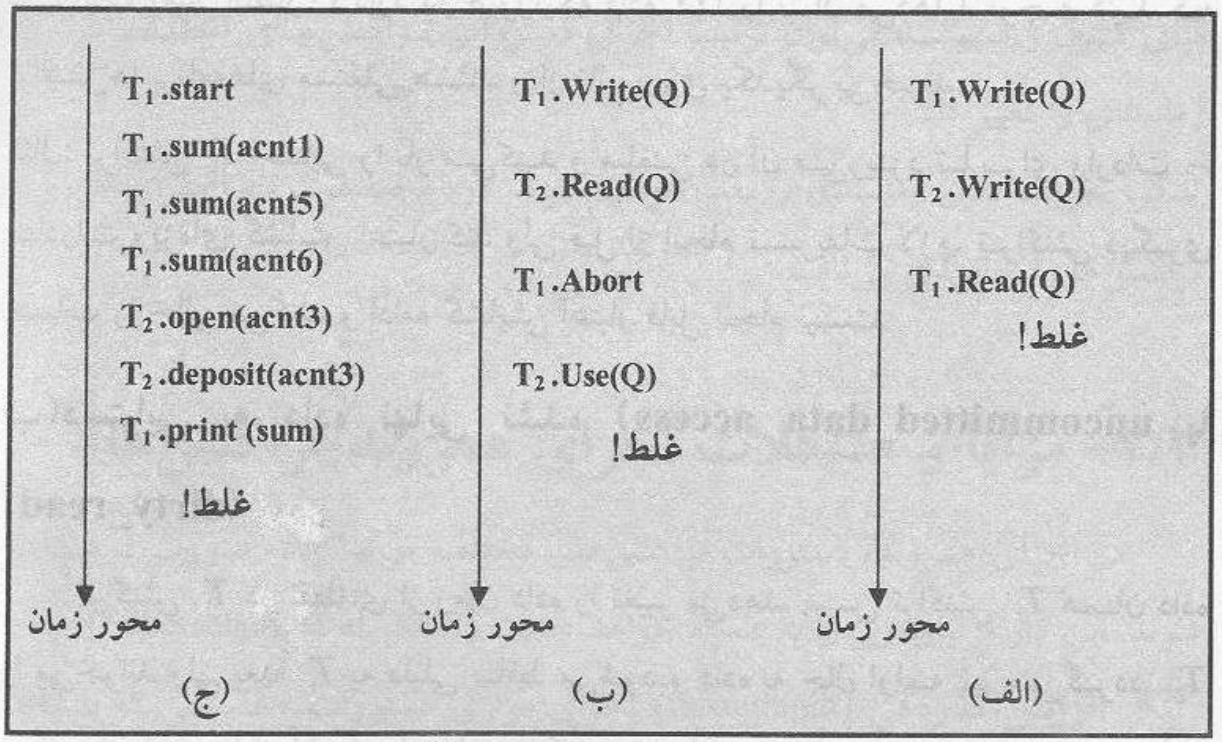
مراحل اجرای تراکنش



مزایای اجراء همروند تراکنش ها

- **افزایش گذردهی (troughput)**: اجراء موازی دستورات که با پردازنده درگیر هستند، با دستورات ورودی خروجی، می تواند تعداد تراکنش های اجراء شده در واحد زمان را افزایش دهد.
- **کاهش میانگین زمان پاسخ دهی**: دیگر تراکنش های با زمان اجراء کوتاه منتظر به اتمام رسیدن تراکنش های بلند مدت نمی مانند.

مشکلات اجراء همروند تراکنش ها



انواع مشکلات همروندی الف) تغییرات گم‌شده ب) دستیابی به داده نهائی نشده ج) بازیابی ناهمگام

شرایط درستی یک طرح همروند

- پی در پی پذیر باشد.
  - CSR –
  - VSR –
- ترمیم پذیر باشد.
  - RC –
  - ACA –
  - ST –

برخورد (conflict)

چنانچه p و q دستورات دو تراکنش مختلف باشند، می گوییم این دو دستور با یکدیگر برخورد دارند اگر:

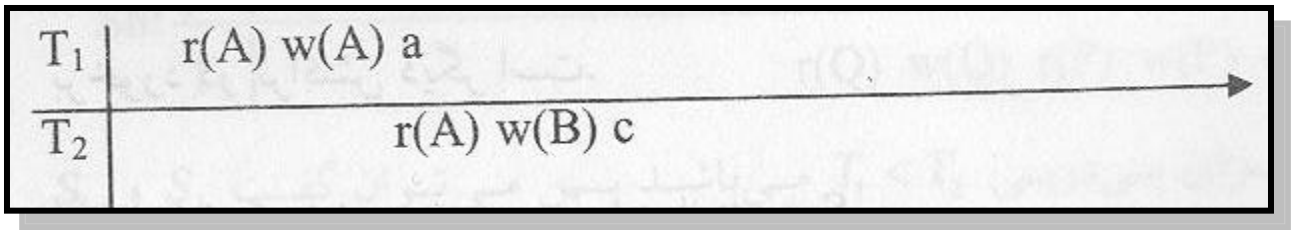
۱. این دستورات از تراکنش های مختلف باشند.
۲. هر دو دستور به یک داده مشترک دسترسی داشته باشند.
۳. حداقل یکی از این دو دستور نوشتن (write) باشد.

جدول برخورد

T_i / T_j	$r_i(Q)$	$w_i(Q)$
$r_j(Q)$	سازگار	ناسازگار
$w_j(Q)$	ناسازگار	ناسازگار

زمانبندی پی در پی

زمانبندی را پی در پی گوییم اگر پایان یکی قبل از شروع دیگری باشد.

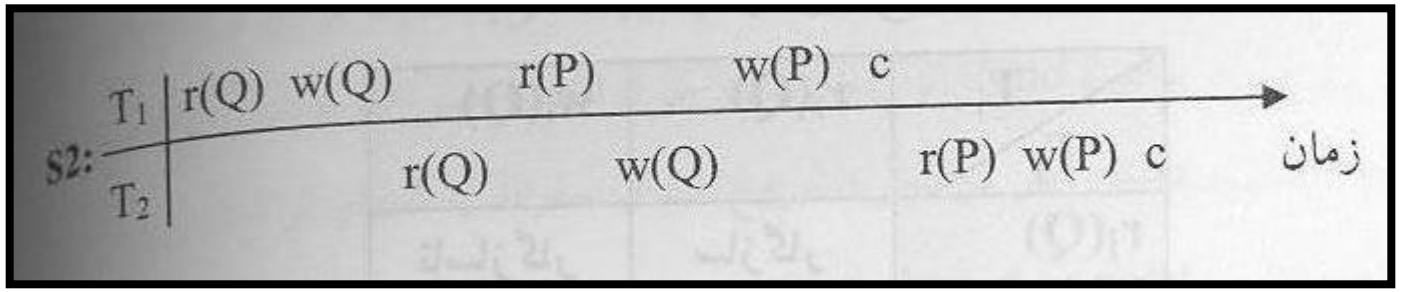


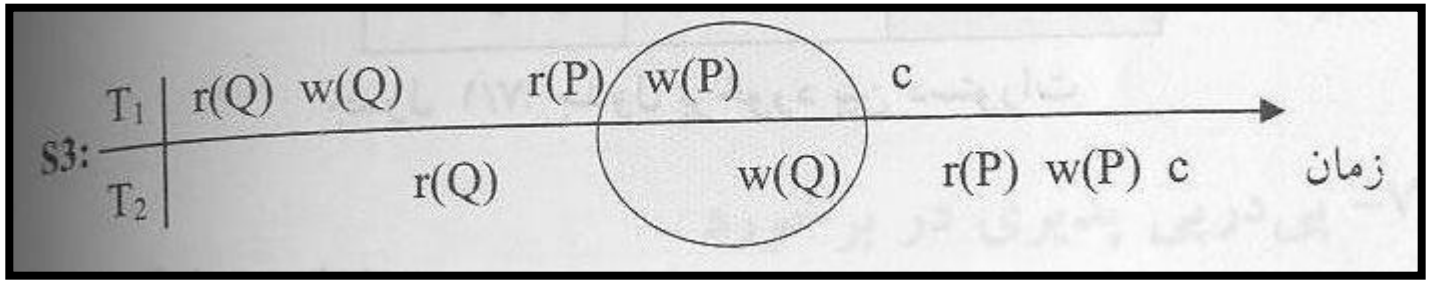
$T_1 < T_2$

زمانبندی معادل در برخورد

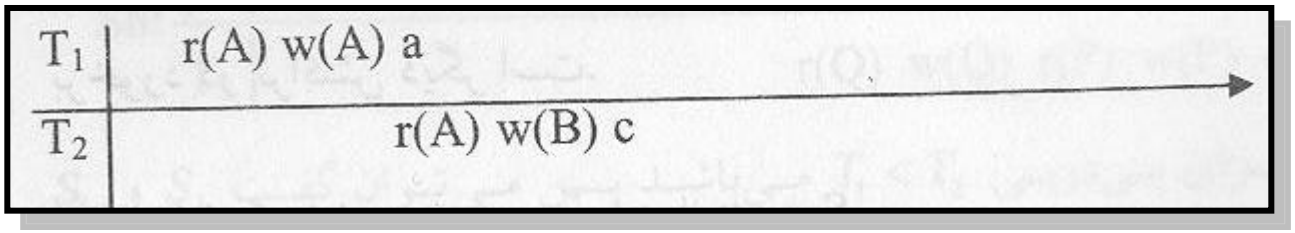
- زمانبندی S2 را معادل در برخورد S1 می گوییم اگر هر دو بر روی یک مجموعه از دستورات و تراکنش ها کار کنند و با جابجا کردن دستورات بدون برخورد در S1 بتوانیم به زمانبندی S2 برسیم.

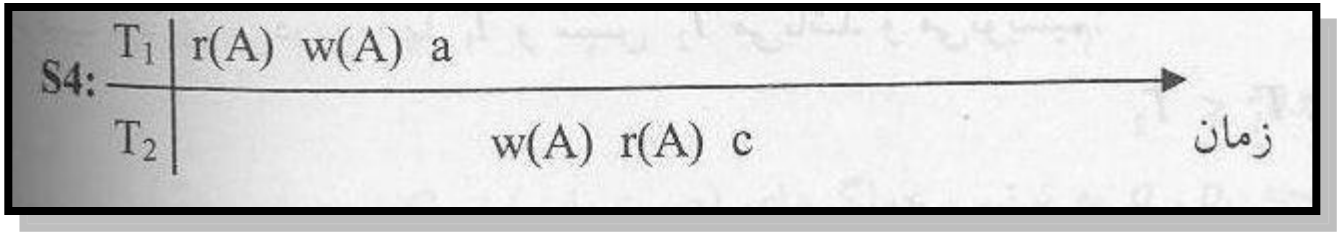
مثال از معادل در برخورد





مثال غلط از معادل در برخورد



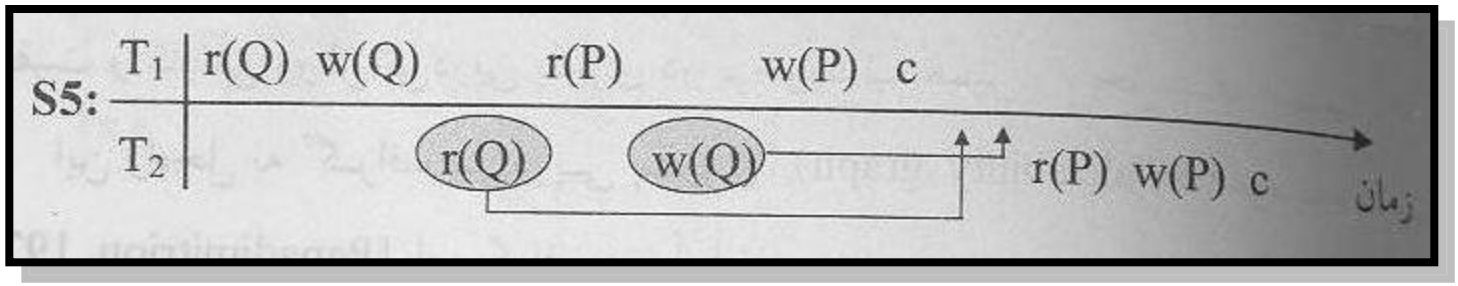


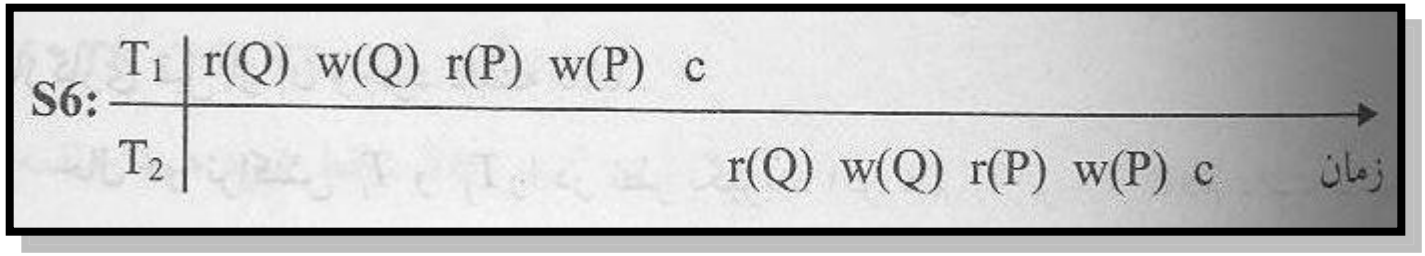
پی در پی پذیر در برخورد (CSR)

- زمانبندی را پی در پی پذیر در برخورد می گوییم اگر معادل در برخورد یک زمانبندی پی در پی باشد.

- زمانبندی همروند به شرط آنکه معادل در برخورد یک زمانبندی پی در پی باشد، مشکل همروندی ندارد.

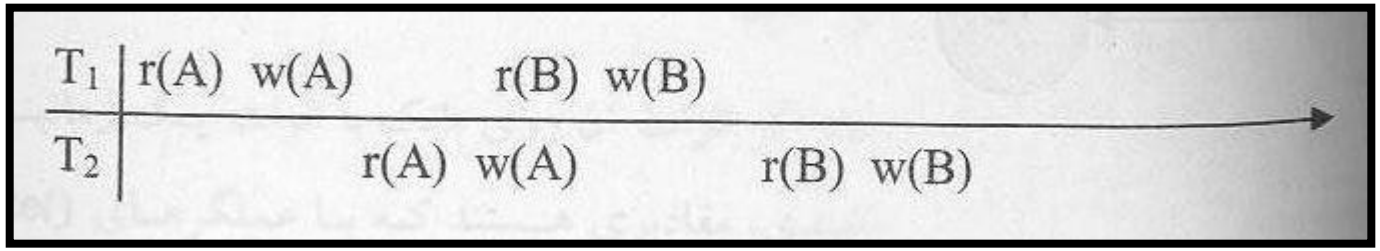
مثال از پی در پی پذیر در برخورد





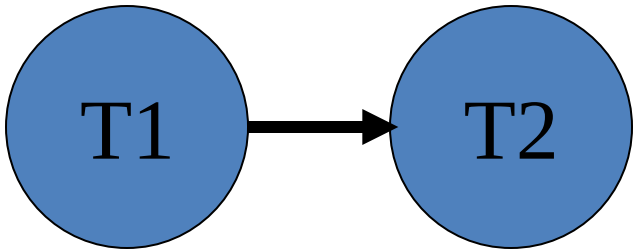
مثال از پی در پی پذیر در برخورد

آیا زمانبندی زیر پی در پی پذیر در برخورد است؟



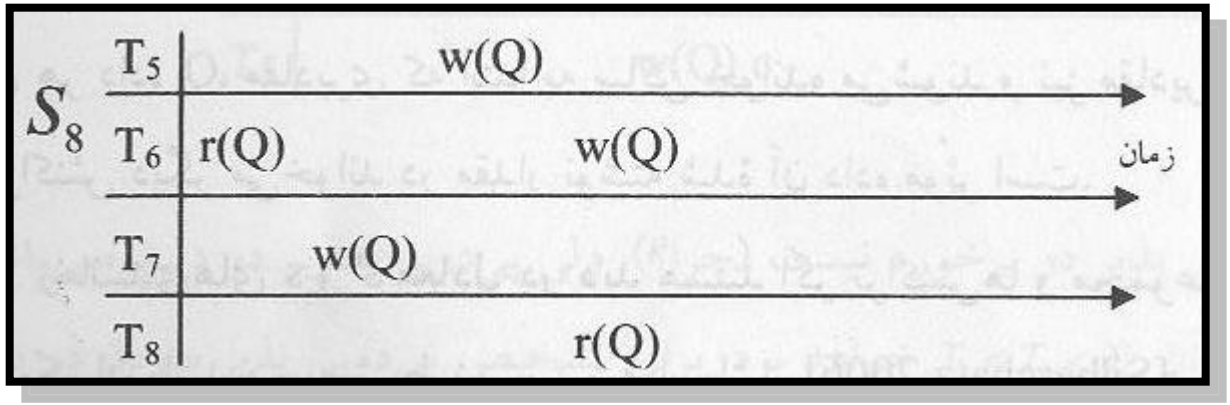
تشخیص پی در پی پذیری در برخورد

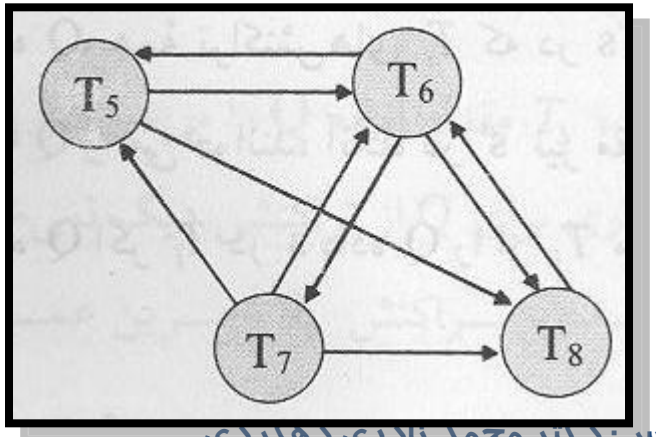
- تاکنون راه حل عملی تشخیص پی در پی پذیری ارائه نشده است.
- راه حل **گراف پی در پی پذیری** بدین منظور ارائه شده است. این گراف مرتباً با تغییر مجموعه دستورات تراکنش ها در زمانبندی بهنگام می شود.
- در صورتیکه در گراف حلقه مشاهده شود، تراکنش (ها) ساقط می گردند. در غیر اینصورت به کار خود ادامه می دهند.
- تراکنش ها رئوس گراف را تشکیل می دهند. یالهای این گراف جهت دار هستند و به معنای وجود دستورات برخورد دار از یک تراکنش به سوی دیگری است.



مثال از بکارگیری گراف پی در پی پذیری

گراف پی در پی پذیری مربوط به زمانبندی زیر را ترسیم نموده و تشخیص دهید که پی در پی پذیر است یا خیر؟





abort

T6 → $T_7 < T_5 < T_8$

پی در پی پذیر در دید (VSR)

- یک زمانبندی پی در پی پذیر است اگر اثرات آن روی بانک با اثرات یک زمانبندی پی در پی معادل باشد.
- اثرات مقادیری هستند که با عملگرهای write() توسط تراکنش های ساقط نشده نوشته می شوند.
- برای یکسان بودن اثر نهایی لازم است آخرین تراکنشی که آن داده را می نویسد، در هر دو زمانبندی یکسان باشد.
- برای یکسان بودن مقادیر نوشته شده هم باید مقادیری که تراکنش می خواند یکسان باشد.

زمانبندی معادل در دید

۱) برای هر داده Q، همهٔ تراکنش‌های T_i که در s مقدار اولیه (مقدار قبل از اولین نوشتن) دادهٔ Q را می‌خوانند، آنگاه در 's نیز مقدار اولیه Q را بخوانند.

۲) برای هر داده Q اگر T_i در s داده Q را از T_j می‌خواند، در 's نیز داده Q را از T_j بخواند.

۳) برای هر داده Q، آخرین تراکنشی از زمانبندی s که روی Q می‌نویسد، همان تراکنشی باشد که در زمانبندی 's آخرین بار روی Q می‌نویسد.

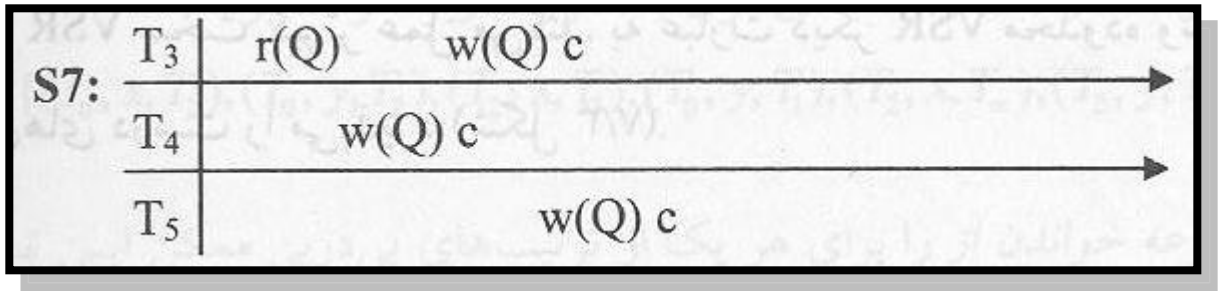
با اعمال این شرط‌ها اولاً تضمین می‌کنیم که هر تراکنش در هر دو زمانبندی مقادیر یکسانی را بخواند. ثانیاً با تضمین یکسان بودن مقادیر نوشته شده روی هر داده، اطمینان حاصل می‌شود که وضعیت نهایی بانک‌اطلاعات در هر دو زمانبندی یکسان است.

تعریف پی در پی پذیر در دید

زمانبندی پی در پی پذیر در دید است اگر معادل در دید یک زمانبندی پی در پی باشد.

مثال از پی در پی پذیر در دید

- آیا زمانبندی زیر پی در پی در برخورد است؟ پی در پی پذیر در دید چطور؟



پی در پی پذیر در برخورد نیست ولی در دید است. $T_3 < T_4 < T_5$

زیرا:

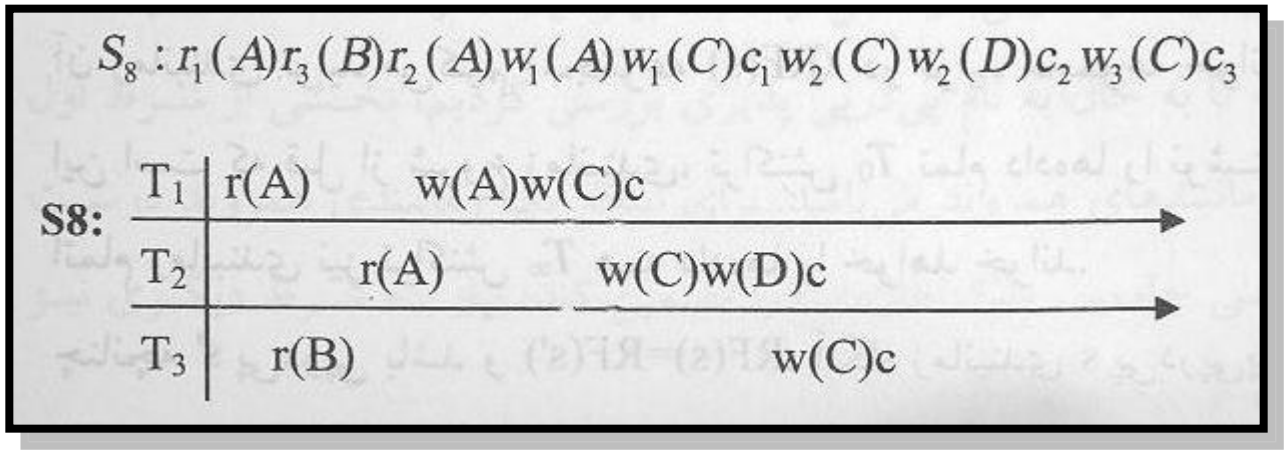
۱- در هر دو زمانبندی T3 مقدار اولیه Q را می خواند.

۲-هیچ تراکنشی مقداری برای Q از دیگری نمی خواند.

۳-در هر دو زمانبندی T5 آخرین عمل نوشتن بر روی Q را انجام می دهد.

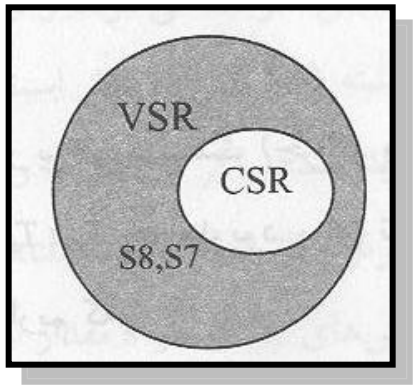
مثال دیگر از پی در پی پذیر در دید

- آیا زمانبندی زیر CSR است یا VSR؟



نکات مهم

- هر زمانبندی که CSR نباشد ولی VSR باشد در آن Blind Write رخ داده است. (بدون آنکه داده را قبلاً خوانده باشد در آن می نویسد).
- هر زمانبندی پی در پی پذیر در برخورد حتماً پی در پی پذیر در دید است. عکس این موضوع الزاما برقرار نیست.
- الگوریتم شناسایی پی در پی پذیری در دید از پیچیدگی بالایی برخوردار است. بهمین دلیل بکارگیری روش CSR بسیار فراگیر شده است.



تشخیص پی در پی پذیری در دید

- از نماد Read From به صورت زیر استفاده می کنیم:

(T_j, x, T_i)

(تراکنش T_i داده x را از تراکنش T_j می خواند)

- مجموعه RF(s) را برای همه داده های s تولید می کنیم با فرض اولیه زیر:

تشخیص پی در پی پذیری در دید

- اگر S2 پی در پی باشد و RF(S1)=RF(S2) باشد، آنگاه زمانبندی S1 پی در پی پذیر در دید است.

مثال از پی در پی پذیر در دید

- آیا زمانبندی زیر پی در پی پذیر در دید است؟

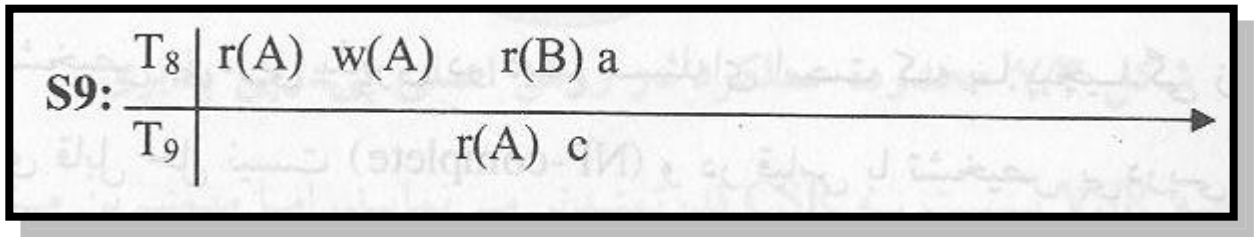
$$s = r_2(x), w_2(x), r_1(x), r_1(y), r_2(y), w_2(y), c_1, c_2$$

$$RF(s) = \{(T_0, x, T_2), (T_0, y, T_2), (T_2, x, T_1), (T_0, y, T_1), (T_2, x, T_\infty), (T_2, y, T_\infty)\}$$

$$RF(T_1 < T_2) = \{(T_0, x, T_1), (T_0, y, T_1), (T_0, x, T_2), (T_0, y, T_2), (T_2, x, T_\infty), (T_2, y, T_\infty)\}$$
$$RF(T_2 < T_1) = \{(T_0, x, T_2), (T_0, y, T_2), (T_2, x, T_1), (T_2, y, T_1), (T_2, x, T_\infty), (T_2, y, T_\infty)\}$$

ترمیم پذیری

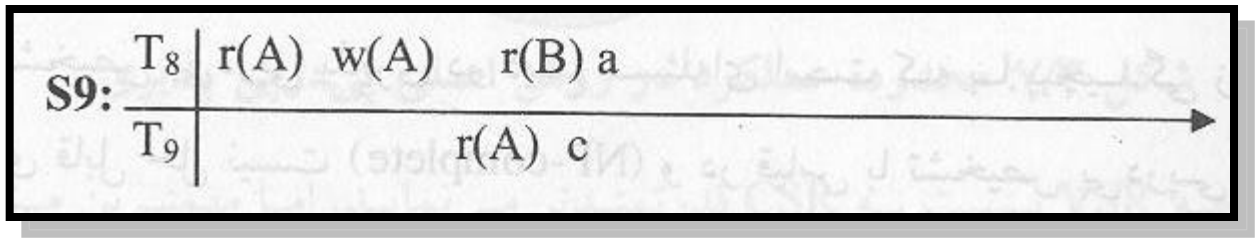
- در سناریو هایی نظیر دسترسی به داده تثبیت نشده تکلیف تراکنشی که با داده غیر معتبر درگیر شده است چیست؟
- برای درستی فرآیند همزمانی، یک زمانبندی علاوه بر پی در پی پذیری باید ترمیم پذیر نیز باشد.



الف– زمانبندی ترمیم پذیر (recoverable-RC)

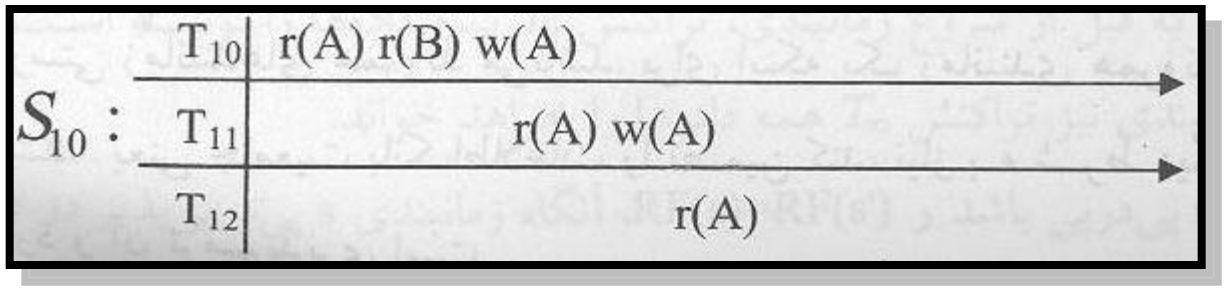
- اگر برای تمام تراکنش های T_j که از T_i می خوانند، تثبیت تراکنش T_i قبل از تثبیت تراکنش T_j صورت پذیرد.

آیا زمانبندی زیر ترمیم پذیر است؟



مثال از ترمیم پذیری RC

آیا زمانبندی زیر RC است؟ چه مشکلی می تواند داشته باشد؟



مشکل زمانبندی ترمیم پذیر

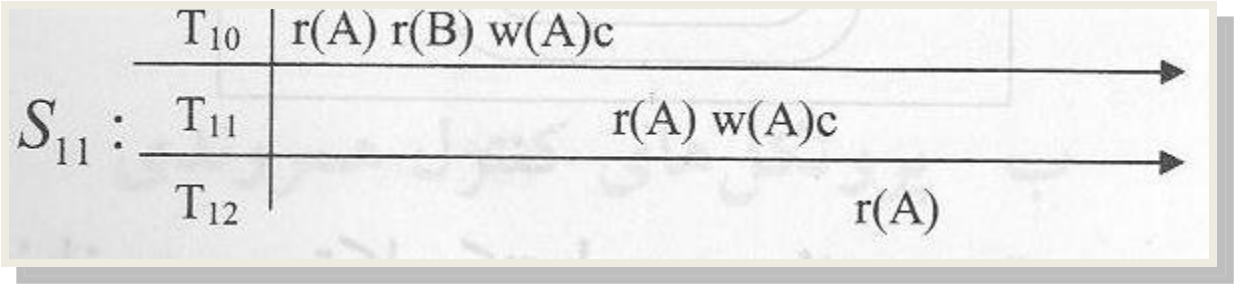
- این زمانبندی ممکن است دچار سبب ساقط شدن تراکنش های دیگر بصورت آبشاری (Cascading Aborts) گردد.

ب– زمانبندی فاقد سقوط های آبشاری (Avoiding Cascading Aborts-ACA)

- اگر برای هر دو تراکنش T_i و T_j، اگر تراکنش T_j که از T_i بخواند، آنگاه تراکنش T_i قبل از خواندن تراکنش T_j تثبیت یا ساقط گردد.
- هر زمانبندی RC، ACA نیز هست.

مثال از ترمیم پذیری ACA

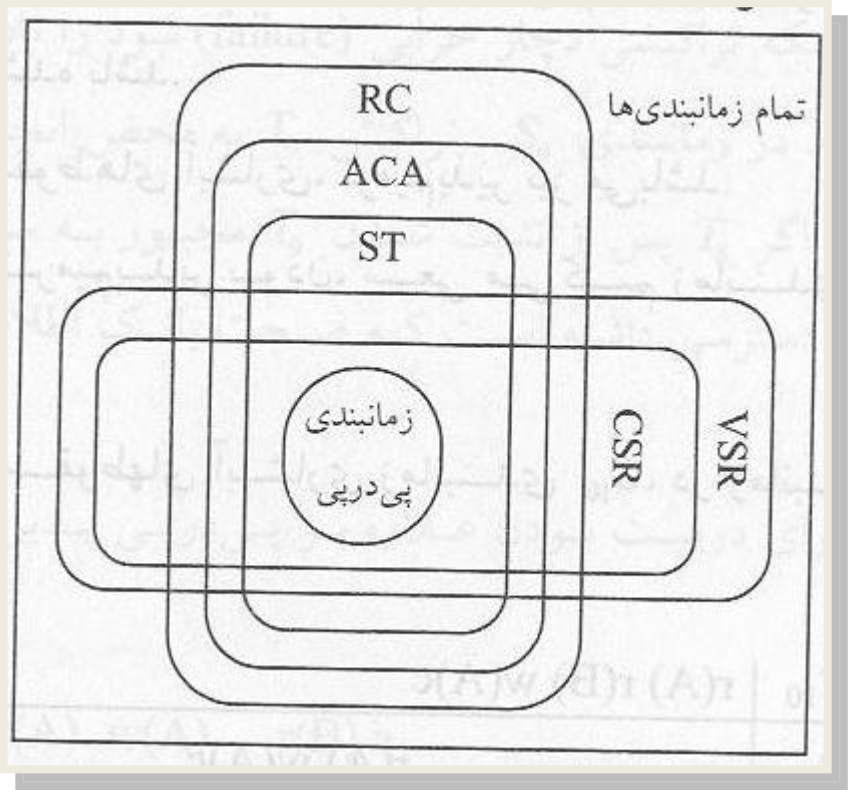
آیا زمانبندی زیر **ACA** است؟



ج– زمانبندی محض (سختگیر) (Strict-ST)

- اگر برای هر دو تراکنش T_i و T_j، اگر تراکنش T_j که از T_i بخواند یا **بنویسد**، آنگاه تراکنش T_i قبل از خواندن یا **نوشتن** تراکنش T_j تثبیت یا ساقط گردد.

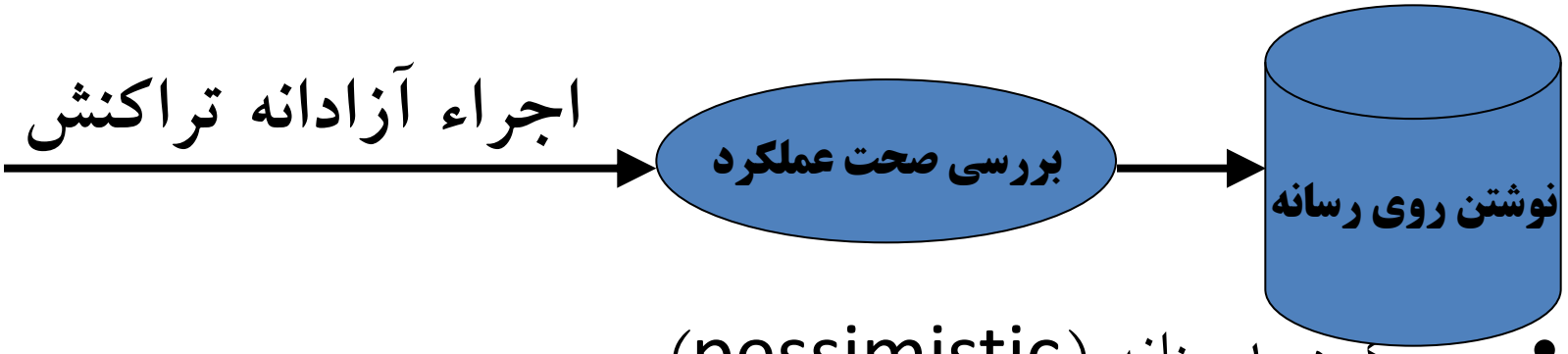
طرحهای مختلف پی در پی پذیری و ترمیم پذیری زمانبندی ها



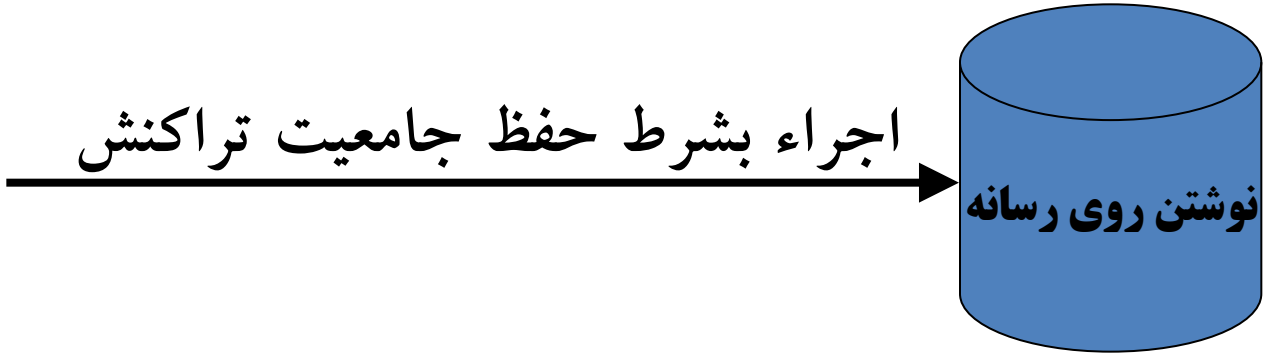
مدیریت تراکنش (پروتکل های همروندی)

رویکردهای زمانبندی تراکنش ها

- رویکرد خوش بینانه (optimistic)



- رویکرد بدبینانه (pessimistic)



تصمیم گیری پیرامون هر عملگردر رویکرد بدبینانه

۱. اجراء عملگر (execute)
۲. به تأخیر اندازی اجراء عملگر و قرار دهی آن در صف (delay)
۳. عدم پذیرش عملگر منجر به ساقط شدن تراکنش می گردد. (reject)

انواع زمانبند ها

- **زمانبند محافظه کار (conservative)**

در این زمانبندی اگر لازم باشد اجراء دستورات تراکنش را به تعویق می اندازد و محتاطانه و محافظه کارانه اجراء دستورات را پی می گیرد تا حتی الامکان تراکنش ها ساقط نشوند.

- **زمانبند بی پروا (aggressive)**

در این زمانبندی هدف پرهیز از تأخیر در اجرای دستورات است. بنابراین در این زمانبندی دستورات فوراً اجراء می شوند و در صورت بروز مشکلی مجبور به ساقط کردن برخی تراکنش هاست.

پروتکل های مبتنی بر قفل (lock-based)

- این پروتکل ها از کاربردی ترین روشهای کنترل همروندی است.
- در این روش به کمک تخصیص داده به تراکنش ها، در زمان خواندن و نوشتن یک داده، ابتدا درخواست قفل مناسب از مدیر قفل (lock manager) می شود.
- مدیر قفل درخواست مورد نظر را با قفل های زده شده از جانب تراکنش های دیگر، مقایسه می کند:
  - در صورت سازگاری با قفلهای پیشین، قفل گذاشته می شود.
  - در صورت عدم سازگاری تراکنش به حالت انتظار می رود تا زمانیکه قفل های زده شده بگونه ای آزاد شوند که قفل درخواستی مجاز باشد.

سازگاری قفل ها

- انواع قفل
  - **قفل دو حالتی (binary)**
    - در این حالت داده یا قفل است یا باز. اشتراک داده وجود ندارد و درخواست تراکنش ها فقط در صورت باز بودن قفل داده پذیرفته می شود.
  - **قفل اشتراکی-انحصاری (shared-exclusive)**
    - به منظور افزایش سطح همروندی تراکنش ها و امکان به اشتراک گذاری داده ها در حالات مختلف قفل ها به دو نوع اشتراکی (S) و انحصاری (X) تقسیم می شوند.

قفل اشتراکی

- از این قفل برای خواندن داده (read) مورد استفاده قرار می گیرد.
- مدیر قفل به همه تراکنش ها اجازه قفل اشتراکی و خواندن داده را می دهد.

قفل انحصاری

- از این قفل برای نوشتن داده (write) مورد استفاده قرار می گیرد.
- داده ای با قفل انحصاری تنها در اختیار یک تراکنش قرار دارد و تراکنش های دیگری به هیچ وجه نمی توانند تا باز شدن این قفل به آن داده دسترسی داشته باشند.

جدول سازگاری قفلهای S و X

$i \neq j$	S_j	X_j
S_i	سازگار	ناسازگار
X_i	ناسازگار	ناسازگار

نکاتی در زمینه قفل گذاری (۱)

- هر تراکنش پیش از r و w باید درخواست قفل مربوطه را به مدیر قفل بدهد. اگر این درخواست پذیرفته شد (در صورت سازگاری) در این صورت تراکنش می تواند r و w را انجام دهد.

- اگر درخواست قفل پذیرفته نشد، تراکنش به حالت انتظار می رود تا قفل های روی آن داده آزاد شود و حالت سازگار پیش آید.

- همزمان چند تراکنش می توانند یک داده را از نوع s قفل کرده و همزمان بخوانند. اما چنانچه تراکنشی داده ای را از نوع x قفل کند هیچ تراکنشی دیگر هیچ نوع قفلی نمی تواند به آن بزند.

نکاتی در زمینه قفل گذاری (۲)

- اگر پس از استفاده از داده فوراً قفل باز شود، ممکن است مشکلاتی پدید آید. بنابراین باز کردن قفل از قواعد خاصی پیروی می کند.
- قفل گذاری می تواند با دانه بندی های (granularity) مختلف از نظر اندازه صورت پذیرد. قفل روی صفت، جدول، صفحه یا کل بانک.
- قفل های با دانه بندی ریزتر (fine grain) باعث افزایش درجه همروندی شده و نیز سربار می شود و قفل های با دانه بندی درشت تر (coarse grain) منجر به کاهش همروندی و سربار می گردد.

نکاتی در زمینه قفل گذاری (۳)

- از جدول قفل (lock table) برای پیاده سازی مدیر قفل استفاده می شود. در این جدول در هر لحظه وضعیت استفاده تراکنش ها از داده ها، نوع قفل ها، اشتراک داده ها و غیره مورد استفاده قرار می گیرد.

- صرف استفاده از قفل گذاری برای تضمین درستی زمانبندی کافی نیست. قفل ها باید بطور مناسب بکار برده شوند.

مثال

- معادل زمانبندی زیر را با قفل های اشتراکی و انحصاری بنویسید:

$$S_1 : r_1(A)\, w_1(A)\, a_1\, w_2(A)\, w_2(B)\, c_2$$

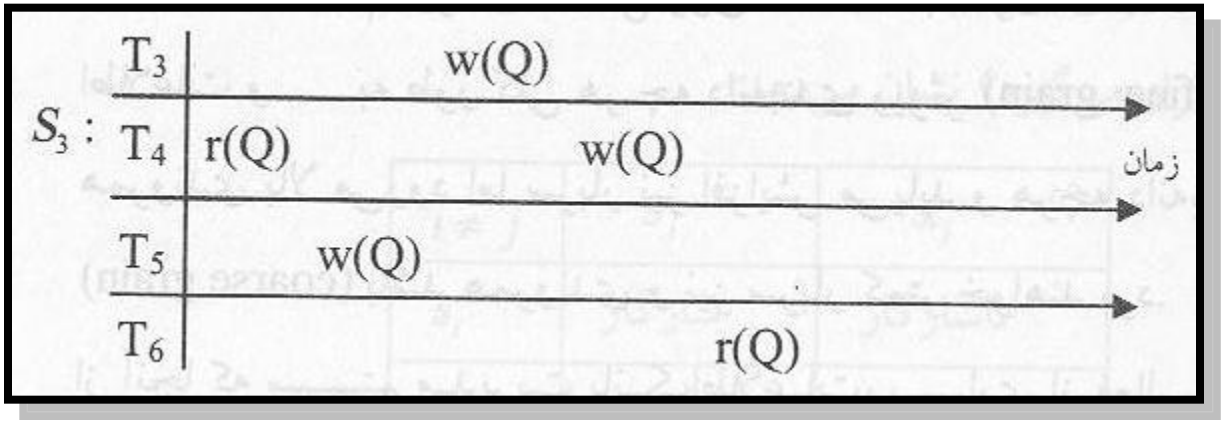
$$S_2 : s_1(A)\, r_1(A)\, x_1(A)\, w_1(A)\, a_1\, u_1(A)\, x_2(A)\, w_2(A)\, x_2(B)\, w_2(B)\, u_2(A)\, u_2(B)\, c_2$$

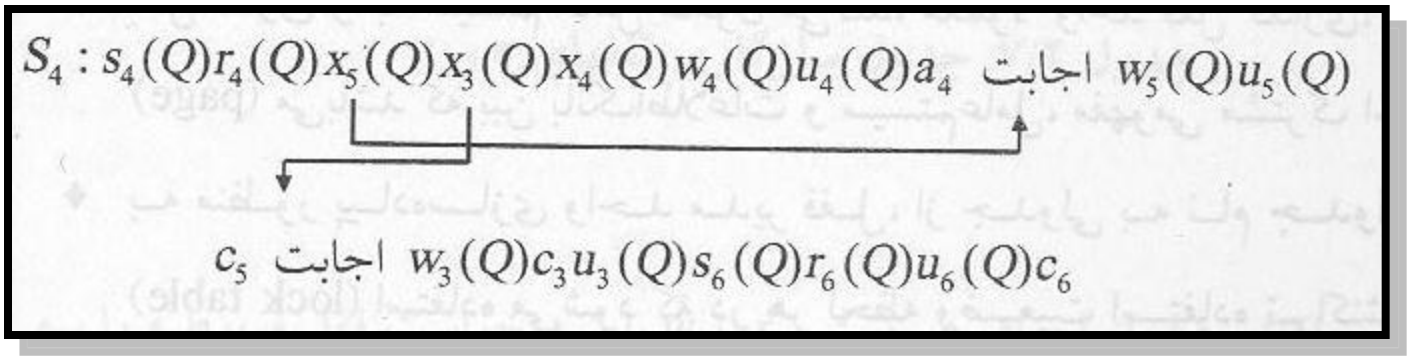
S_2	t_1	$s(A), r(A), x(A), w(A), u(A), a$
	t_2	$x(A), w(A), x(B), w(B), u(A), u(B), c$

باز کردن قفل قواعدی دارد که در ادامه به آن خواهیم پرداخت.

مثال

- معادل زمانبندی زیر را با قفل های اشتراکی و انحصاری بنویسید:





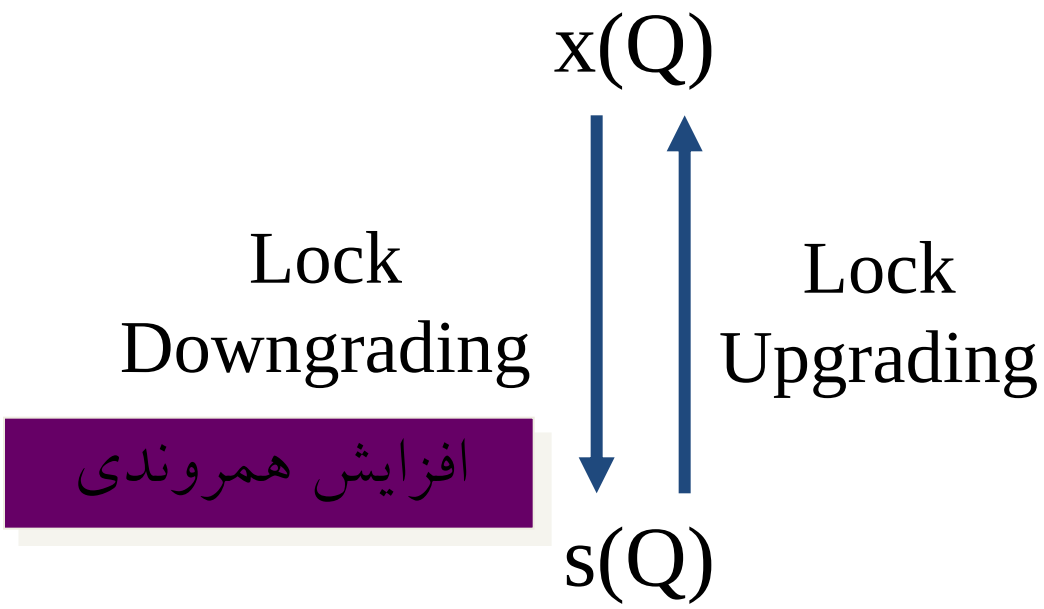
نکات دیگر در زمینه قفل گذاری

- تثبیت یا سقوط تراکنش ها به عواملی خاص بستگی دارد که ارتباطی با قفل گذاری ندارد.
- در دو مثال قبل برای باز کردن قفل ها محدودیتی اعمال نگردید. در این حالت ممکن است مشکل بازیابی ناهمگام رخ دهد.
- یک تراکنش باید در صورت لزوم قفل خود را تبدیل کند.

نکات دیگر در زمینه قفل گذاری

- یک تراکنش **باید** در صورت لزوم قفل خود را تبدیل کند:
  - اگر تراکنشی قفل s(Q) دارد و بعداً عمل w(Q) میخواهد انجام دهد باید تقاضای x(Q) بدهد. **(این درخواست ممکن اجابت نشود اگر تراکنش های دیگر هم Q را قفل اشتراکی کرده باشند)**
  - قفل x(Q) برای خواندن باید به قفل r(Q) تبدیل شود تا دیگران هم بتوانند بخوانند.

تبدیل قفل ها به یکدیگر



بن بست (Deadlock) و قحطی (Starvation)

- زمانبندی زیر را در نظر بگیرید:

T_7	$x(B)w(B)$	$x(A) \rightarrow$ انتظار
T_8	$s(A)r(A)s(B) \rightarrow$ انتظار	

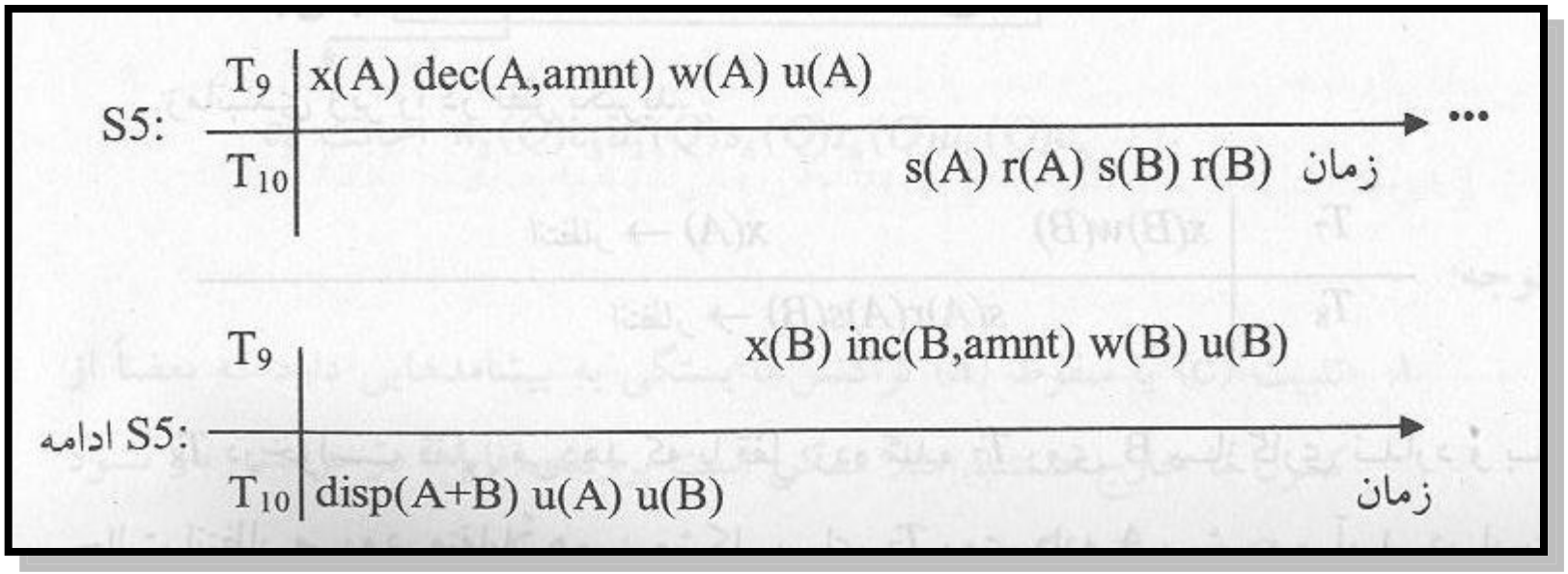
- هر یک از T7 و T8 منتظر دیگری هستند تا منبع مورد نیازشان را آزاد کند.
- در واقع یک انتظار چرخشی میان تراکنشها بوجود آمده است که هیچکدام نمی توانند کار خود را انجام دهند.
- به این انتظار چرخشی **بن بست** می گوییم.

بن بست (Deadlock) و قحطی (Starvation)

- یک راه حل بن بست آن است که یکی از این تراکنشها ساقط شود تا قفل هایش آزاد شده و دیگری بتواند با زدن قفل مورد نظر، به کارش ادامه دهد.
- ولی راه حل فوق ممکن است باعث بروز **قحطی** شود.
- برای مثال یک تراکنش که می خواهد قفل X را روی داده ای بزند، منتظر دنباله ای بی پایان از تراکنشهایی شود که همگی می خواهند قفل S روی همان داده بزنند و تراکنش مورد نظر تا زمان نامعلومی باید منتظر بماند. تراکنش فوق دچار **قحطی** می شود.

پروتکل های قفل دو مرحله ای (2PL)

- فرض کنید در زمانبندی زیر، تراکنش T_9 مبلغی از حساب بانکی A به حساب بانکی B منتقل می کند. تراکنش همروند T_{10} نیز جمع موجودی دو حساب را بدست می آورد:



- علیرغم رعایت همه قوانین قفل گذاری ولی حاصل نهایی T_{10} صحیح نیست.

مشکل کجاست؟

- مشکل از آنجا ناشی می شود که در جدول سازگاری قفل ها، فقط بر روی داده مشترک تمرکز شده است.
- این تمرکز، تنها مشکلات همروندی تغییرات گم شده (Lost Updates) و دستیابی به داده نهایی نشده (Uncommited Data Access-Dirty Read) را رفع می کند و مشکل بازیابی ناهمگام (Inconsistent Retrieval-Phantom Problem) برطرف نمی شود.

راه حل چیست؟

- برای رفع این مشکل پروتکل های قفل گذاری معرفی شده اند.
- این پروتکل ها هم برای قفل گذاری و هم بازکردن قفل ها استفاده می شوند.
- تراکنش ها اجازه ندارند بمحض اتمام کارشان با یک داده قفل های آن را باز کنند!

پروتکل قفل دو مرحله ای

این پروتکل تضمین می کند که زمانبندی پی در پی پذیر در برخورد باشد و هر سه مشکل همروندی را نیز حل کند.
این پروتکل شامل دو مرحله است:

۱. مرحله اول (مرحله رشد-growing)
   - در این مرحله تراکنش فقط می تواند قفل بگیرد و با داده کار کند ولی نمی تواند قفلی را آزاد کند.

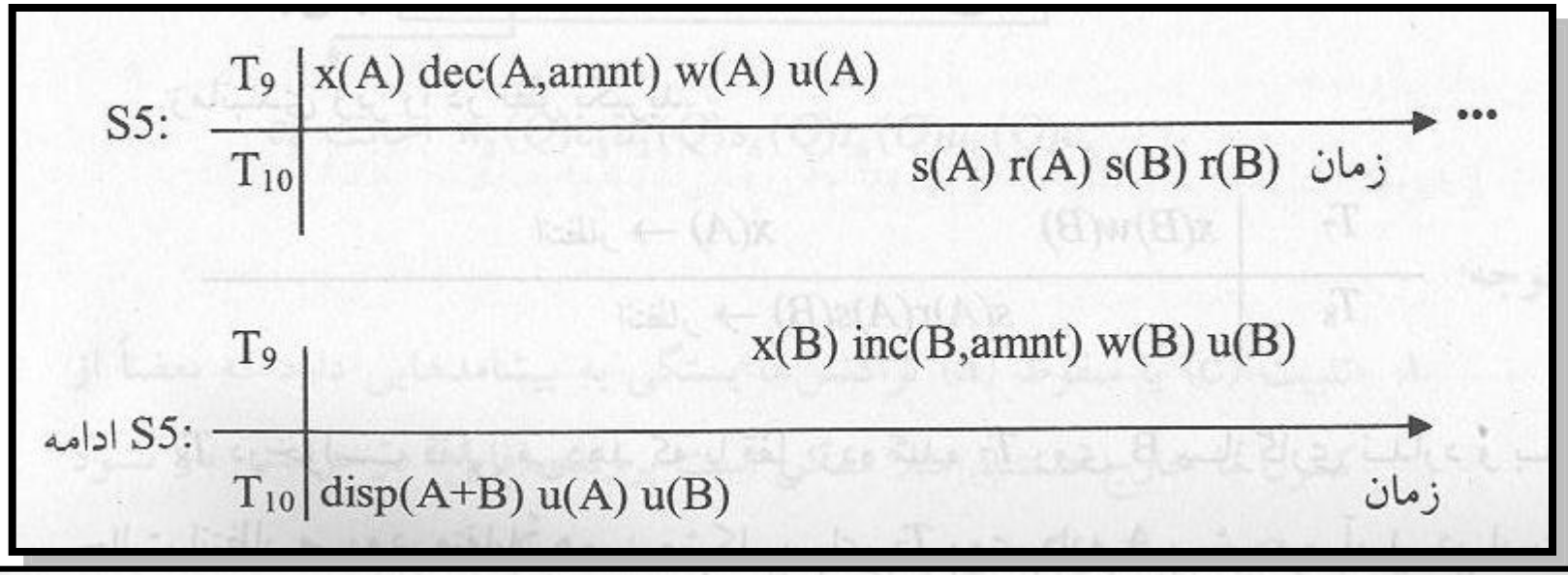
۲. مرحله دوم (مرحله نقصان یا عقب نشینی shrinking)
   - در این مرحله تراکنش فقط می تواند قفل آزاد کند و با داده کار کند ولی نمی تواند قفل جدیدی را روی داده ای بگذارد.

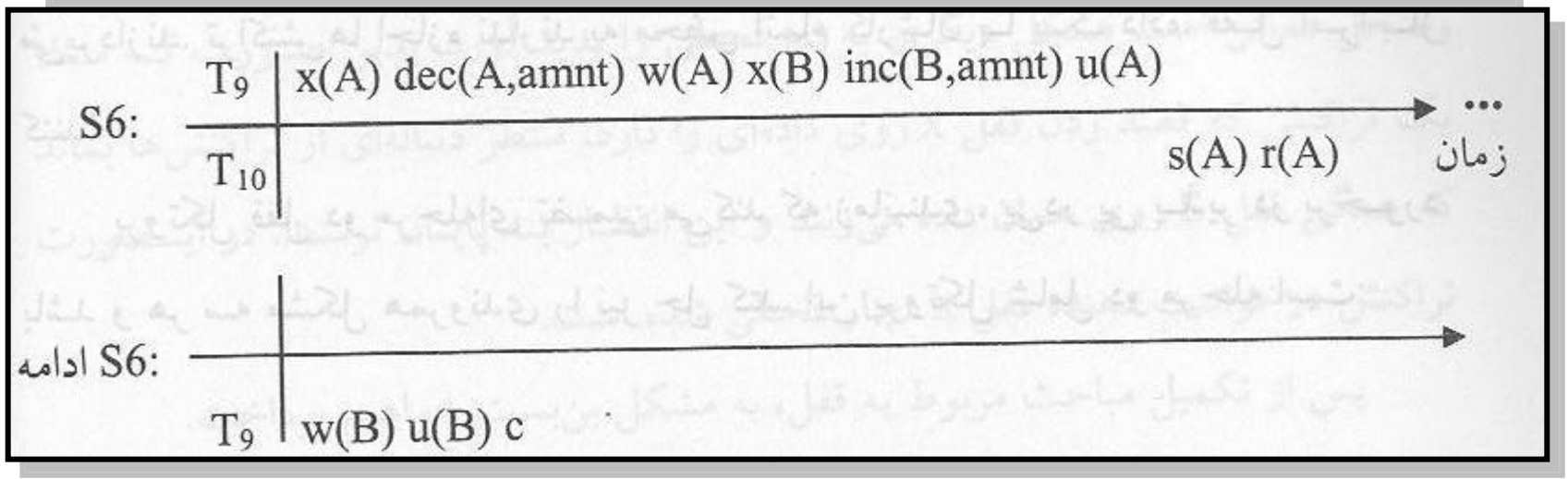
پروتکل قفل دو مرحله ای پایه (B2PL)

- در این پروتکل تراکنش ها شروع به گرفتن قفل های مورد نیاز خود می کنند و در صورت توفیق در این امر دستورات خود را اجراء می کنند.
- بمحض اینکه یکی از تراکنش ها قفلی را آزاد کرد، وارد مرحله دوم می شود که از این پس دیگر نمی تواند قفلی بگیرد.
- در این پروتکل ترتیبی برای باز کردن قفل ها وجود ندارد و به مشکل بن بست هم نمی پردازیم.
- اجراء این پروتکل تضمین می دهد که تراکنش ها روی نقطه قفل (نقطه ای که تراکنش آخرین قفلش را زده است) پی در پی پذیر شوند.

مثال

- زمانبندی معادل B2PL زیر چیست؟





توضیح مثال

- تراکنش T9 قفل داده A را باز نمی کند تا قفل B به او داده شود.
- بنابراین کار T10 تا رها شدن B به تعویق می افتد.
- با این روش دو تراکنش بصورت پی در پی پذیر کار کرده اند و نتیجه نیز صحیح است.
- در B2PL امکان وقوع بن بست است.
- برای رفع این مشکل پروتکل C2PL معرفی شده است.

پروتکل قفل دو مرحله ای محافظه کارانه (C2PL)

- در این پروتکل پیش از شروع اجراء اولین دستور تراکنش باید همه قفلهای مورد نیاز آن تراکنش گرفته شده باشد.

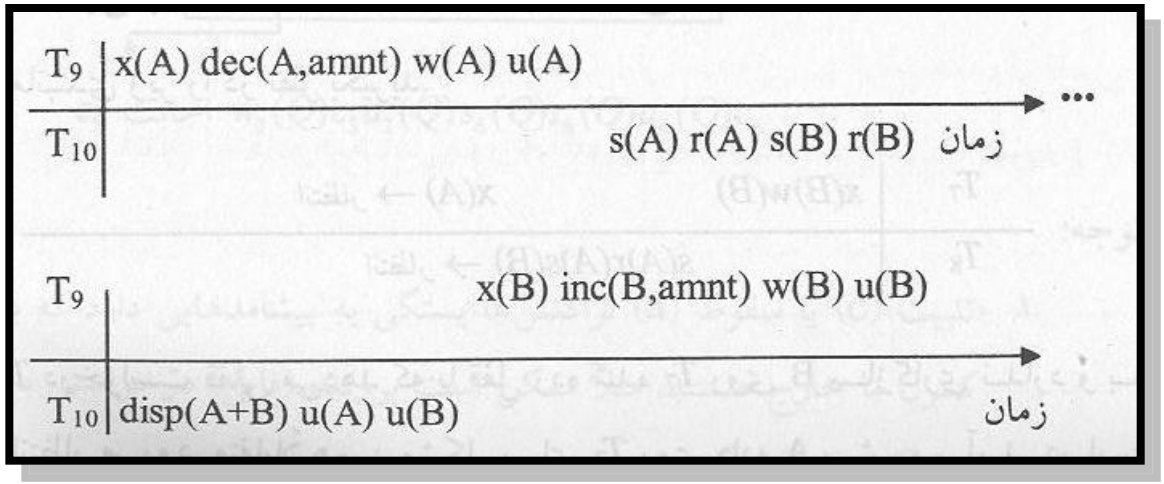
- یعنی اگر موفق به گرفتن حتی یک قفل نشد دوباره در صف قرار می گیرد و قفل های گرفته اش را باز می کند.

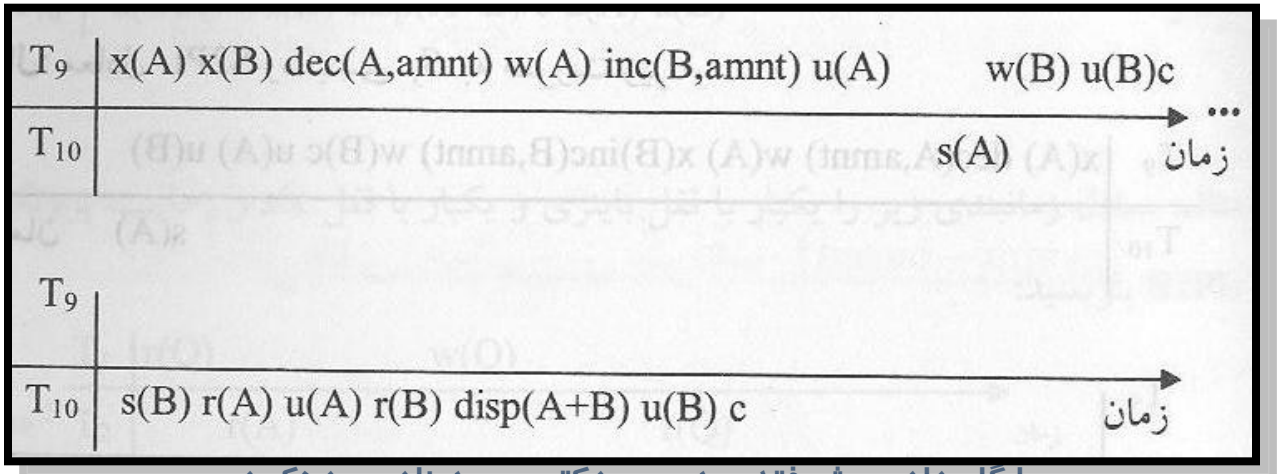
مزایا و معایب پروتکل قفل دو مرحله ای محافظه کارانه (C2PL)

- **مهمترین مزیت: بدون بن بست خواهد بود.**
  - زیرا حالتی پیش نمی آید که درخواست قفل داده ای که در اختیار دیگری است را داشته باشد و خودش هم برخی از قفل ها را در اختیار داشته باشد.
- **مشکلات اصلی در این پروتکل:**
  - کاهش سطح همروندی است
  - نیاز به دانستن مجموعه قفلهای مورد نیاز هر تراکنش پیش از اجراء آن است.
  - در دنیای واقعی احتمال بروز بن بست خیلی زیاد نیست و در اینجا به این بهانه کارایی سیستم بسیار افت می کند.

مثال

- زمانبندی معادل C2PL زیر چیست؟





پروتکل قفل دو مرحله ای محض (S2PL)

- در پروتکل B2PL علاوه بر بن بست امکان سقوط های آبشاری نیز وجود دارد.
- در پروتکل قفل دو مرحله ای محض باز کردن قفل های x تا بعد از اتمام تراکنش (a تا c) به تعویق می افتد.
- قفل های s می توانند کمی زودتر (بعد از آخرین دستور تراکنش و قبل از a و یا c) باز شوند.
- در سایر موارد عملکرد این پروتکل با B2PL یکسان است.

تذکر

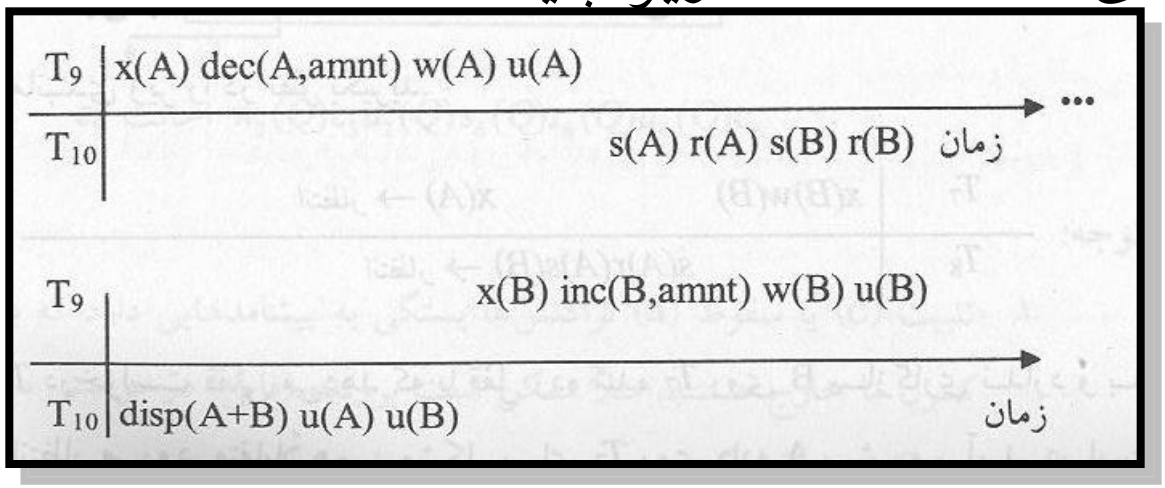
- در برخی پروتکل ها به نام SS2PL یا R2PL آزاد سازی هر دو نوع قفل S و X بعد از اتمام تراکنش خواهد بود.
- از آنجایی که ضرورتی برای نگهداشت قفل های S نیست، لذا عملکرد S2PL بهتر خواهد بود.

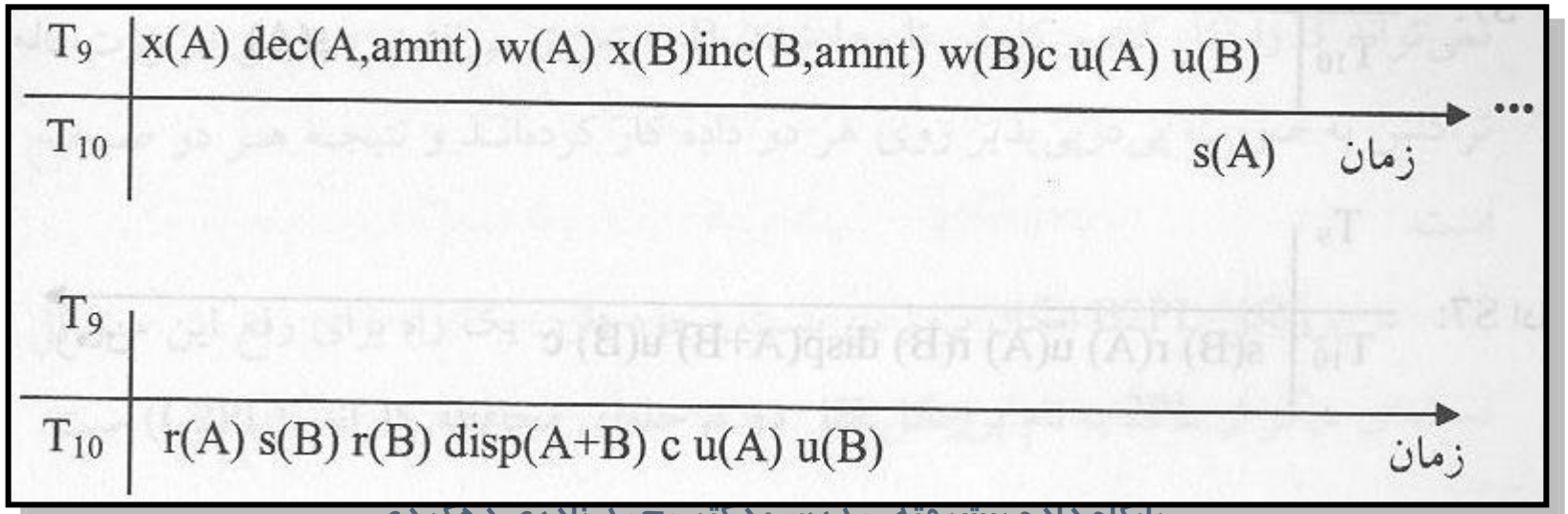
ویژگی های پروتکل S2PL

- سخت گیرانه عمل می کند و ممکن است که بسیاری از زمانبندی های درست را نپذیرد.
- با این وجود یکی از بهترین گزینه ها در اکثر سیستم های مدیریت بانک اطلاعات است.
- مزیت های این پروتکل که آنرا به پرکاربرد ترین و مناسب ترین گزینه بدل کرده است:
  - تضمین توأمان پی در پی پذیری و ترمیم پذیری
  - کاهش میزان پیامها در بانک اطلاعات نامتمرکز است. زیرا نیازی به بازکردن قفل ندارد.

مثال

- زمانبندی معادل S2PL زیر چیست؟





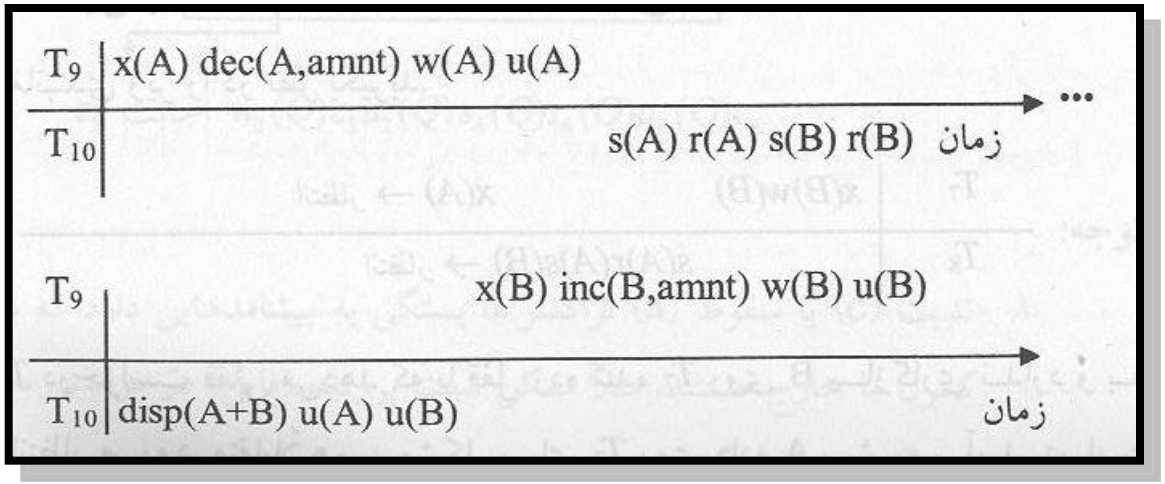
پروتکل SC2PL

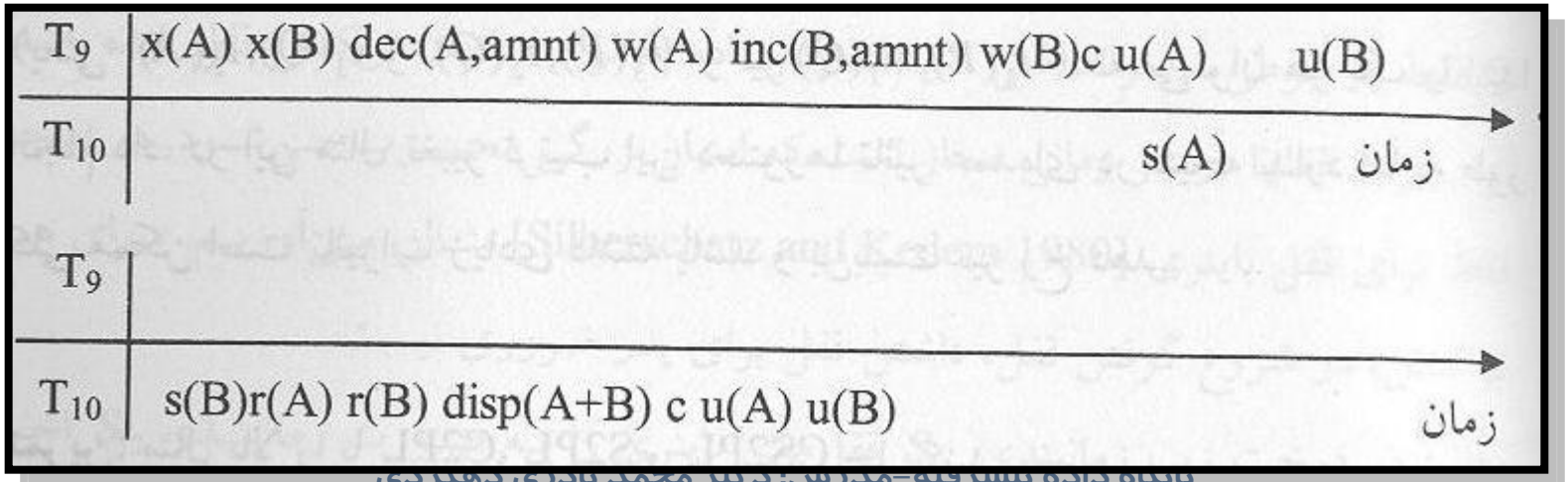
- این پروتکل مزایای هر دو پروتکل C2PL و S2PL را در کنار هم داراست.

- از قواعد C2PL برای قفل کردن داده ها و از قواعد S2PL برای آزاد سازی قفل ها استفاده می شود.

مثال

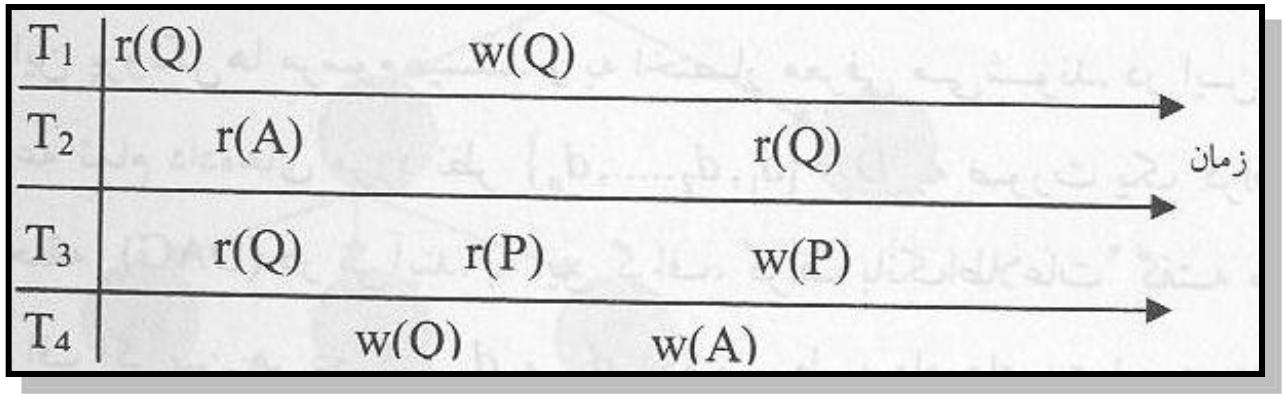
- زمانبندی معادل SC2PL زیر چیست؟

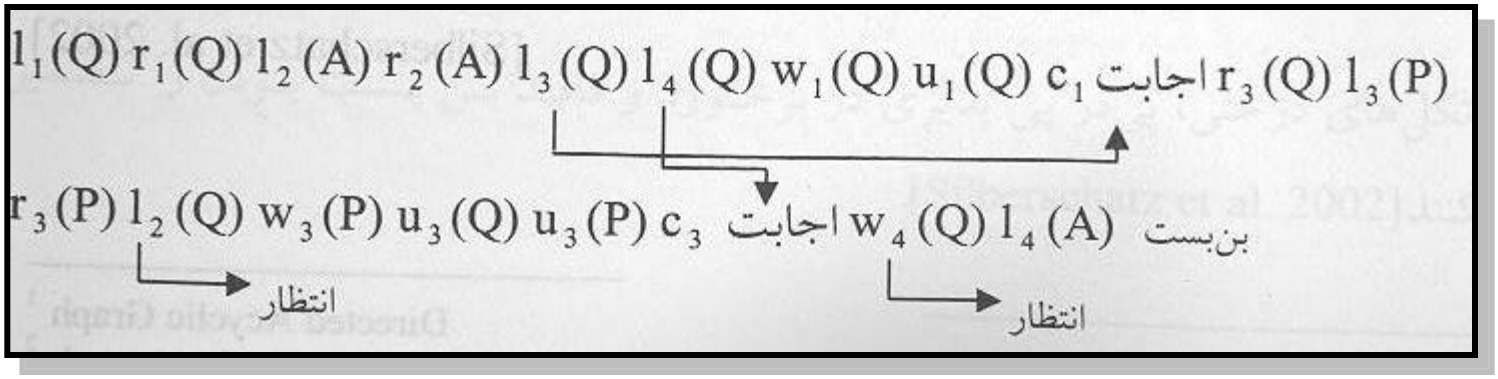




مثال ۱

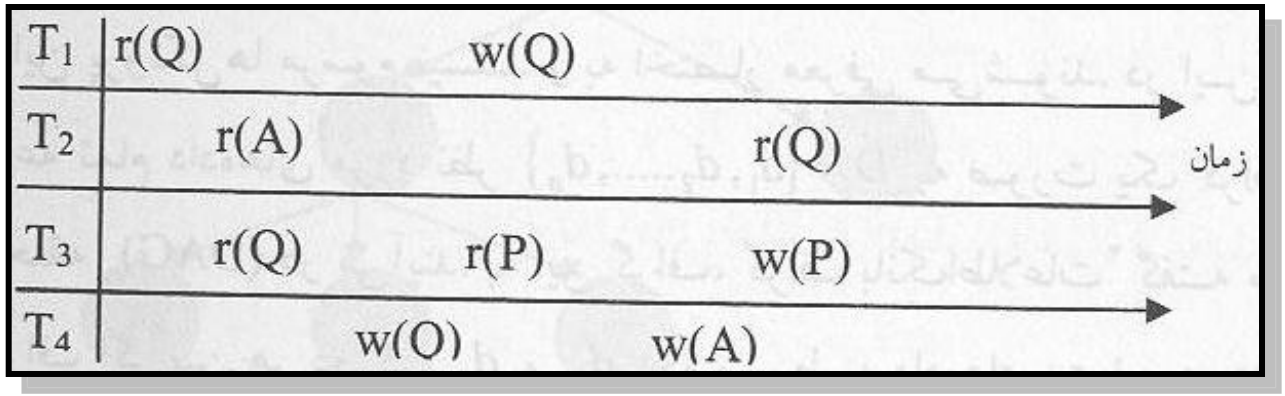
- معادل زمانبندی زیر را با قفل باینری بنویسید:





مثال ۲

- معادل زمانبندی زیر را با قفل s/x بنویسید:



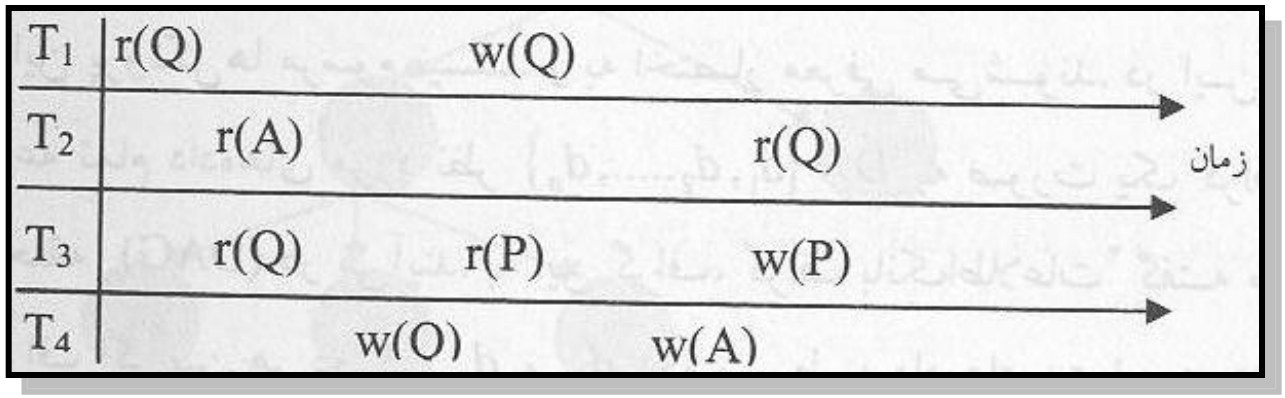
$s_1(Q)\ r_1(Q)\ s_2(A)\ r_2(A)\ s_3(Q)\ r_3(Q)\ x_4(Q)\ x_1(Q)\ s_3(P)\ r_3(P)\ s_2(Q)\ r_2(Q)$

$u_2(A)\ u_2(Q)\ c_2\ x_3(P)\ w_3(P)\ u_3(Q)\ u_3(P)\ c_3$ اجابت $w_1(Q)\ u_1(Q)\ c_1$ اجابت

$w_4(Q)\ x_4(A) w_4(A)\ u_4(A)\ u_4(Q)\ c_4$

تمرین

- زمانبندی زیر را با C2PL، S2PL و CS2PL حل کنید.



پروتکل های دیگر

- پروتکل های مبتنی بر گراف
- پروتکل های مبتنی بر مهر زمانی

پروتکل های مبتنی بر مهر زمانی

- در این پروتکل ها که بویژه در بانکهای اطلاعات نامتمرکز کاربرد دارد، هر تراکنش بمحض ورود یک مهر زمانی تصاعدی (مثل زمان ورود تراکنش به سیستم) تخصیص داده می شود.
- $TS(T_i)$ به معنای مهر زمانی تراکنش T_i است.
- برای دو تراکنش T_i و T_j که T_j دیرتر از T_i وارد سیستم شده باشد داریم: $TS(T_i) < TS(T_j)$

روشهای تولید مهر زمانی

- ساعت سیستم در سیستم های نامتمرکز
- زوج مرتب (مهر زمانی محلی سایت و شناسه سایت) در سیستم های نامتمرکز

پروتکل های مبتنی بر گراف

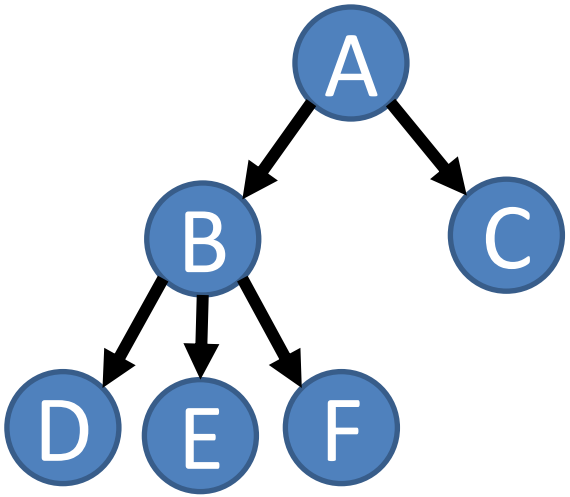
در این پروتکل ها مجموعه تمامی داده های مورد نظر $D=\{d_1,d_2,...,d_n\}$ به صورت یک گراف جهت دار بدون حلقه در می آید. به این گراف گراف بانک اطلاعاتی گفته می شود. در این گراف بین هر دو گره d_i و d_j که مربوط به داده های متمایز هستند، لبه ای به شکل $d_i \rightarrow d_j$ وجود داشته باشد، آنوقت هر تراکنش که می خواهد از داده d_j استفاده نماید ، باید قبل ازd_j داده d_i را نیز قفل کند (یعنی کل مسیر مورد نیاز را قفل نماید). در این پروتکل ها از قفل s/x استفاده می شود.

مثال با قفل باینری • پروتکل های مبتنی بر گراف

هر تراکنش اولین قفلش را روی هر داده ای می تواند بزند اما از آن پس داده ای مثل Q به شرطی می تواند قفل شود که والد داده Q توسط همان تراکنش قفل شده باشد. آزاد سازی قفل ها در هر زمانی مجاز است که امتیاز بزرگی محسوب می شود. (برای هر تراکنش در شروع داشتن قفل برای والد ضروری نیست).

مثال : در درخت زیر زمانبندی S_{11} قفل های باینری را به صورت زمانبندی S_{12} اخذ می نماید.

S_{11} : $w_1(E)$, $w_2(D)$, $w_1(C)$, $w_2(E)$, $w_2(F)$



S_{12} : $l_1(E)$, $w_1(E)$, $u_1(E)$, $l_2(D)$, $w_2(D)$, $u_2(D)$, $l_1(A)$, $l_1(C)$, $w_1(C)$, $u_1(C)$, $u_1(A)$, $l_2(A)$, $l_2(B)$, $l_2(E)$, $w_2(E)$, $u_2(E)$, $l_2(F)$, $w_2(F)$, $u_2(F)$, $u_2(B)$, $u_2(A)$

یکی از مشکلات این دسته پروتکل ها

- آن است که بسیاری از داده ها که شاید نیاز به دسترسی به آنها نیست هم باید قفل شوند.
- این امر علاوه بر افزایش سربار اضافی سطح همروندی را نیز کاهش می دهد. ضمناً مشکل سقوط آبشاری به قوت خود باقی است.

نکته : این پروتکل، یک پروتکل پرهیز از بن بست است و برای جلوگیری از سقوط آبشاری طراحی نشده است.

پروتکل های مبتنی بر مهر زمانی

- در این پروتکل ها که به ویژه در بانک های اطلاعات نامتمرکز به کار می روند، به هر تراکنش، به محض ورود، یک مهر زمانی تصاعدی (مثلاً زمان ورود تراکنش به سیستم) تخصیص داده می شود.
- فرض کنید مهر زمانی تراکنش Ti را با TS(Ti) نمایش دهیم برای دو تراکنش Ti و Tj در صورتی که Tj دیرتر وارد سیستم شده باشد، TS(Ti) < TS(Tj) می باشد. بر این اساس پروتکل های مبتنی بر مهر زمانی تراکنش ها را به ترتیب مهر زمانی آنها، به صورت پی در پی پذیر اجرا می کند.
- در سیستم های متمرکز از ساعت سیتم جهت تخصیص مهر زمانی و در سیستم های نامتمرکز می توان از یک سایت یا زوج مرتب مهر زمانی محلی و ID سایت استفاده کرد.

خواندن و نوشتن در پروتکل های مبتنی بر مهر زمانی

برای هر داده Q مهر زمانی خواندن و نوشتن به این صورت تعریف می شود:

- W-TS(Q) (مهر زمانی نوشتن داده Q) : برابر است با بزرگترین مهر زمانی تراکنشی (که به صورت موفقیت آمیز) روی Q نوشته است.
- R-TS(Q) (مهر زمانی خواندن داده Q) : برابر است با بزرگترین مهر زمانی تراکنشی (که به صورت موفقیت آمیز) Q را خوانده است.

نکته : منظور از بزرگترین، آخرین تراکنش و یا جدید ترین تراکنش می باشد.

اعمال قواعد خواندن و نوشتن پروتکل های مبتنی بر مهر زمانی تضمین می کند:

- که دستورات خواندن و نوشتن که با هم برخورد دارند به ترتیب مهر زمانی ایجاد شوند
- و زمانبندی های مربوطه پی در پی پذیر باشند.

۱. قواعد خواندن :

فرض کنید تراکنش **Ti** شامل یک دستور **read(Q)** است.

- اگر **TS(Ti) <= W-TS(Q)** باشد آنگاه تراکنش **Ti** داده ای را می خواند که مقدارش انگار بعداً نوشته می شود. پس در این صورت با دستور خواندن این تراکنش موافقت نمی شود و تراکنش **reject** می شود.

- اگر **TS(Ti) >=W-TS(Q)** باشد، آنگاه دستور خواندن تراکنش **Ti** اجرا می شود و مهر زمانی خواندن **Q**، با ماکزیمم مهر زمانی تراکنش **Ti** و مهر زمانی خواندن **Q** مقدار دهی می شود.

۲. قواعد نوشتن:

فرض کنید تراکنش Ti شامل یک دستور write(Q) باشد.

- اگر TS(Ti) < R-TS(Q) یا TS(Ti) < W-TS(Q) باشد آنگاه با دستور نوشتن تراکنش موافقت نمی شود و تراکنشTi، رد می شود.

- در غیر این صورت نوشتن اجرا می شود و مهر زمانی نوشتن Q، با TS(Ti) مقدار دهی می شود.

تذکر: رد شدن (reject) ، به معنای در انتظار ماندن یا سقوط و شروع مجدد می باشد.

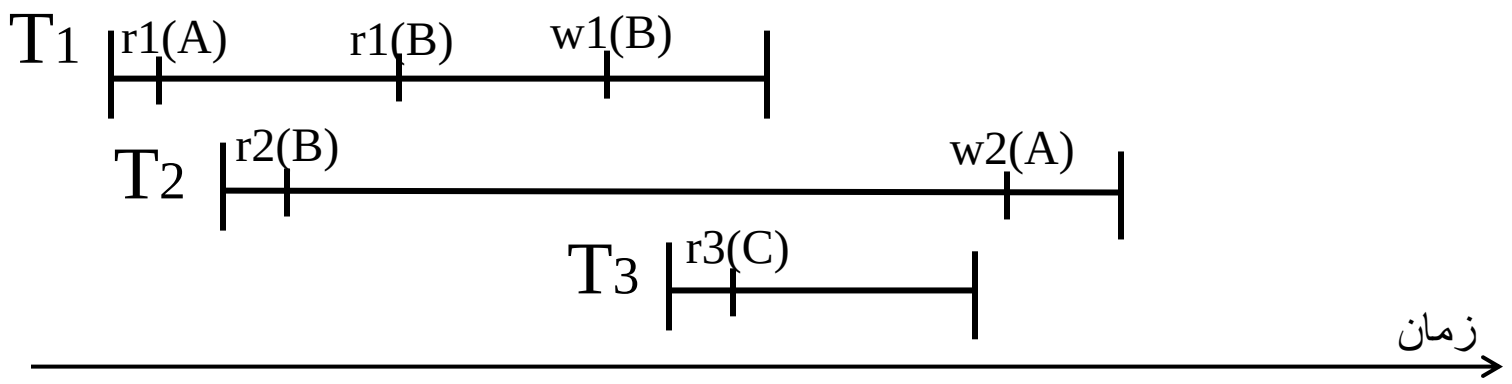
نکته: این پروتکل مبتنی بر گراف مطمئناً فاقد حلقه بوده و زمانبندی آن پی در پی پذیر است.

مثال : زمانبندی زیر را با پروتکل **S2PL** و مهر زمانی بنویسید :

S2PL: s1(A) , r1(A) , s2(B) , r2(B) , s1(B) , r1(B) , x1(B) , s3(C) , r3(C) , c3 , u3(C) , x2(A) بن بست

انتظار ← انتظار ↓

با پروتکل مهر زمانی می توان دید که **TS(T3) > TS(T2) > TS(T1)** اجرای زمانبندی در شکل زیر مشاهده می شود.



روش های مدیریت بن بست

بن بست چیست؟

سیستم زمانی دچار بن بست می شود که مجموعه ای از تراکنش ها برای همیشه منتظر یکدیگر باشند.

روش های مدیریت بن بست:

- چشم پوشی (ignore)
- فرصت (timeout)
- پیشگیری (prevention)
- اجتناب (avoidance)

چشم پوشی (ignore)

از آنجایی که در عمل احتمال برقراری همه شرایط وقوع بن بست پایین و هزینه مقابله یا رفع آن بالاست، یک راه حل برای مدیریت بن بست چشم پوشی از آن می باشد. که برخورد با آن به برنامه نویس، مدیر سیستم و ... واگذار می شود.

فرصت (timeout)

تراکنش که به حالت انتظار می رود، فقط برای مدت زمان معینی منتظر بماند و پس از آن در صورت سرویس نگرفتن ساقط شود. روشی ساده ولی منجر به قحطی می شود.

پیشگیری (prevention)

این روش تضمین می کند که سیستم هرگز دچار بن بست نشود.

- از پروتکل هایی استفاده نمود که در آن هر تراکنش قبل از شروع به اجرا تمامی قفل های مورد نیازش را بگیرد (C2PL,SC2PL)
- بین داده ها، ترتیبی در نظر گرفته شود و تراکنش ها فقط بر اساس این ترتیب مجاز به قفل کردن باشند. (پروتکل های مبتنی بر گراف)

اجتناب (avoidance)

تراکنش ها به پیش میروند و هنگام درخواست داده احتمال وقوع بن بست بررسی و از آن اجتناب می شود.

یکی از روش ها استفاده از مهر زمانی مخصوص بن بست است و یکی از روشهای زیر را می توان اعمال کرد:

- روش **wait-die** : تراکنش پیر تر(با مهر زمانی کمتر) منتظر تراکنش جوانتر می ماند تا قفل مربوطه را آزاد کند. در مقابل تراکنش جوانتر هرگز منتظر نمی ماند، بلکه ساقط می شود (می میرد). ممکن است یک تراکنش چندین بار بمیرد.

- روش **wound-wait** : تراکنش پیر تر به جای انتظار، تراکنش جوامتر را می کشد (قبضه ای یا **preemptive** نام دارد). تراکنش جوانتر منتظر تراکنش پیر تر می ماند. در این روش تراکنش پیر تر اولویت بالایی دارد و با کشتن تراکنش جوانتر، فرایند اجرا را به قبضه خود در می آورد.

نکته : روش wound-wait ممکن است کمتر منجر به ساقط شدن تراکنش ها شود زیرا از تعداد تراکنش های پیرتر کاسته می شود و از بروز قحطی ممانعت به عمل می آید.

مثال : فرض کنید Ti درخواست قفل روی داده Q دارد و Tj داده Q را قفل ناسازگار کرده است. در این صورت طبق جدول زیر عمل خواهد شد:

	TS(Ti) < TS(Tj)	TS(Ti) > TS(Tj)
wait-die	Ti منتظر می ماند	Ti با همان TS شروع مجدد می شود
wound-wait	Ti اقدام به کشتن Tj میکند (Tj با همان TS شروع مجدد می شود)	Ti منتظر می ماند

تشخیص و رفع بن بست

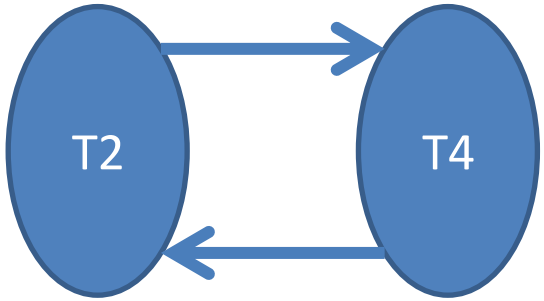
برای تشخیص بن بست گرافی به نام گراف انتظار رسم می نماییم.
گره های آن، تراکنش ها و یال های آن، انتظار آنها می باشد.

$T_i \rightarrow T_j$: تراکنش Ti منتظر تراکنش Tj میباشد.

چنانچه گراف انتظار فاقد حلقه باشد بن بست وجود ندارد.

در صورت وجود حلقه، جهت شکستن حلقه، می توان یکی از گره ها را که در این حلقه وجود دارد حذف کرد (انتخاب گره قربانی (victim) و ساقط کردن آن). معمولا گره ای به عنوان گره قربانی استفاده می شود که ساقط کردن آن، کمترین هزینه را داشته باشد.

مثال : گراف انتظار زمانبندی S10 با قفل باینری به این صورت می باشد:



می توان یکی از تراکنش ها را به عنوان قربانی ساقط نمود تا دیگری بتواند ادامه دهد.

مهمترین معیار های انتخاب تراکنش قربانی

- تعداد حلقه هایی از گراف انتظار که این تراکنش در آنها شرکت دارد و با ساقط شدن آن، این حلقه ها نیز شکسته خواهد شد.
- حجم کاری که تا کنون در آن تراکنش صورت گرفته است.
- تعداد به روز رسانی های انجام شده در آن.
- حجم کاری از تراکنش که برای اتمام آن باقی مانده است.

اگر همواره یک تراکنش قربانی شود قحطی پیش می آید. یک راهکار برای رفع این مشکل این است که تعداد دفعات ساقط شدن به عنوان فاکتوری در انتخاب قربانی در نظر گرفته شود. اگر با ساقط شدن یک قربانی، بن بست از بین نرود، قربانی بعدی را ساقط می کنیم.

سطوح جامعیت

آنچه تا به اینجا به عنوان شرط درستی زمانبندی ها، به نام پی در پی پذیری و ترمیم پذیری بررسی کردیم، بالا ترین سطح از اطمینان در مورد صحت و جامعیت بانک اطلاعات را برای ما به ارمغان می آورد.

به همین منظور سطوح پایین تری از جامعیت نیز وجود دارند. حتی در بعضی از نسخه های SQL می توان مشخص کرد که تراکنش مورد نظر با چه سطحی از جامعیت اجرا شود.

- **سطح پی در پی پذیر و ترمیم پذیر :** بالا ترین سطح جامعیت و آنچه تا به حال معرفی شد که پیش فرض ما نیز همین سطح بوده است.
- **سطح قابل تکرار (repeatable read) :** فقط داده های نهایی شده را می توانیم بخوانیم. در این سطح اگر یک داده را چندین بار هم بخوانیم، همیشه مقدار یکسانی برای آن دریافت خواهیم کرد.
- **سطح قابل تثبیت شده (read committed):** فقط داده های نهایی شده را می توانیم بخوانیم اما تضمینی نیست که با خواندن های بعدی هم هیمن مقدار را بخوانیم.
- **سطح قابل خواندن تثبیت نشده (read uncommitted):** اجازه خواندن داده های نهایی نشده را نیز می دهد.

نمونه دستورات در اوراکل 10g برای تعیین سطح جامعیت تراکنش:

```
Set transaction isolation level serializable;

Set transaction isolation level read committed;
```

که به ترتیب پی در پی پذیر و خواندن تثبیت شده تعیین می کند.

مدیریت ترمیم

- طبق خواص پایایی و یکپارچگی که برای تضمین جامعیت توسط تراکنش ها الزامی هستند، هر تراکنشی که commit می شود، باید از آن پس اثرات آن در بانک اطلاعاتی، حتی در صورت وقوع خرابی، دائمی و همیشگی باشد و برای تراکنشی که فقط برخی از دستورات آن اجرا شده اند، باید اثر دستورات اجرا شده آن خنثی باشد.
- تامین این دو ویژگی از جمله وظایف بخشی از مدیریت تراکنش ها به نام واحد مدیریت ترمیم می باشد.

انواع خرابی (failure)

۱. خرابی تراکنش (transaction failure): ممکن است منطق تراکنش دچار خطا شود و یا اینکه تراکنش به خودی خود درست اجرا شود اما با قرار گرفتن در شرایط سیستم، خطایی منجر به خرابی این تراکنش شود:

- خطای منطقی : تراکنش به دلیل شرایط داخلی خود (خواندن مقدار از نوع داده غلط، پیدا نکردن داده مورد نظر، تجاوز از محدوده روی رسانه و ...) نتواند کامل اجرا شود.
- خطای سیستمی : تراکنش به خودی خود درست کار می کند اما در سیستم شرایطی پیش می آید که این تراکنش را از کار می اندازد.(مثال : وقوع بن بست)

۲. خرابی سیستم (system crash): خرابی سخت افزاری یا نرم افزاری که سبب از کار افتادن سیستم (down شدن) میشود (مثل قطع برق ...)، رایج ترین نوع خرابی هستند. در این نوع خرابی ها اطلاعات حافظه اصلی سیستم از بین می رود ولی آسیبی به اطلاعات روی دیسک (حافظه جانبی – رسانه) وارد نمی شود.

۳. خرابی رسانه : خرابی که سبب شود اطلاعات روی رسانه از بین برود یا قابل بازیابی نباشد، مانند خرابی دیسک یا خرابی هد یا کنترل کننده دیسک.

٤. خرابی ارتباطات : این دسته از خرابی ها مختص بانک های نامتمرکز می باشند.

الگوریتم های ترمیم

- برای تضمین جامعیت بانک اطلاعات و اعمال خواص یکپارچگی و پایایی تراکنش ها در صورت وقوع خرابی، از الگوریتم هایی استفاده می کنیم که عموماً شامل دو مرحله زیر می باشند :
  - ۱. مرحله اول : در حین عملکرد عادی سیستم، اطلاعاتی که برای انجام ترمیم (پس از وقوع خرابی) به آنها نیاز داریم در جایی ثبت شوند.
  - ۲. مرحله دوم : پس از وقوع خرابی (وقتی سیستم دوباره بالا می آید)، با استفاده از اطلاعات ثبت شده در مرحله قبل، خواص یکپارچگی و پایایی تراکنش ها اعمال گردند.

- طبق خاصیت پایایی، اثرات تراکنش که انجام شده باید دائمی و همیشگی گردد.
- طبق خاصیت یکپارچگی، تراکنشی که نیمه کاره متوقف شده است نباید هیچ اثری روی بانک اطلاعاتی گذاشته شود.
- پس در مرحله دوم ار الگوریتم ترمیم، هر تراکنشی که انجام شده است را تکرار کرده (دوباره اجرا می کنیم تا اثراتش روی بانک اطلاعاتی دائمی شود) و هر تراکنشی که ساقط شده را خنثی می کنیم.

دسته بندی الگوریتم های ترمیم

- رویکرد کارنامه (log- based)
- رویکرد رونوشت (shadow paging)

رویکرد کارنامه

- در این رویکرد اطلاعات مورد نیاز برای انجام ترمیم را در حافظه پایدار ثبت می کنیم. برای هر یک از دستورات یک تراکنش، رکوردی به نام رکورد کارنامه با ساختار زیر نوشته می شود.
  - برای شروع تراکنش Ti ، رکورد <Ti,start>
  - برای دستور write(x) از تراکنش Ti ، در حالت کلی رکورد <Ti,x,V1,V2> که V1و V2 به ترتیب مقادیر قبل و بعد از انحام نوشتن x می باشد.
  - با اجرای آخرین دستور تراکنش Ti ، رکورد <Ti,Commit> در کارنامه ثبت می شود.

در رویکرد مبتنی بر کارنامه که شاید رایج ترین روش ترمیم است، انعکاس تغییرات روی بانک اطلاعات (روی رسانه) به یکی از دو روش زیر می تواند صورت پذیرد:

- انعکاس معوق تغییرات در بانک اطلاعات
- انعکاس فوری تغییرات در بانک اطلاعات

انعکاس معوق تغییرات در بانک اطلاعات

- تمامی رکورد های کارنامه مربوط به انجام تغییرات، ثبت می شود اما انعکاس این تغییرات روی رسانه (یعنی اجرای واقعی write ها) تا زمان پس از انجام جزیی (partial commit) اجرای آخرین تراکنش به تعویق می افتد.
- در رکورد های کارنامه مربوط به دستور نوشتن تراکنش Ti، لازم نیست مقدار قدیمی را ثبت کنیم.
- در صورت خرابی در سیستم، به کارنامه مراجعه کرده و تمامی تراکنش هایی که انجام شده اند (یعنی هم رکورد های <Ti,Start> و هم رکورد های <Ti,commit> برای آنها موجود است) را تکرار می کنیم.
- تکرار شدن یک تراکنش یعنی تمامی داده هایی که تراکنش روی آنها نوشته است، با مقدار جدید موجود در رکورد کارنامه دستور نوشتن، مقدار دهی می شوند.

نکته : چنانچه از روش انعکاس معوق تغییرات استفاده کنیم، در انجام ترمیم نیازی به خنثی (undo) کردن نداریم، زیرا تا تراکنشی تثبیت نشده باشد، نمی تواند محتوای رسانه را تغییر دهد و اگر هم نیمه کاره ساقط شود چون تاثیری روی بانک اطلاعاتی نگذاشته ، نیازی به خنثی کردن نیست.

انعکاس فوری تغییرات در بانک اطلاعات

- در این روش به محض اجرای دستور نوشتن تراکنش آن تغییرات فوراً در رسانه منعکس می شود.
- از آنجا که تراکنش های تثبیت نشده هم قادر به تغییر بانک اطلاعات هستند، ممکن است خرابی اتفاق بافتد و نیاز به خنثی کردن تراکنش های نیمه کاره داشته باشیم پس ساختار رکورد کارنامه دستور نوشتن به صورت عمومی (Ti,x,V1,V2) است یعنی هم مقدار قدیمی و هم مقدار جدید x را ثبت می نماییم.

سؤال

آیا برای یک دستور نوشتن، ترتیب اجرای نوشتن روی رسانه و ثبت رکورد دستور نوشتن روی کارنامه تاثیر و اهمیت دارد؟

پاسخ

- دو سناریو زیر را در نظر بگیرید:

۱. ابتدا دستور نوشتن روی رسانه را اجرا کنیم و سپس رکورد کارنامه را ثبت کنیم.
   - ممکن است بین اجرا این دو دستور خرابی اتفاق بافتد و چون رکورد کارنامه ثبت نشده است در مرحله دوم الگوریتم ترمیم نمی توانیم ترمیم را انجام دهیم.

۲. ابتدا رکورد مربوط به دستور نوشتن روی کارنامه ثبت و سپس دستور نوشتن روی رسانه اجرا شود.
   - در این صورت اگر بین این دو دستور هم خرابی رخ دهد، چون قبلا رکورد نوشتن را ثبت کرده ایم می توانیم عمل ترمیم را انجام دهیم. این قاعده پروتکل **WAL (write ahead log)** نام دارد.

در مرحله دوم از الگوریتم ترمیم که تغییرات را فوراً روی دیسک منعکس می کند:

- تمامی تراکنش هایی مثل Ti که انجام شده اند (هم رکورد های <Ti,Start> و هم رکورد های <Ti,commit> برای آنها وجود دارد) را تکرار کرده و تمامی تراکنش هایی که شروع شده اند ولی اجرا نشده اند را خنثی می نماییم.

تکرار یعنی قرار دادن مقدار جدید و خنثی یعنی قرار دادن مقدار قدیمی در متغیر مربوطه.

چند نکته مهم

۱. ابتدا خنثی کردن و سپس تکرار ها را انجام می دهیم.
۲. برای تکرار از آغاز کارنامه شروع کرده و به ترتیب تراکنش های مربوطه را تکرار می کنیم تا به پایان برسیم.
۳. اما برای خنثی کردن از انتهای کارنامه به سمت شروع آن پیش می رویم و تراکنش های مورد نظر را خنثی می کنیم.
۴. در صورت وجود خرابی در حین عملکرد عادی سیستم و یا ترمیم، خنثی کردن و تکرار باید به گونه ای باشد که اثر چندین بار اجرا شدن آنها معادل اثر یکبار اجرای آن باشد. (خاصیت همانی)

معایب رویکرد کارنامه

- بزرگ شدن اندازه فایل کارنامه
- زمانگیر بودن جستجو در کارنامه
- احتمال تکرار مجدد تراکنش هایی که قبلا آنها را تکرار کرده ایم
- افزایش هزینه به روز رسانی بانک اطلاعات به دلیل کار با رسانه

برای رفع این معایب از نقاط بازرسی یا checkpoint استفاده می نماییم.

نقاط بازرسی

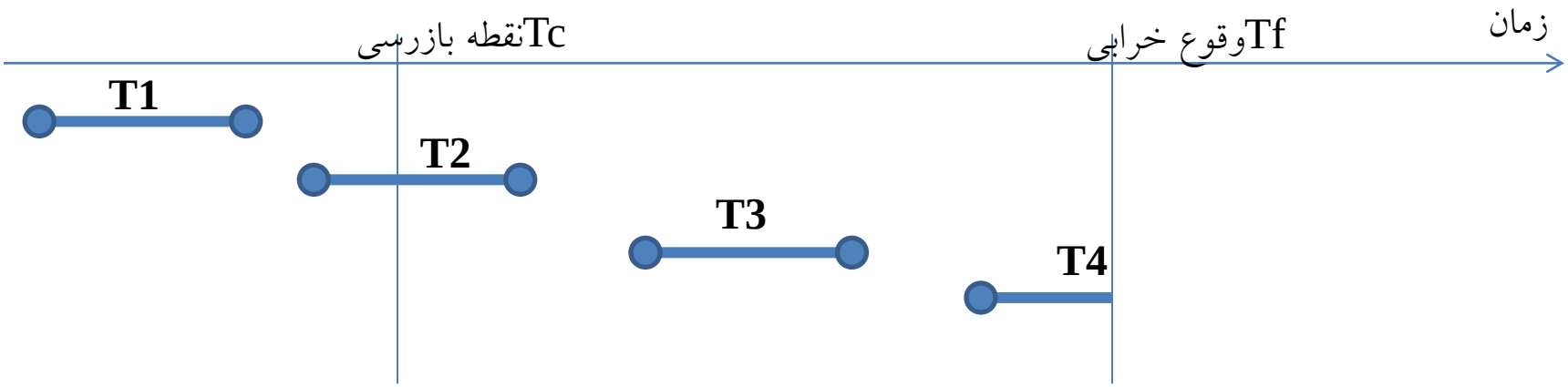
- در این روش به طور متناوب در برهه های زمانی معینی از عملکرد عادی سیستم، کارهای انجام شده روی بانک اطلاعات (تا آن زمان) را قطعی و نهایی می کنیم که به آن نقاط بازرسی گوییم.

عملیات لازم در نقاط بازرسی

- رکورد های کارنامه (که در حین عملکرد عادی سیستم در بافر نوشته می شدند) به حافظه پایدار منتقل می گردد.
- داده های تغییر یافته در بافر به دیسک منتقل می گردند.
- رکوردی به نام رکورد بازرسی با ساختار<checkpoint> ثبت می شود.
- پس حالا عملیات ترمیم فقط مربوط به بخش هایی از کارنامه می باشد که بعد از آخرین رکورد بازرسی هستند.
- برای مدیریت فایل کارنامه می توان از برش های کارنامه که کارنامه را به دو قسمت فعال و بایگانی تقسیم می کند، استفاده کنیم.

مثال

در این شکل پس از وقوع خرابی، به چه صورت عمل می شود؟



- تراکنش **T1** در زمان **Tc** نهایی شده و نیازی به ترمیم ندارد.
- تراکنش **T2**و **T3** را بنا به خاصیت پایایی باید تکرار نمود تا اثر آنها در سیستم نهایی و دائمی شود. (برای **T2** فقط بخشی از دستورات که پس از **Tc** اجرا شده اند را تکرار می کنیم.)
- تراکنش **T4** که نیمه کاره مانده است را بنا به خاصیت یکپارچگی، خنثی می کنیم.

رویکرد رو نوشت

- قبل از انجام تغییرات (قبل از شروع به اجرای تراکنش) یک نسخه کپی از صفحات مورد نیاز بانک اطلاعات تهیه (نسخه سایه) و در رسانه غیر فرار ذخیره می نماییم.

- نسخه جاری، نسخه اصلی است که تغییرات مورد نظر روی آن انجام می شود.
  - چنانچه تراکنش به انجام رسید، این نسخه را به عنوان بانک اطلاعات جدید تلقی کرده و نسخه سایه را از بین می بریم.
  - اما اگر تراکنش نتوانست انجام شود، نسخه جاری را از بین برده و بانک اطلاعاتی معادل همان نسخه سایه خواهد بود.

مزایا و معایب رویکرد رونوشت

- مزایا
  - سربار مربوط به نوشتن رکورد کارنامه را ندارد
  - ساده بودن انجام عملیات ترمیم
- معایب
  - برای پیاده سازی نیاز به کارنامه داریم
  - کپی کردن اطلاعات (حتی با وجود بهینه سازی های انجام شده) پر هزینه است
  - توسعه در حالت اجرا همروند مشکل است (بر خلاف کارنامه)
  - سربار انجام عملیات تثبیت زیاد است

در سیستم های اطلاعاتی معمولا از رویکرد کارنامه برای انجام ترمیم استفاده می شود.