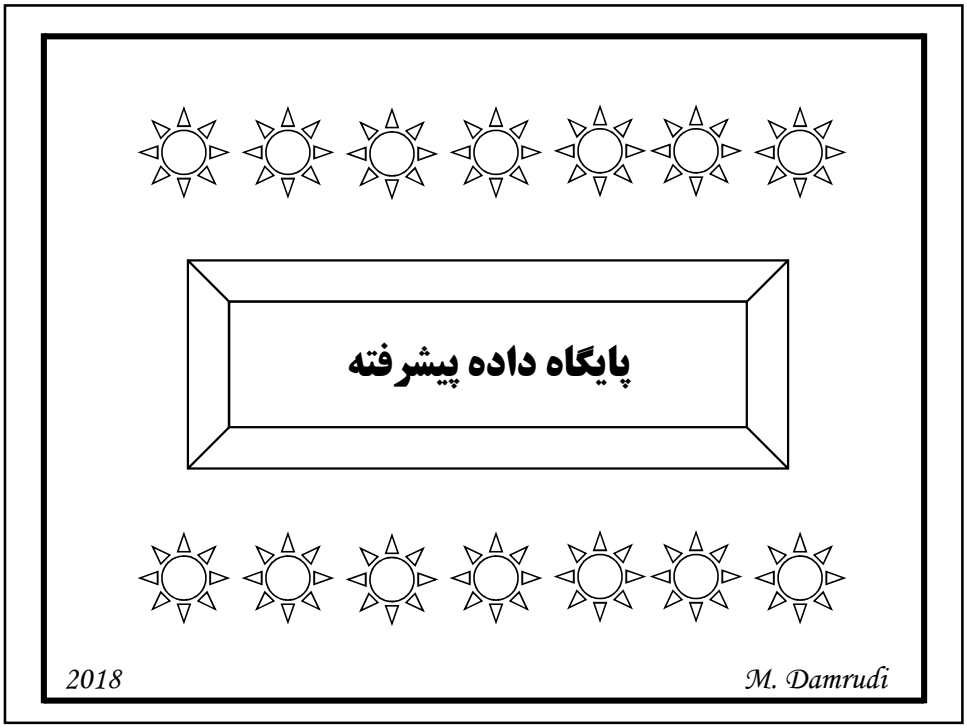


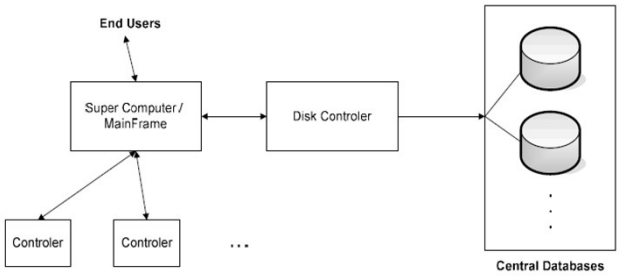
عوامل مؤثر در معماری پایگاه داده

- 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)

متمرکز 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.

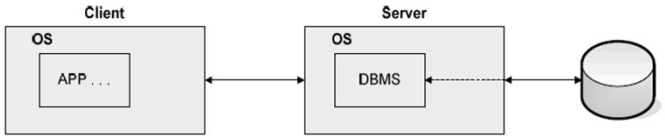
انواع معماری نامتمرکز

- معماری سیستم پایگاهی مشتری-خدمتگزار(سرویس دهنده- سرویس گیرنده)
- معماری سیستم پایگاهی توزیع شده (مبادله پیام)
- معماری با پردازش موازی (سخت افزار مشترک)
- 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

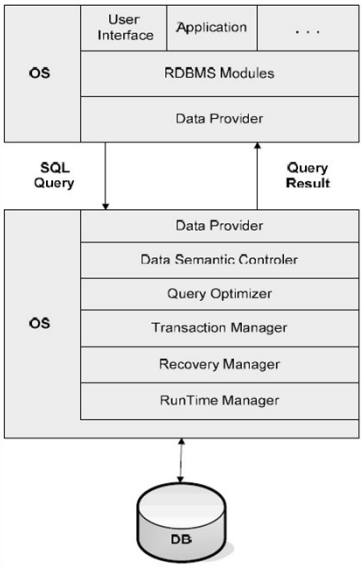


- کاربر با وجود واسط هایی نظیر زبان داده فرعی، واسط گرافیکی و واسط فرمی عمل می کند.

مزایای معماری مشتری–خدمتگزار

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

اجزاء معماری پایگاهی مشتری–خدمتگزار

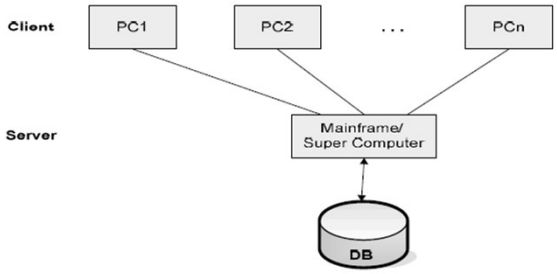


- اجزاء با در نظر گرفتن RDBMS
- ابزارهایی مثل ODBC و JDBC برای تسهیل تماس مشتری و خدمتگزار
- امکان فراخوانی راه دورRPC

طرح های معماری مشتری–خدمتگزار

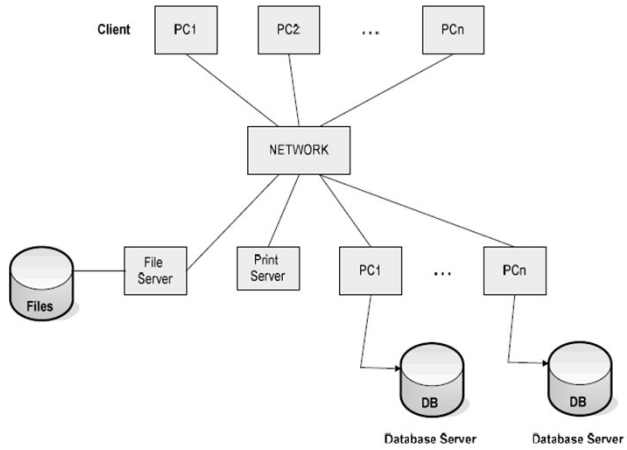
- از نظر تعداد مشتری
  - یک مشتری - یک خدمتگزار
  - یک مشتری - چند خدمتگزار
  - چند مشتری - یک خدمتگزار
  - چند مشتری - چند خدمتگزار
- از نظر پیکربندی کامپیوتری
  - معماری حول کامپیوترهای بزرگ
  - معماری حول شبکه

معماری حول کامپیوترهای بزرگ



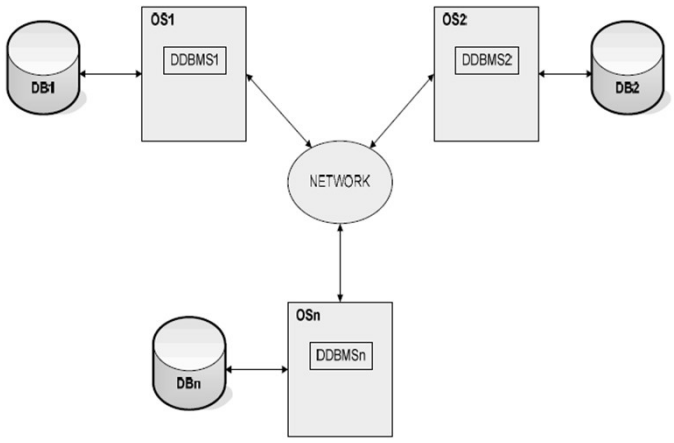
- ماشین خدمتگزار یک کامپیوتر بزرگ است و پایگاه داده روی همین ماشین ایجاد و مدیریت می شود.
- تعدادی کامپیوتر شخصی از خدمات پایگاهی این کامپیوتر بزرگ استفاده می کنند.

معماری حول شبکه



❖ تعدادی از کامپیوترها بعنوان مشتری و تعدادی خدمتگزار هستند و با شبکه به یکدیگر مرتبط هستند.

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 به صورت 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 به صورت 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.

معماری چهار لایه ای بانک اطلاعات

- رایج ترین مدل داده ای برای تصویر اداراکی عام: EER، ER، UML
- برای پیاده سازی بانک اطلاعات در سطح تصویر ادراکی خاص: رابطه ای، شی گرا، XML

تصویر خارجی	کاربر ۱ ... کاربر ۲	دیدهای کاربران مختلف (views)
تصویر ادراکی عام	Entity1 — Entity2	کل بانک بدون توجه به مدلی خاص
تصویر ادراکی خاص	Table | A | B | C	کل بانک در قالب مدل انتخابی
تصویر فیزیکی	...	کل بانک روی رسانه ها

مهمترین مزایای مدل رابطه ای

- سادگی و قابل فهم بودن برای عموم
- پشتوانه تئوریک قدرتمند
  - جبر رابطه ای
  - حساب رابطه ای تاپلی
  - حساب 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

- OMG یک گروه سازمان یافته در رابطه با بانک های اطلاعات مبتنی بر شی ء است.
- 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.
- OMG یک کارگروه ویژه به نام ODBTWG تشکیل داده است برای:

(Object Database Technology Work Group)

- توسعه استانداردهای مختلف برای پایگاه داده شیء گرا و تشویق تولیدکنندگان این سیستم به رعایت استانداردهای مذکور
- توسعه فناوریهای پایگاه داده شیء گرا مثل(Replication)
- توسعه مدیریت پایگاه داده مثل (Spatial Indexing)
- توسعه فرمت های مختلف نگهداری اطلاعات مثل (XML)
- فعالیت در زمینه سیستم های بلادرنگ
- فعالیت در زمینه نرم افزارهای متن باز مثلdb4o

مفاهیم مدل داده شی گرا

- کلاس: قالب اطلاعاتی که ویژگی های مورد نظر برای مجموعه همسان از اشیاء دنیای واقعی را بیان می کند. (کلاس دانشجو)
- شیء: مجموعه مقادیر داده هایی که یک پدیده دنیای واقعی را معرفی می کنند و نمونه ای از کلاس ویژه ای می باشند. (هر یک از دانشجوها)

مدل داده ای شیء گرا	مدل داده ای رابطه ای
کلاس Class	موجودیت Entity
شیء Object	نمونه Instance

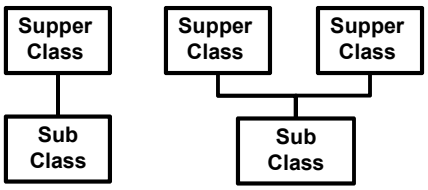
- هر کلاس شامل مجموعه متغیرها که قالب داده های مربوط به کلاس را نگهداری می کنند و مجموعه ای از متدها که هر یک کدی است که برای پیاده سازی یک رفتار نوشته شده است.
- پیام به معنی فراخوانی یک رویه است. در هر متد به متغیرهای همان شیء به صورت مستقیم و به داده‌های اشیا دیگر با ارسال پیام می توان دسترسی داشت.

شناسه شیء O-ID

- برای 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 یا یک مقدار منحصر به فرد در عالم خارج مثل شماره ملی فرد.

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 های دیگر است.



- سلسله مراتبی از کلاس ها ایجاد می شود که قدرت و انعطاف پذیری زیادی به برنامه می دهد. سلسله مراتب کلاس ها به صورت درخت است.
- در این درخت کلاس ها به صورت supper class و sub class در رده های مختلف در کلاسها وجود دارد.
- گاهی ممکن است یک کلاس از بیش از یک کلاس ارث ببرد که multiple inheritance یا ارث بری چندگانه می نامند. مشکلات برخورد نام و ارث بری تکراری را به همراه دارد.

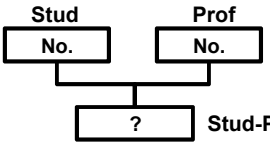
برخورد نام ها (Name Clash)

- در سلسله مراتب کلاسها ممکن است یک نام با معانی مختلفی آمده باشد.
- کلاس مشتق ازدوکلاس که یک نام بامعانی مختلف داردچه حالتی پیدامی کند:

۱- نامهای تکراری را معادل فرض کند.

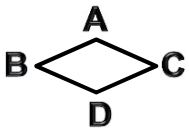
۲- ارث بری را با ترتیب خاصی انجام دهد. (از آخرین کلاس ارث ببرد)

۳- با استفاده از نام کلاس بالایی نامها را از حالت تکراری در آورد. Stud.No, Prof.No



ارث بری تکراری (Repeated Inheritance)

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



بسته بندی (Encapsulation)

- کنار هم قرار دادن داده ها و رفتارها در یک بسته (کپسول).
- باعث "استقلال پدیده ها" می گردد یعنی هر چه به یک پدیده مربوط است همراه اوست. باعث "پنهان سازی اطلاعات" می گردد.
- کلاسها و اشیاء دو چهره دارند:
  - چهره درونی (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 (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.

بسته بندی (Encapsulation)

- قانون مهم در encapsulation: "چهره رابط باید کمینه باشد":
- پنهان سازی اطلاعات تا حد ممکن، راه حل اساسی برای حل مشکل نگهداری سیستم و عدم سرایت تغییرات به بخشهای دیگر.
- هر سیستم پس از تولید با تغییر شرایط باید مرتبا به روز در آید. تغییر بخشی باعث سرایت مشکلات به بخشهای دیگر می شود.
- در دنیای شی گرایی این سرایت مشکلات به حداقل می رسد، زیرا اشیا با چهره درونی هم کاری ندارند.
- چهره رابط باید با حالتی پایا و با آینده نگری لازمه طراحی شود تا به سادگی نیاز به تغییر نداشته باشد.

چند ریختی (Polymorphism)

- در زبانهای برنامه نویسی استفاده از اسامی تکراری به صورت 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(overloading).
- باعث کوتاه شدن و ساده شدن برنامه ها می شود.

پیام رسانی (Message Passing)

- به معنای فراخوانی رفتارهای اشیاء است. در ارسال پیام به اشیا، نام شیء، نام رفتار و پارامترهای رفتار (ورودی، خروجی) را باید تعیین کرد. رفتارها مانند توابع پیاده سازی می شوند. my_cat.run(100)
- اشیاء با دریافت پیام عکس العمل مناسب نشان می دهند.
- شرایط پیام:
  - پیش شرط‌ها (Pre-Conditions): شرایطی که پارامتر ورودی باید داشته باشد و در صورت عدم برآورده‌شدن آن شیء اعلام خطا خواهد کرد و پاسخ مورد نظر را نخواهد داد.
  - پس شرط ها (Post-Conditions): شرایطی که پارامترهای خروجی باید داشته باشند.
  - اقداماتی که شیء در صورت بروز خطا یا اتفاق های گوناگون انجام خواهد داد. مثلاً اگر شیئ 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 گوناگون استفاده می نماید.

انواع پیام

- پیام هماهنگ (Synchronous): فرستنده و گیرنده درگیر آن هستند تا کار تمام شود. فرستنده منتظر پاسخ می ماند و کار دیگری انجام نمی دهد. (تلفن)
- پیام ناهماهنگ (Asynchronous): فرستنده بدون توجه به وضعیت گیرنده پیام را ارسال می کند و خودش بلافاصله کار دیگری انجام می دهد. ممکن است از نتیجه به صورت غیر مستقیم استفاده کند یا نکند. (نامه)
- پیام آینده (Future): فرستنده پیام را ارسال می کند و کار دیگری انجام می دهد. در زمان مقرر یا درفواصل زمانی مشخص، دریافت پاسخ را بررسی کرده یا منتظر پاسخ می ماند.
- پیام فرصتی (Timeout): فرستنده مدت مشخصی منتظر گیرنده می ماند که یا آماده دریافت پیام شود یا پاسخ دهد. پس از اتمام مدت، یا ارتباط قطع می شود و یا پیام تکرار می شود.
- پیام فوری (Balking): فرصت پیام نزدیک به صفر است. یعنی اگر گیرنده بلافاصله آماده دریافت یا پاسخ گویی 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.
- این کلاس ها در مرحله اجرا مشخصاتشان کامل می شود.
- باعث افزایش انعطاف پذیری در پیاده سازی اشیاء می شود.
- باعث کاهش سرعت اجراء برنامه ها نیز می گردد.

پیمانه (Module)

- آخرین و پیشرفته ترین مرحله برنامه سازی قبل از اوج گرفتن شی گرایی modular programming است. زیربرنامه های مربوط به یک پدیده در درون واحد بزرگتری به نام پیمانه قرار میگیرد.
- هرسیستم نرم‌افزاری مجموعه‌ای از پیمانه ها است. پیمانه ها ازاجزاهم بی‌خبرند.
- پیمانه ها دو چهره دارند. چهره internal که به خود پیمانه مربوط می شود. چهره interface که به پیمانه های دیگر مربوط می شود.
- تغییر ساختار درونی پیمانه ها تاثیری روی قسمتهای دیگر سیستم نمی گذارد.

انواع کلاس

- کلاس مفهوم زیربنا در شی گرائیست.
- کلاس پایه (base Class): توسط زبان برنامه نویسی مورد نظر قبلاً تعریف شده اند. زبانهای برنامه سازی شی گرا یک پیکره اصلی دارند و تعداد زیادی کلاس که کاربران می توانند در برنامه های خود به ارث ببرند.
- کلاس مجرد (abstract Class): هدف از تعریف این کلاسها، استفاده از آنها در ساخت کلاسهای دیگر است و نه تولید شیء. در درخت وراثت، همه کلاس‌ها برای ایجاد شی نیستند. فقط برگ ها اشیا را می سازند و سایر کلاس‌ها برای تعریف کلاس های دیگر مورد استفاده قرار می گیرند. نام دیگر این دسته از کلاسها بدون نمونه (non-instance) است.
- فوق کلاس (meta Class): به کلاسهایی گفته می شود که کلاسهای دیگر ایجاد می کنند، همچنانکه کلاسهای معمولی اشیاء را ایجاد می کنند.

معیارهای سنجش کیفیت کلاس

- همبستگی Coupling: کلاس ها نباید به صورت نامعقول به هم وابسته باشند. زیرا تغییر یک کلاس باعث تغییر کلاسهای وابسته به آن می شود. وابستگی در ارث بری، تعریف پیمانه و ... ایجاد می شود.
- پیوستگی Cohesion: داده ها و رفتارهایی که در یک کلاس تعریف می شوند باید پیوستگی ذاتی و طبیعی داشته باشند.
- کامل بودن Completeness: چهره رابط (interface) کلاس باید تمام خصوصیات آن را در رابطه با کاربرد مورد نظر در بر بگیرد. طراح کلاس باید آینده نگری داشته باشد تا از افزودن رابط های اضافی در آینده بی نیاز باشد.
- کفایت Sufficiency و بی پیرایگی Primitiveness: جامع و مانع بودن. طراح کلاس نباید هر آنچه در ارتباط با موضوع به ذهنش می رسد اضافه کند. باید کاربرد را در نظر گرفت و از آوردن مشخصات اضافی خودداری نمود.

داده مانا (Persistent Data)

- اهمیت داده در مقابل برنامه از ویژگیهای بانک اطلاعات است. در بانک اطلاعات داده ها اصالت دارند و برنامه ها در حال تغییر. داده ای که طول عمرش بیشتر از برنامه یا برنامه های مربوطه است داده مانا گویند. داده ها باید مورد حفاظت قرار بگیرند.
- مانایی جنبه های مختلفی دارد.
- مانایی زمانی (در طول زمان اشیا و کلاسها مانا باشند)
  - مانایی اشیاء: داده ها به صورت شیء نگهداری می شوند و به روز در می آیند.
  - مانایی کلاس ها: در طول زمان کلاس ها هم تغییر می کنند. کلاس هم نسخه های مختلفی دارد. کلاس ها هم باید مانا باشند. نسخه های مختلف آن ها تا زمانی که حتی یک شیء از آن نسخه در بانک وجود دارد باید نگهداری شود.
- مانایی مکانی
  - در بانک اطلاعات نامتمرکز مانایی مکانی وجود دارد. ممکن است نسخه های یک کلاس در جاهای مختلفی نگهداری شوند یا یک شیء از محلی به محل دیگر منتقل شود(object migration).

طراحی بانک اطلاعات شیء گرا

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

مهمترین انواع انتزاع پیشنهادی درپژوهشهای مختلف

- کلاس بندی (Classification):
- به مفهوم کلاس بر می گردد و رابطه تنگاتنگی با تعریف نوع داده دارد.
- با عنوان instance-of شناخته شده است.
- کلاس، کارگاه تعریف داده ها و رفتارهای تعدادی پدیده که خصوصیت مشترکی دارند است.
- هر پدیده باید شناسایی واحد داشته باشد که در مدت طول عمر آن پدیده تغییر نکند.
- هیچ شیئی نمی تواند وجود داشته باشد مگر اینکه از کلاس مشخصی باشد.

مهمترین انواع انتزاع پیشنهادی درپژوهشهای مختلف

- گروه بندی (Aggregation و یا Grouping):
- به معنای تشکیل مجموعه ای از اشیاء است.
- نوع خاصی از این انتزاع به نام شیء مرکب معروف است که از مجموعه چند شیء یک شیء بزرگتر به وجود می آید.
- 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 نوع خاص آن (شیء مرکب) به نام is-part-of شناخته شده است.
- مثال: "شیء آدرس"

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

مهمترین انواع انتزاع پیشنهادی درپژوهشهای مختلف

- ارث بری (generalization/specialization و sub type/super type)
- همان مفهوم ارث بری است.
- با عنوان is-a شناخته شده.

- ارتباط (association)
- نشان دهنده ارتباط بین اشیاء مختلف است.
- برخلاف نظر برخی از افراد، این ارتباط خود می تواند یک شیء باشد. مانند مدل رابطه ای که ارتباط بین جداول را با جداول دیگر نشان می دهد.
- این رابطه با عنوان member-of شناخته شده است.

روشهای گوناگون طراحی بانک اطلاعات شیء گرا

- استفاده از ساختارهای یک زبان های شیء گرای موجود (Java , C++) که قرار است برای پیاده سازی بانک اطلاعات استفاده شود. در این روش طراحی و پیاده سازی مراحل یک کار هستند. پیاده سازی به معنای ریختن مستقیم داده ها در قالب هایی که در مرحله طراحی تهیه شده اند و کارکردن با داده ها توسط متدهایی که از قبل آماده شده اند.
- استفاده از مکانیزم های انتزاعی ویژه که برای طراحی بانک اطلاعات شیءگرا پدید آمده اند. این روش با مدل EER و هدایت آن به سمت پشتیبانی از مدل شیء گرا آغاز گردیده است و سپس مدلهایی مانند ODL و UML برای طراحی بانک شی گرا و شئ - رابطه ای مورد توجه بسیار قرار گرفته اند.

ODL

- ODL (Object Definition Language) زبانی بیانی برای مدلسازی و طراحی بانک اطلاعات شی گرا است که توسط گروه ODMG به جهان فناوری اطلاعات معرفی شد.
- ODMG به استاندارد کردن ابزارها و روش ها برای طراحی می پردازد.
- مهمترین استانداردها (Interface Definition Language) IDL، ODL، (Object Query Language) OQL برای طراحی و پیاده سازی بانک اطلاعات شی گرا است.
- ODL مورد پذیرش عام است.
- با C++، Java، small talk قابل پیاده سازی است. امکان پرس و جو در ODL وجود ندارد زیرا زبان پیاده سازی نیست و تنها برای طراحی شمای بانک اطلاعات شی گرا به کار می رود.
- از تمام انتزاع های بانک اطلاعات مثل گروه بندی و ارتباط استفاده می کنند.

ODL

- تعریف کلاس در ODL حداقل موارد زیر را شامل می شود:
  - کلمه interface که مشخص می کند آنچه تعریف می‌شود فقط چهره رابط است.
  - نام رابط تعریف شده.
  - اجزاء کلاس مانند صفت ها (attributes)، ارتباط ها (relationships) و روال ها یا متدها (methods) که درون {} تعریف می شوند.
- 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;
        };
```


پیاده سازی بانک اطلاعات شی گرا

- از شی گرایی به عنوان ابزاری در طراحی استفاده کرده و برای کد کردن مدلهای دیگر (مانند رابطه ای) به کار گرفته می شود.
- مفاهیم شی گرایی در زبان برنامه نویسی که برای کار با بانک اطلاعات به کار می رود جای داده شود. یکی از دو انتخاب زیر:
  - زبانهای برنامه نویسی مانا: توسعه زبان برنامه نویسی شی گرا برای کار با بانک اطلاعاتی
  - سیستم های شئ-رابطه ای: اضافه کردن مفاهیم شی گرا و انواع داده پیچیده به زبانهای رابطه ای مثل SQL
- مهمترین مشکلات زبانهای مانا:
  - به دلیل قدرتمندی و انعطاف زیاد زبان برنامه نویسی امکان خرابی در بانک اطلاعات افزایش می یابد.
  - پیچیدگی زبانها بهینه سازی اتوماتیک سطح بالا را مشکل می کند.
  - پردازش پرس و جو ها به صورت بیانی نخواهد بود.

معایب مدل شئ گرا

- کمبود تجربه کافی
- کمبود استانداردهای لازم
- کمبود مبانی تئوریک قوی
- ضعف از نظر امنیت
- ضعف در پشتیبانی از دیدها
- پیچیدگی بالا
- یکی از بهترین و رایج ترین سیستم های مدیریت بانک اطلاعات مبتنی بر مدل شئ گرا O_2 است.
- متخصصان بانک اطلاعات برای یافتن مدلی بهتر، به تحقیق و پژوهش ادامه دادند که نتیجه آن ارائه مدل‌های داده‌ای دیگری از جمله مدل شئ-رابطه‌ای بود.
- یکی ازمهمترین ابزارها برای طراحی بانک اطلاعات شئ-رابطه‌ای UML است.

XML و بانک اطلاعات XML

- XML(eXtensible Markup Language)
- در سال ۱۹۹۶ توسط انجمن W3C(World Wide Web Consortium) معرفی شد.
- علی رغم شباهت بسیار زیاد به HTML اهداف طراحی آنها متفاوت است.
- HTML به منظور نمایش داده و XML برای ذخیره سازی و انتقال داده طراحی گردیده است.
- در HTML، tag ها ثابت هستند و در XML، تعریف tag در اختیار کاربر است.
- XML به عنوان مکمل HTML طراحی شده است.
- XML رایج‌ترین ابزار انتقال داده بین انواع مختلفی از برنامه های کاربردی است.

XML و بانک اطلاعات XML

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>

ویژگی های جالب XML

- برچسب یا Tag از پیش تعریف شده ای ندارد.
- اسناد XML خود تعریف هستند.
- XML داده را از HTML جدا می کند.
- XML اشتراک داده را آسان می کند.
- XML انتقال داده را آسان می کند.
- XML تغییرات Platform را آسان می کند.
- XMLبرای تولید زبان های اینترنتی بسیاری به کار می رود.

ساختار اسناد XML

- اسناد XML از واحدهایی به نام المان تشکیل شده اند.
- هر المان در سند XML می تواند رفتارهایی داشته باشد که اطلاعات اضافه ای درباره آن در اختیار ما قرار می دهند.
- هر المان ممکن است حاوی المان های دیگر، متن و یا ترکیبی از آنها می باشد.
- المان ها در ساختار XML یک ساختار درختی را تشکیل می دهند که از یک ریشه آغاز می شود و به برگ ها منتهی می شود.
- هر سند XML باید یک المان ریشه داشته باشد. این المان پدر تمامی المان های دیگر موجود در آن سند است.

ساختار اسناد XML

- مثال: سند XML مربوط به داده های آموزش

```
<edu>
<stud>
        <sname>mohamadi</sname>
        <city>tehran</city>
        <avg>17.42</avg>
        <crs>
        <c#>10172</c#>
        <cname>simulation</cname>
        <unit>3</unit>
        </crs>
</stud>
</edu>
```

قوانین نحوی XML

- عناصر XML باید برچسب خاتمه داشته باشند.
- برچسب های XML نسبت به بزرگ و کوچک بودن حروف حساس هستند.
- برچسب های عناصر تو در تو (nested) باید به همان ترتیبی که باز شده اند بسته شوند.
- اسناد XML باید یک عنصر ریشه داشته باشند.
- مقادیر رفتارهای عناصر XML باید در " " قرار گیرند.
- برخی از کاراکترها در XML معنای خاصی دارند. باید به جای آنها از اشاره گرها استفاده کرد. در زبان XML به این اشاره گرها موجودیت گویند.
- نحوه نوشتن توضیحات در قوانین نحوی XML به صورت زیر است:

```
<!-- This is a comment -->
```

- در حالت پیش فرض، تمام کاراکترهای فضای خالی در نظر گرفته می شوند و از هیچ یک چشم پوشی نمی شود.

قوانین نحوی XML

- در نامگذاری المان ها نیز باید از قوانین زیر پیروی کرد:
  - نام ها می توانند شامل حروف، اعداد و یا کاراکترهای دیگر باشند.
  - نام ها نمی توانند با یک عدد یا یک کاراکتر نشانه گذاری آغاز شوند.
  - نام ها نمی توانند با xml یا Xml و یا هر ترکیب دیگر آن آغاز شوند.
  - نام ها نمی توانند دارای کاراکتر فضای خالی باشند.
  - در XML هیچ کلمه رزرو شده ای وجود ندارد.

موجودیت های از پیش تعریف شده XML

کاراکتر	اشاره گر (علامت ارجاع)
<	<
>	>
&	&
'	'
" "	"

قوانین نحوی XML

- به یک سند XML که از لحاظ نحوی درست باشد، خوش ترکیب می گویند. اما خوش ترکیب بودن یک سند XML تضمینی برای عدم وجود خطا در آن نیست.
- برای اعتبارسنجی یک سند XML باید ساختار مجاز واحدهای سازنده آن را با استفاده از زبانهای DTD یا XML Schema تعریف کرد.
- اعتبارسنجی اسناد XML به ویژه برای حصول اطمینان از تفسیرهای درست از داده های دریافتی اهمیت دارد.
- هنگامی که داده ای از فرستنده به گیرنده ارسال می شود، اینکه هر دو طرف تفسیرهای یکسانی از محتوای آن داده داشته باشند اهمیت می یابد.

ساختار بانک اطلاعات XML

- تعریف ساختار با DTD (Document Type Definition)
- یک DTD لیستی از ساختارهای مجاز واحدهای سازنده یک سند XML را توصیف می کند. دو روش برای تعریف DTD وجود دارد:

  ۱- تعریف در داخل سند XML به عنوان بخشی از خود سند که به آن DTD داخلی می گویند.

  ۲- تعریف در خارج از سند XML و ارجاع به آن در داخل سند که به آن DTD خارجی می گویند.

- در صورتیکه DTD داخلی باشد باید در یک برچسب DOCTYPE با قالب زیر قرار گیرد:
- <! DOCTYPE root-element [element-declaration]>
- در صورتیکه DTD خارجی باشد در یک فایل جداگانه ذخیره می گردد و در سند XML با برچسب DOCTYPE با قالب زیر مورد ارجاع قرار می گیرد:
- <! DOCTYPE root-element SYSTEM "filename">

ساختار بانک اطلاعات XML

❖ مثال: DTD مربوط به سیستم آموزش که به صورت داخلی تعریف شده است:

```
<!DOCTYPE edu[
<! ELEMENT edu (book*)>
<! ELEMENT book(title,authors+,chapter*,ref*)>
<! ELEMENT chapter(text | section)*>
<! ELEMENT ref book>
<! ELEMENT title #PCDATA>
<! ELEMENT authors #PCDATA>
<! ELEMENT section #PCDATA>
<! ELEMENT text #PCDATA>]
>
```


ساختار بانک اطلاعات XML

❖ مثال: DTD که به صورت خارجی تعریف شده است:

```
<! ELEMENT edu (book*)>
<! ELEMENT book(title,authors+,chapter*,ref*)>
<! ELEMENT chapter(text | section)*>
…
<! ELEMENT text #PCDATA>
```

محتویات فوق در فایل education.dtd قرار دارد. در سند XML مربوط به سیستم آموزش به صورت زیر ارجاع می شود.

```
<!DOCTYPE edu SYSTEM "education.dtd">
<edu>
        <stud>
        …
        </stud>
</edu>
```

ساختار بانک اطلاعات XML

- عبارت SYSTEM که در مثال قبل استفاده شد به این معنا است که از یک فایل dtd شخصی استفاده می شود که شناخته شده نیست. چنانچه از یک dtd عمومی و شناخته شده استفاده شود به صورت زیرعمل می شود:

```
<!DOCTYPE note PUBLIC "-//liz Castro//DTD End_species//EN//"
"http://www.cookwood.com/xml/examples/dtd_creating/end_species.dtd">
```

- برای تعریف DTD به صورت عمومی از شناسه عمومی استفاده می کنیم. یک شناسه عمومی دارای فرم زیر است:
- reg.type//owner//DTD description//language//
- reg.type: اگر DTD استاندارد باشد و به ثبت رسیده باشد علامت + و اگر غیراستاندارد باشد علامت - می گذاریم.
- owner: نام DTD یا نام شرکتی است که DTD را به ثبت رسانده است.
- description: یک توضیح متنی ساده در خصوص DTD است.
- language: کد دو حرفی استاندارد زبانی است که DTD به آن زبان نوشته شده است.

ساختار بانک اطلاعات XML

- اصلی ترین واحد سازنده اسناد XML المان ها هستند اما از دیدگاه DTD همه اسناد XML از واحدهای زیر تشکیل شده اند:

PCDATA	CDATA	
المان (ELEMENT)	رفتار (ATTRIBUTE)	موجودیت‌ها (ENTITIES)

- PCDATA
  - ✓ معنای تحت الفظی PCDATA، داده کاراکتری قابل پیمایش است.
  - ✓ داده کاراکتری همان متنی است که بین برچسب آغازین و پایانی یک المان ظاهر می شود. PCDATA متنی است که توسط پیمایش گر پیمایش می شود.
  - ✓ در واقع چون یک PCDATA کاملاً پیمایش و بررسی می گردد نمی توانیم در آن از کاراکترهایی مانند "<" به صورت مستقیم استفاده کنیم و باید حتما به جای آنها از اشاره گرهای متناظرشان استفاده نماییم.

ساختار بانک اطلاعات XML

❖ CDATA

✓ CDATA نیز به معنای داده کاراکتری است با این تفاوت که پیمایش نمی شود و در نتیجه محدودیت های PCDATA را از لحاظ استفاده مستقیم از کاراکترهای خاص ندارد. در صورت استفاده از کاراکتری مانند ">" هیچ خطایی ایجاد نمی‌گردد زیرا CDATA پیمایش نمی شود.

❖ المان (ELEMENT)

✓ المان ها در DTD به یکی از دو صورت زیر تعریف می شوند:

<! ELEMENT element-name category>

<! ELEMENT element-name (element-content)>

✓ و از لحاظ گروه بندی (category) به دو دسته زیر تقسیم می شوند:

۱- المان تهی: <! ELEMENT element-name EMPTY>

۲- المان های any: <! ELEMENT element-name ANY>

ساختار بانک اطلاعات XML

✓ المان ها از لحاظ محتوا به چند دسته تقسیم می شوند:

✓ المان هایی که حاوی PCDATA یا CDATA هستند.

<! ELEMENT element-name (#CDATA)>

<! ELEMENT element-name (#PCDATA)>

✓ المان هایی که حاوی المان (های) دیگر بعنوان فرزند هستند:

<! ELEMENT element-name (child1)>

<! ELEMENT element-name (child1, child2, …)>

تعداد دفعات ظاهرشدن المان‌های فرزند را می توان با علائم خاصی مشخص کرد:

معنا	قالب
المان فرزند دقیقاً یک بار ظاهر می شود.	<!ELEMENT element-name (child-name)>
المان فرزند یک بار یا بیشتر می تواند تکرار شود.	<!ELEMENT element-name (child-name+)>
المان فرزند میتواند از صفر تا بینهایت بار ظاهر شود.	<!ELEMENT element-name (child-name*)>
المان فرزند یا اصلا وجود نداردیا حداکثر یکبار ظاهرشود.	<!ELEMENT element-name (child-name?)>

ساختار بانک اطلاعات XML

❖ رفتار (ATTRIBUTE)

✓ برای تعریف رفتار از قالب زیر استفاده می شود:

<!ATTLIST element-name attribute-name attribute-type default-value>

✓ نوع رفتار می تواند یکی از موارد زیر باشد:

نوع رفتار	توصیف
CDATA	مقدار مجاز از نوع داده کاراکتری است.
(en1|en2|...)	مقدار مجاز یکی از enumeration های موجود در لیست می باشد.
ID	مقدار یک شناسه منحصر به فرد است.
IDREF	مقدار شناسه یک المان دیگر است.
IDREFS	مقدار لیستی از شناسه های المان دیگر است.
NMTOKEN	مقدار یک نام معتبر در زبان XML است.
NMTOKENS	مقدار لیستی از نام های معتبر در زبان XML است.
ENTITY	مقدار یک موجودیت است.
ENTITIES	مقدار لیستی از موجودیت ها است.
NOTATION	مقدار نام یک نماد است.
XML	یک مقدار از پیش تعریف شده در زبان XML است.

ساختار بانک اطلاعات XML

✓ مقدار پیش فرض یک رفتار می تواند یکی از مقادیر زیر باشد:

مقدار	توضیح
VALUE	یک مقدار دلخواه
#REQUIRED	نشان دهنده الزامی بودن وجود رفتار است
#IMPLIED	هیچ الزامی در وجود رفتار نیست
#FIXED	مقدار رفتار ثابت و غیرقابل تغییر است

✓ المان یا رفتار

✓ یکی از نکات مهم و قابل توجه، تصمیم گیری در خصوص زمان استفاده از رفتار یا المان است.

✓ بهترین راهکار این است که تنها اطلاعاتی را که واقعاً بخشی از داده های مورد نظر ما هستند بعنوان المان تعریف کنیم و اطلاعاتی که بخشی از داده های ما نیستند اما مشخصات اضافی و مفیدی درباره آنها در اختیار ما قرار می دهند بعنوان رفتار ذخیره نماییم.

ساختار بانک اطلاعات XML

- موجودیت ها (ENTITIES)
  - ✓ در واقع اشاره گرهایی (pointer) به یک قطعه متن و یا کاراکترهای استاندارد هستند که به دو گروه تقسیم می شوند:
  - ✓ موجودیت های پیش فرض مانند آنچه در قسمت قوانین نحوی آورده شد.
  - ✓ موجودیت های کاربر تعریف (user defined) که توسط کاربر ایجاد می شوند و می توانند داخلی یا خارجی باشند.
  - ✓ قالب تعریف موجودیت کاربر تعریف داخلی:

<! ENTITY entity-name "entity-value">

  - ✓ قالب تعریف موجودیت کاربر تعریف خارجی:

<! ENTITY entity-name SYSTEM "URI/URL">

  - ✓ در هر دو صورت برای استفاده از موجودیت تعریف شده از قالب زیر استفاده می شود: &entity-name;

ساختار بانک اطلاعات XML

- تعریف ساختار با XML Schema
- XML Schema نیز مانند DTD زبانی برای توصیف ساختارهای مجاز اسناد XML است با این تفاوت که خود نیز بر پایه XML می باشد.
- اهم ویژگی های XML Schema نسبت به DTD:
- XML Schema از انواع مختلف داده پشتیبانی می کند و در نتیجه انجام عملیات زیر ساده تر می گردد:
  - تعریف محتویات مجاز اسناد XML
  - اعتبار سنجی صحت داده ها
  - کار کردن با داده های یک پایگاه داده
  - تعریف محدودیت ها برای داده ها
  - تعریف الگوها یا قالب های داده ای
  - تبدیل داده ها به انواع مختلف

ساختار بانک اطلاعات XML

- XML Schema از قوانین نحوی XML استفاده می نماید. این امر مزایای زیر را دارد:
  - نیازی به آموختن یک زبان جدید نیست.
  - می توان از همان ویرایشگری که برای ویرایش XML استفاده می شود برای ویرایش فایل های XML Schema به کار برد.
  - می توان همان پیمایشگر مورد استفاده برای XML را جهت پیمایش فایل های XML Schema به کار برد.
- XML Schema باعث ایجاد امنیت در ارتباطات داده ای می گردد (تفسیر یکسان از داده توسط فرستنده و گیرنده).
- XML Schema قابل گسترش است. این قابلیت امکانات زیر را فراهم می کند:
  - استفاده مجدد از XML Schemaهای دیگر (Reusability)
  - ایجاد انواع داده کاربر تعریف بر اساس انواع داده استاندارد
  - ارجاع به چند XML Schema در یک سند XML

ساختار بانک اطلاعات XML

- از دیدگاه XML Schema المان های سازنده اسناد XML را می توان به دو دسته المان های ساده و المان های پیچیده تقسیم کرد.
- المان های ساده: المان هایی که تنها می توانند حاوی متن باشند و نمی توانند هیچ رفتار یا المان فرزندی داشته باشد.
- متن موجود در المان ساده می تواند از انواع مختلف اعم از انواع از پیش تعریف شده در XML Schema یا انواع کاربر تعریف باشد.
- می توان محدودیت هایی برای مقادیر مجاز این انواع داده ای قرار داد یا آنها را ملزم به مطابقت با الگویی خاص نمود.
- هر چند المان های ساده فاقد رفتار می باشند اما رفتارهایی که در XML Schema تعریف می شوند بعنوان یک المان ساده در نظر گرفته می شوند.

ساختار بانک اطلاعات XML

- تعریف المان های ساده در XML Schema به صورت زیر است:

```
<xs: element name="someName" type="someType"/>
```

- انواع داده از پیش تعریف شده و رایج در XML Schema:

(xs:time, xs:date, xs:boolean, xs:integer, xs:decimal, xs:string)

- المان های ساده ممکن است مقادیر ثابت یا پیش فرض داشته باشند که در قالب های زیر مشخص می شوند:

```
<xs: element name="someName" type= "someType" default= "value"/>
<xs: element name="someName" type= "someType" fixed= "value"/>
```

- از آنجا که رفتارها بعنوان المان ساده در نظر گرفته می شوند، قالب آنها به شکل زیر تعریف می شود:

```
<xs:attribute name="someName" type="someType"/>
```

- و اگر دارای مقادیر ثابت یا پیش فرض باشند:

```
<xs:attribute name="someName" type="someType" default="value"/>
<xs:attribute name="someName" type="someType" fixed="value"/>
```


ساختار بانک اطلاعات XML

- وجود یک رفتار ممکن است اختیاری(optional) یا الزامی(required)باشد. حالت پیش فرض اختیاری است مگراینکه الزامی‌بودن را با قالب زیر بیان کنیم:

```
<xs:attribute name="someName" type="someType" use="required"/>
```

- برای توصیف مقادیر مجاز المان ها و رفتارها از محدودیت ها استفاده می شود.
- اعمال محدودیت به‌صورت یک بازه مجاز (avg بامقادیر مجاز بین صفرتا ۲۰)

```
<xs:element name="avg">
        <xs:simpleType>
                <xs:restriction base="xs:integer">
                        <xs:minInclusive value="0"/>
                        <xs:maxInclusive value="20"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- محدودیت به صورت مجموعه ای از مقادیر مجاز (روش اول)
- تنها دروس DDB، compiler، و distributed ارائه شود.

```
<xs:element name="cname">
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="DDB"/>
                        <xs:enumeration value="compiler"/>
                        <xs:enumeration value="distributed .."/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- محدودیت به صورت مجموعه ای از مقادیر مجاز (روش دوم)

```
<xs:element name="car" type="courseType"/>

<xs:simpleType name="courseType">
        <xs:restriction base="xs:string">
                <xs:enumeration value= "DDB "/>
                <xs:enumeration value="compiler"/>
                <xs:enumeration value="distributed .."/>
        </xs:restriction>
</xs:simpleType>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان initList نشان دهنده حرف اول نام خانوادگی دانشجو است که می تواند یکی از مقادیر کاراکترهای a تا z را داشته باشد.

```
<xs:element name="initLast">
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:pattern value="[a-z]"/>
                </xs:restriction>
        </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان initList نشان دهنده سه حرف اول نام خانوادگی دانشجو است که می تواند دنباله ای از سه کاراکتر A تا Z را داشته باشد.

```
<xs:element name="initLast">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان level نشان دهنده مقطع تحصیلی دانشجو است که می تواند یکی از مقادیر (A: کارشناسی، B: کارشناسی ارشد، C: دکتری) را داشته باشد.

```
<xs:element name="level">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[ABC]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان userName نشان دهنده نام کاربری دانشجو است که می تواند دنباله ای از شش کاراکتر A تا Z و a تا z را داشته باشد.

```
<xs:element name="userName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z] [a-zA-Z] [a-zA-Z] [a-zA-Z] [a-zA-Z] [a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان password نشان دهنده رمز عبور دانشجو است که می تواند دنباله ای ۶ رقمی از ارقام ۰ تا ۹ باشد.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9] [0-9] [0-9] [0-9] [0-9] [0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان password نشان دهنده رمز عبور دانشجو است که می تواند ۸ کاراکتر از a تا z یا A تا Z و یا ۸ رقم از ۰ تا ۹ را بپذیرد.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت به صورت دنباله ای از مقادیر مجاز
- المان comment نشان دهنده توضیحات مربوط به دانشجو است که می تواند دنباله ای از هر تعداد (حتی صفر) کاراکتر را با هر ترتیبی بپذیرد.

```
<xs:element name="comment">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- اعمال محدودیت روی کاراکتر فضای خالی
- المان address نشان دهنده آدرس دانشجو است که در آن تمام فضاهای خالی در نظر گرفته می شوند.

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="preserve"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت روی کاراکتر فضای خالی
- المان address نشان دهنده آدرس دانشجو است که در آن تمام فضاهای خالی حذف می شوند.

```
<xs:element name="address">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- اعمال محدودیت روی طول
- المان password نشان دهنده رمز عبور دانشجو است که طول آن ۸ کاراکتر است.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

ساختار بانک اطلاعات XML

- اعمال محدودیت روی طول
- المان password نشان دهنده رمز عبور دانشجو است که طول آن بین ۵ تا ۸ کاراکتر است.

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```


ساختار بانک اطلاعات XML

- المان های پیچیده را می توان به ۴ نوع مختلف تقسیم کرد:
  - ✓ المان های تهی: این نوع المان ها حاوی متن و یا المان دیگر نیستند بنابراین درصورتی پیچیده می شوند که دارای رفتار باشند.
  - ✓ المان هایی که حاوی المان های دیگری هستند.
  - ✓ المان هایی که تنها حاوی متن هستند و هیچ المان فرزندی ندارند اما دارای رفتار هستند.
  - ✓ المان هایی که هم شامل متن و هم دارای المان دیگری هستند و ممکن است رفتار داشته باشند یا نداشته باشند.

ساختار بانک اطلاعات XML

❖ قالب تعریف المان های تهی پیچیده که تنها دارای رفتار هستند (روش اول)

```
<xs:element name="someName">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="baseType">
        <xs:attribute name="someName" type="someType"/>
        ……
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```


ساختار بانک اطلاعات XML

❖ قالب تعریف المان های تهی پیچیده که تنها دارای رفتار هستند (روش دوم)

```
<xs:element name="someName">
  <xs:complexType>
    <xs:attribute name="someName" type="someType"/>
  </xs:complexType>
</xs:element>
```

ساختار بانک اطلاعات XML

❖ قالب تعریف المان هایی که حاوی المان های دیگر هستند

```
<xs:element name="someName" type="myComplexType"/>

<xs:complexType name="myComplexType">
  <xs:sequence>
    <xs:element name="xxx" type="yyy"/>
    <xs:element name="zzz" type="eee"/>
    …
  </xs:sequence>
</xs:complexType>
```


ساختار بانک اطلاعات XML

❖ قالب تعریف المان های پیچیده ای که تنها دارای متن هستند و چون پیچیده هستند رفتار نیز دارند (روش اول)

```
<xs:element name="someName">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        <xs:attribute name="someName" type="someType"/>
        …
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

ساختار بانک اطلاعات XML

❖ قالب تعریف المان های پیچیده ای که تنها دارای متن هستند و چون پیچیده هستند رفتار نیز دارند (روش دوم)

```
<xs:element name="someName">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        <xs:attribute name="xxx" type="yyy"/>
        …
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```


ساختار بانک اطلاعات XML

❖ قالب تعریف المان هایی پیچیده که هم متن و هم المان های دیگری دارند (روش اول)

```
<xs:element name="someName">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="xxx" type="yyy"/>
      <xs:element name="zzz" type="eee"/>
        …
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

ساختار بانک اطلاعات XML

❖ قالب تعریف المان هایی پیچیده که هم متن و هم المان های دیگری دارند (روش دوم)

```
<xs:element name="someName" type="myComplexType"/>

<xs:complexType name="myComplexType" mixed="true">
  <xs:sequence>
    <xs:element name="xxx" type="yyy"/>
    <xs:element name="zzz" type="eee"/>
      …
  </xs:sequence>
</xs:complextype>
```


ساختار بانک اطلاعات XML

❖ شاخص ها

❖ شاخص ها برای کنترل چگونگی ظاهر شدن المان های فرزند در اسناد XML به کار می روند و به سه دسته تقسیم می شوند:

- ❖ شاخص های ترتیب
- ❖ شاخص های تکرار
- ❖ شاخص های گروه بندی

ساختار بانک اطلاعات XML

- شاخص های ترتیب
- جهت تعیین ترتیب ظاهرشدن المان ها به‌کار می‌روند. شامل موارد زیر هستند:
  - all: المان‌های فرزند می توانند با هر ترتیبی و فقط یک بار ظاهر می شوند.
  - choice: نشان می دهد که یکی از المان‌های فرزند می تواند ظاهر شود.
  - sequence: المان‌های فرزند باید به همان ترتیبی که تعریف‌شده‌اند ظاهرشوند.
- قالب استفاده از شاخص های ترتیب:

```
<xs:element name="someName">
  <xs:complexType>
    <xs:one of the order indicators (all/choice/sequence)>
      <xs:element name="xxx" type="yyy"/>
      <xs:element name="zzz" type="eee"/>
      …
    </xs:one of the order indicators (all/choice/sequence)>
  </xs:complexType>
</xs:element>
```


ساختار بانک اطلاعات XML

- شاخص های تکرار
- برای تعیین تعداد دفعات ممکن ظاهرشدن المان‌های فرزند به کار می‌روند:
- maxOccurs: نشان‌دهنده حداکثر تعداد دفعاتی است که یک المان فرزندمی‌تواند تکرار شود.
- minOccurs: نشان‌دهنده حداقل تعداد دفعاتی است که یک المان فرزند می‌تواند تکرار شود.
- این شاخص ها به عنوان رفتاری برای المان های فرزند در نظر گرفته می شود. البته یک رفتار اختیاری است که در نتیجه ممکن است هر یک برای بعضی از المان ها وجود داشته باشند و برای بعضی دیگر وجود نداشته باشند.

```
<xs:element name="someName">
  <xs:complextype>
    <xs:sequence>
      <xs:element name="xxx" type="yyy"/>
      …
      <xs:element name="zzz" type="eee" maxOccurs="somenumber"
minOccurs="somenumber"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

ساختار بانک اطلاعات XML

- شاخص های گروه بندی
- برای تعیین مجموعه ای از المان ها یا رفتارهای مرتبط به کار می روند و شامل موارد زیر هستند:
- group: برای گروه بندی المان هایی که ترتیب آنها با یکی از شاخص های ترتیب مشخص شده به کار می رود.
- قالب تعریف آن به شکل زیر می باشد:

```
<xs:group name="groupname">
  some ordered element definition
</xs:group>
```

- پس از تعریف یک گروه از المان ها می توانیم هر جا که به همان ترتیب از همان المان ها نیاز داشتیم با استفاده از قالب زیر به آن ارجاع نماییم:

```
<xs:group ref="groupname"/>
```


ساختار بانک اطلاعات XML

- شاخص های گروه بندی
- Attribute group: برای گروه بندی تعدادی رفتار به کار می رود.
- قالب تعریف آن به شکل زیر می باشد:

```
<xs:attributeGroup name="groupname">
  some attribute definition
</xs:attributeGroup>
```

- پس از تعریف گروهی از رفتارها می توانیم هر جا که به همان گروه از رفتارها نیاز داشتیم با استفاده از قالب زیر به آن ارجاع نماییم:

```
<xs:attributeGroup ref="groupname"/>
```

پرس و جو در بانک اطلاعات XML

- پرس و جو در بانک اطلاعات XML
- پردازش پرس و جو در بانک اطلاعات و تبدیل داده از یک Schema به Schemaی دیگر (transforming) بسیار مرتبط هستند و معمولا توسط ابزارهای یکسانی به انجام می رشند.
- رایج ترین زبان های استاندارد XPath، XQueryو XSLT است.

پرس و جو در بانک اطلاعات XML

- زبان پرس و جوی Xpath
- در این زبان برای انتخاب هر داده از یک سند XML، از عبارت مسیری (path expression) متناظر با آن داده استفاده می شود (مانند آدرس دهی سیستم فایل در سیستم عامل).
- هر عبارت مسیری دنباله ای از مراحل است که از المان ریشه شروع شده با "/" از هم جدا می شوند و نتیجه آن مقادیر (المانها/صفات) متناظر با آن مسیر خواهد بود. هر مرحله از عبارت مسیری، روی نتیجه مراحل قبل عمل می کند.
- مثال: در سیستم آموزش: /edu/stud/sname

  Output <sname>mohamadi</sname>
  <sname>alinaghizade</sname>
- چنانچه بخواهیم برچسب ها حذف شوند و فقط خود مقدار داده برگردانده شود از تابع text() استفاده می شود: /edu/stud/sname/text()

  Output mohamadi
  alinaghizade

پرس و جو در بانک اطلاعات XML

- زبان پرس و جوی Xpath
- برای گزینش در XPath شرط مورد نظر داخل [] و در آخرین مرحله قرار داده می شود.
- مثال: مشخصات دانشجویان با معدل بالای ۱۸

/edu/stud [Avg>18]

- نام دانشجویان با معدل بالای ۱۸(عملگر پرتو @)

/edu/stud [avg>18] @sname

- توابع و عملگرهای زیادی در XPath پشتیبانی می شوند (مثل تابع (count).
- برای دسترسی به محتویات محلی که یک اشاره گر (یا همان IDREF) به آن اشاره می کند از تابع id() استفاده می شود.
- مثال: اطلاعات رئیس دانشکده

/edu/clg/id(@head)

پرس و جو در بانک اطلاعات XML

- زبان پرس و جوی Xpath
- از نماد | بجای عملگر مجموعه ای اجتماع در جبر رابطه ای استفاده می شود:

/edu/stud [avg>18] | /edu/stud [city='Tehran']

- با استفاده از نمادگذاری مرسوم در سیستم فایل های سیستم عامل، می توان به هر یک از نوه های فرزند، پدر و ... رفت. نماد "//" معنی پرش از چندین مرحله را می دهد و با آن می توان تمام نوه های آن را به دست آورد و یا نماد ".." که پدر یک نوه را نشان می دهد.
- مثال: نام تمام دانشجویان را می دهد.

/edu//stud

پرس و جو در بانک اطلاعات XML

- زبان پرس و جوی XQuery
- ساختار یک پرس و جو در زبان XQuery (به نام عبارت FLWR) شامل بخش های زیر است: For …… Let …… Where ……Return
- for معادل from، where معادل where و return معادل select در زبان SQL هستند و Let معادلی در SQL ندارد و برای تعریف متغیرهای موقتی لازم به کار می رود.
- در بخش for از عبارات مسیری استفاده می شود و نتیجه محاسبه شده را در یک متغیر می ریزند. با کمک بخش های where و let داده مورد نظر را از این متغیر جدا کرده و این داده در عبارت Return توسط برچسب های مناسب و در قالب مورد نظر برگردانده می شود.
- مثال:

```
For $x in /edu/stud
Let $average:=$x/@avg
Where $x/avg >18
Return <average>$average</average>
```

- معادل Query بالا

```
For $x in /edu/stud [avg>18]
Return <average>$x/@avg</average>
```

بانکهای اطلاعات مبتنی برشیء و XML
XML و بانک اطلاعات XML

پرس و جو در بانک اطلاعات XML

- زبان پرس و جوی XQuery
- توابع زیادی از جمله distinct() (برای حذف جواب های تکراری)، count()، sum() و ... قابل استفاده می باشند.
- در XQuery از عملگر گروه بندی (group-by) پشتیبانی نمی شود اما می توان با استفاده از عبارت FLWR دورن عبارت return (یعنی FLWR تودرتو) جواب های محاسبه شده را گروه بندی کرد.
- در XQuery به جای عملگر id() زبان XPath، از عملگر پیکان (arrow) → استفاده می شود.

ذخیره سازی در بانک اطلاعات XML

- دو راه حل کلی برای ذخیره سازی در بانک اطلاعات XML وجود دارد:
۱. ذخیره سازی به صورت رابطه ای (پشتیان XML):
- باید داده های XML را به فرم رابطه تبدیل کرد.
- مزیت: پس از تبدیل یک سیستم بانک اطلاعات رابطه‌ای خواهیم داشت.
- عیب: سربار (overhead) انجام تبدیلات لازم برای داده ها و پرس و جوها.
۲. ذخیره سازی به صورت غیر رابطه ای (ذاتا XML): به دو طریق امکانپذیر
- استفاده از فایل های معمولی (flat files): اگر چه با مدل XML سازگاری دارد ولی نوعی بازگشت به عقب و استفاده از همان سیستم های پردازش فایل و مشکلات مربوط به آنها (مثل نداشتن همروندی، ترمیم و ...) است.
- بانک های اطلاعات ذاتاً XML (native XML): نیاز به سیستم های بانک اطلاعات جدیدی داریم که اساساً مدل داده ای آنها XML باشد. در حال حاضرسیستم تجاری با این مشخصه رایج و مرسوم نمی باشد.

ذخیره سازی در بانک اطلاعات XML

- مزایای روش ذاتاً XML:
- بازیابی ساده اطلاعات، حفظ ساختار اصلی، عدم نیاز به ترجمه و نگاشت (mapping)، سرعت بالای دستیابی.
- معایب روش ذاتاً XML:
- پیچیدگی مدیریت و نگهداری داده های غیر XML، فقدان سیستم های تجاری رایج.

- در ذخیره سازی بصورت رابطه ای (XML-enabled) سه انتخاب داریم:
- نمایش داده ها بصورت رشته (string)
- نمایش داده ها بصورت درخت (tree)
- نگاشت به رابطه ها

ذخیره سازی در بانک اطلاعات XML

- نمایش رشته ای
- ساده ترین روش نگاشت در پایگاه داده رابطه ای است هر المان اصلی به صورت یک رابطه ذخیره می شود و زیر المان های آنها به عنوان صفتی از رابطه که مقدار آنها می تواند از نوع رشته باشد(جداول استاد و دانشجو ...).
- با توجه به اینکه شمای سند ذخیره نمی شود امکان پرس و جوی داده ها به صورت مستقیم وجود ندارد. راه حل ساده آنست که المان ها با نوع های مختلف در جداول متفاوت ذخیره شوند. در این صورت امکان پرس و جو داده ها به صورت ساده بوجود می آید. برای این کار نیز دو حالت ممکن است:
  - ✓ استفاده از یک رابطه برای نگهداری تمام المان ها
  - ✓ استفاده از رابطه های جداگانه برای هر یک از المان ها

ذخیره سازی در بانک اطلاعات XML

- به دلیل اینکه عنوان و مقادیر المان ها بصورت متن کامل در یک رابطه نگهداری می شوند، برای دسترسی سریع به یک مقدار خاص نیاز به شاخص روی برخی از المان ها و یا مقادیر آنها داریم.
- مزایا:
- در این روش نیازی نیست که حتماً DTD را داشته باشیم.
- اگر المان های سطح بالا خوب پخش شده باشند می توانند تا حد زیادی برای برخی پرسش ها کار را سریع کند.
- معایب:
- تا حد زیادی برای برخی پرسش ها هزینه جستجوی خطی را داریم.
- به تعداد زیادی شاخص احتیاج پیدا خواهد شد که باید آنها را به خوبی مدیریت و کنترل کرد.

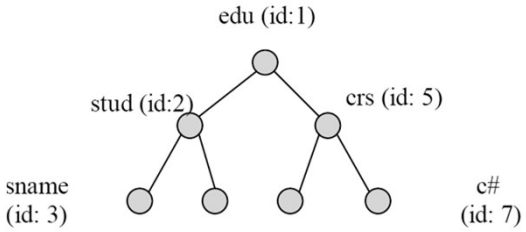
ذخیره سازی در بانک اطلاعات XML

- نمایش درختی
- یک مدل درختی از داده XML به کمک رابطه های زیر ساخته می شود.
- برای اطلاعات گره ها

nodes(id, type, label, value)

- برای ارتباط بین گره ها

edge(child-id, parent-id)



ذخیره سازی در بانک اطلاعات XML

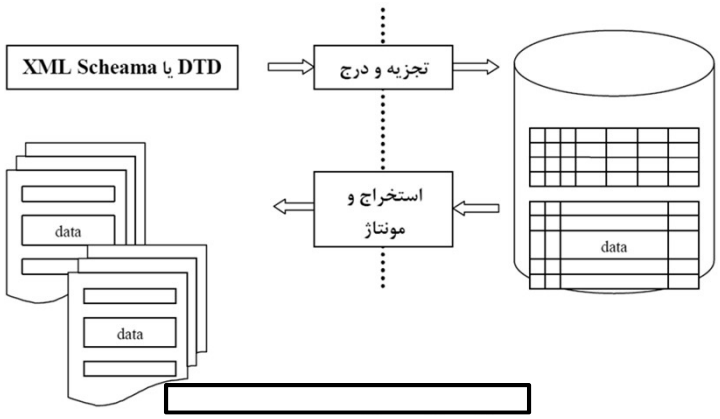
- به هر المان یک شناسه یکتا داده می شود و به ازای هر المان یک تاپل در رابطه nodes ذخیره می شود. از رابطه edge برای ایجاد ارتباط بین المان ها استفاده می شود. می توان کل سند XML را بصورت رابطه ای نشان داد و بسیاری از پرس و جوها قابل اجرا می باشند.
- داده های XML را به صورت درختواره در نظر گرفته سپس درخت حاصل را به کمک دو رابطه ذکر شده ذخیره سازی می نمایند.
- مزایا:
- از داشتن DTD آزاد هستیم (بدون DTD می توان اسناد را ذخیره کرد).
برای حفظ ترتیب المانها می توان از یک صفت استفاده کرد.
- معایب:
- نظم ساختاری (یعنی امکان خاصی برای قرار دادن فرزند و والد در کنار یکدیگر) در نظر گرفته نمی شود و برای کوچکترین درخواستی باید چندین ادغام انجام دهیم.

ذخیره سازی در بانک اطلاعات XML

- نگاشت به رابطه ها
- المان هایی که دارای شمای مشخص می باشد به صورت رابطه ای نگاشت می شوند و در صورتی که شمای مشخصی نداشته باشد به صورت رشته ای یا درختی ذخیره می شوند.
- به ازای هر نوع المان که شمای مشخص دارد یک رابطه ساخته می شود و هر صفت المان یک صفت از رابطه را تشکیل می دهد.
- زیر المان ها که تنها یک بار درون این المان ها قرار بگیرند، به عنوان یک صفت از المان ذخیره می شوند، در غیر این صورت (اگر بیش از یک بار وجود داشته باشند) یک رابطه مناسب با زیرالمان ساخته می شود.
- ارتباط بین رابطه ها به کمک رابطه nodes برقرار می شود.
- در این روش به DTD نیاز می باشد و از روی DTD گراف ساخته می شود.
- گراف حاصل به رابطه های معادل آن تبدیل می شود.

ذخیره سازی در بانک اطلاعات XML

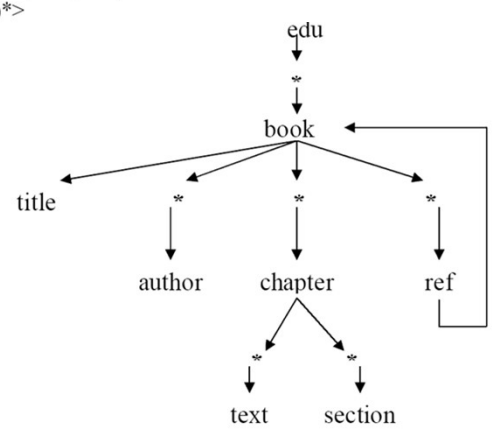
- معماری ساده از بانک اطلاعات XML-Enabled



ذخیره سازی در بانک اطلاعات XML

❖ گراف معادل DTD اطلاعات کتاب ها

```
<! ELEMENT edu (book*)>
<! ELEMENT book (title, authors*, chapter*, ref*)>
<! ELEMENT chapter (text | section)*>
<! ELEMENT ref book>
<! ELEMENT title #PCDATA>
<! ELEMENT author #PCDATA>
<! ELEMENT section #PCDATA>
<! ELEMENT text #PCDATA>
```



ذخیره سازی در بانک اطلاعات XML

❖ یکی از الگوریتم های معمول تبدیل، الگوریتم قطعه بندی داخلی (internal partitioning) است.

❖ در این الگوریتم درخت حاصل از DTD با الگوریتم پیمایش عمقی پیمایش می‌شود. در این پیمایش برای گره های با خصوصیات زیر رابطه ایجاد می شود.

❖ ریشه

❖ هر گره ای که نود قبلی آن * باشد.

❖ هر گره ای که بازگشتی باشد.

❖ هر گره با ورودی بیشتر از یک

❖ هر گره‌ای که یک ورودی داشته باشد به صورت صفت برای پدر خود می آید.

❖ برای بیان ارتباط ها شماره یکتای پدر در رابطه های فرزندانش می آید.

❖ از ترتیب شماره دهی می توان برای نگهداری ترتیب استفاده کرد.

❖ خصوصیات یک المان نیز در رابطه مربوط به آن به عنوان صفت می آید.

ذخیره سازی در بانک اطلاعات XML

- رابطه های معادل اطلاعات کتاب عبارتند از:

edu(eduID)
book(bookID, parentID, code, title: string)
author(authorID, bookID, author: string)
chapter(chapterID, bookID)
ref(refID, bookID)
text(textID, chapterID, text: string)
section(sectionID, chapterID, section: string)

- در این روش ذخیره‌سازی اطلاعات بصورت بهینه و نرمال‌تری صورت می‌گیرد.
- تبدیل درخواست های مرتبط با XML به SQL ساده تر می باشد.
- اما حتما باید DTD وجود داشته باشد.
- Overhead نسبتاً زیادی برای تبدیل schemaها وجود دارد.

ذخیره سازی در بانک اطلاعات XML

- نمایش سند XML در وب با XSLT و CSS
- XSLT (EXtensible Stylesheet Language Transformations) سندی است که به فرمت اسناد XML نوشته می شود و وظیفه اصلی آن تبدیل سند XML به HTML و یا XHTML 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.
- یکی دیگر از ابزار نمایش سند XML در browserها CSS است.
- Browserها 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(Cascading Style Sheets) در واقع نقش تگ های HTML را برای اسناد XML بازی می‌کند و نحوه نمایش هر یک از داده‌های سند XML را معین می کند. browserها 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 می کند.

ذخیره سازی در بانک اطلاعات XML

- بانک اطلاعات XML در SQL:2003
- یکی از مهمترین قابلیتهای جدید SQL:2003 پشتیبانی از مدل داده XML می باشد. امکانات فراوانی از جمله برای تبدیل داده های SQL به XML و بالعکس تعبیه شده است. برخی از مهمترین توابع در زیر آمده است:
- XML element: نام یک المان و مجموعه ای از صفات المانها به صورت دلخواه دریافت کرده و یک نمونه (instance) از نوع XML را برمی گرداند.
- XML forest: هریک از پارامترهایش را به XML تبدیل کرده سپس آنها را به هم متصل (concatenate) نموده و بصورت یک تکه برمی گرداند.
- XML concat: یک سری نمونه از انواع XML به عنوان ورودی دریافت کرده و آنها را به هم متصل می نماید.
- XML agg: تکه هایی را به عنوان ورودی دریافت کرده و مجموعه آنها را به صورت یک سند XML برمی گرداند.

ذخیره سازی در بانک اطلاعات XML

- برای ایجاد، تبدیل، انجام پرس و جو و به روز رسانی داده های XML قابلیتهای زیر ارائه شده است:
- XML ColAttVal: یک تکه XML می سازد. هر تکه XML نام ستونی که به عنوان پارامتر تابع دریافت شده را خواهد داشت.
- XML Sequence: برعکس XML concat عمل می کند.
- ExtractValue: یک نمونه نوع XML و یک عبارت XPath به عنوان پارامتر دریافت کرده و مقدار گره مربوطه را برمی گرداند.
- UpdateXML: می تواند مقداری در سند XML مربوطه را تغییر دهد.

مدیریت تراکنش (مفاهیم)

- امنیت security:
محافظت از داده ها در برابر خطراتی از قبیل آتش سوزی و نیز جلوگیری از دسترسی غیر مجاز به آنها. راه های مختلفی برای جلوگیری از دستیابی غیر مجاز وجود دارد که متداولترین آنها استفاده از رمز و الگویتم های خاص جهت تغییر داده است. البته همواره کسانی هستند که این رمزها و قفل ها را بگشایند.
- جامعیت integrity:
صحت داده‌ها و پردازش‌ها و پیروی از مقررات سیستم. مثلا موجودی حساب بانکی نباید منفی باشد و یا شخص نتواند بیش از موجودی از حسابش برداشت کند.
- هدف اصلی در بانک اطلاعات، حفظ جامعیت (integrity) است.
- بانک اطلاعات برخلاف سیستم های معمولی داده ها را در حصاری محکم به نام سیستم مدیریت بانک اطلاعات (DBMS) قرار می دهد و هر گونه تماس مستقیم کاربران با داده ها ممنوع است.

مدیریت تراکنش (مفاهیم)

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

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

- برنامه های کاربران به عنوان تراکنش به سیستم مدیریت بانک اطلاعات تحویل می شود. بخشی از یک DBMS که وظیفه مدیریت تراکنش ها را بر عهده دارد، واحد مدیریت تراکنش (transaction management) نام دارد.

مدیریت تراکنش (مفاهیم)

- این بخش از DBMS چهار کنترل موسوم به ACID را روی تراکنش ها اعمال و در نهایت این برنامه ها یا به خوبی اجرا شده و پایان می یابند که به این حالت انجام یا تثبیت commit می گویند و یا اینکه ساقط abort می شود.
- برای اینکه تراکنش جامعیت بانک اطلاعاتی را حفظ نماید، باید چهار خاصیت ACID را رعایت نماید.
- خواصی که صحت و جامعیت بانک اطلاعات را حفظ می نمایند عبارتند از:
- یکپارچگی Atomicity:

  به معنی "همه" یا "هیچ"

  یا تمامی دستورالعمل‌های یک تراکنش باید اجرا شود و یا هیچکدام از آنها.
- همخوانی Consistency:

  تراکنش باید تمامی قوانین جامعیت بانک اطلاعات را رعایت کند. هر تراکنش اگر به تنهایی اجرا شود بانک اطلاعات را از حالتی صحیح به حالت صحیح دیگری منتقل می کند.

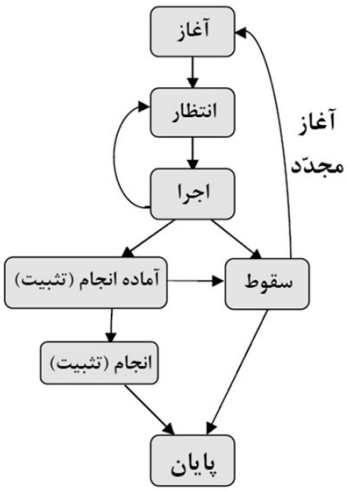
مدیریت تراکنش (مفاهیم)

- انزوا Isolation:

  اثر تراکنش‌های همروند روی یکدیگر چنان است که گویا هر کدام در انزوا هستند. (همروندی تراکنش‌ها کنترل می‌شود تا اثر مخرب روی هم نداشته باشند. این عمل توسط بخشی از DBMS به نام واحد کنترل همروندی (concurrency control) انجام می‌شود.
- پایایی Durability:

  تراکنش‌هایی که به مرحله انجام commit برسند، اثرشان ماندنی است و هرگز بطور تصادفی از بین نمی‌رود.
- دو عمل یکپارچگی و پایایی توسط واحدی از DBMS به نام واحد مدیریت ترمیم (recovery management) انجام می‌گیرد. وظیفه این واحد جلوگیری از تاثیر تراکنش های نیمه کاره بر روی بانک اطلاعات و از بین بردن آثار آنها است.

مراحل اجرای تراکنش



- مراحل انتظار و اجرا آنقدر تکرار می شوند تا تراکنش به طور موفق پایان پذیرد یا به مشکلی برخورد کرده ساقط شود.
- هر گاه تراکنش نیاز به منبعی داشت که در اختیار تراکنش های دیگر است باید منتظر بماند.
- اگر تراکنش مرتکب خطا شده باشد یا سر راه دیگر تراکنش ها قرار گرفته باشد سقوط (abort) رخ می دهد و بانک اطلاعات مربوطه به مرحله قبل از شروع آن تراکنش برمی گردد.
- اگر تراکنشی ساقط شود، می توان آن را مجدداً آغاز کرد و اگر تراکنشی آماده انجام شود، ممکن است باز هم سقوط کند.

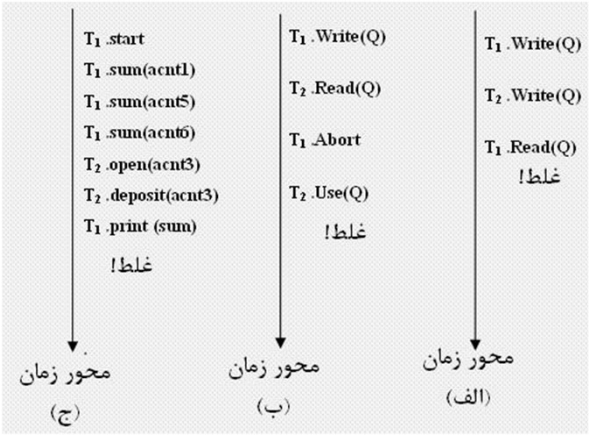
مراحل اجرای تراکنش

- ready- to-commit (آماده انجام) به معنی زمانی است که تراکنش همه کارهای مربوطه را انجام داده اما باید بخش های مختلفی را با هم هماهنگ کند. (در بانک های اطلاعاتی نامتمرکز اهمیت دارد)
- انجام (commit) زمانی که تمام بخش ها برای نهایی کردن کار توافق کرده اند. (تضمین عدم خدشه دار شدن تراکنش)
- اجرای همروند تراکنش ها در قالب ساختاری به نام (schedule) زمانبندی بررسی می شوند.
- در یک زمانبندی می توان تراکنشی را منتظر تراکنش دیگری گذاشت یا می توان اجرای یک تراکنش را به تعویق انداخت. اما در درون یک تراکنش نمی توان دستورها را جابه جا کرد به دلیل اینکه منطق آن تراکنش به هم می ریزد.

مزایای اجرای همروند تراکنش ها

- همواره با تراکنش های همروند یعنی تعدادی تراکنش که با یکدیگر کار می کنند سر و کار داریم. اگر بخواهیم تراکنش ها را یکی پس از دیگری اجرا کنیم، در این صورت هم گذردهی سیستم پایین می آید و هم میانگین زمان پاسخ دهی افزایش پیدا می کند.
- در حین اجرای یک تراکنش طولانی مدت شاید بتوان صدها تراکنش کوتاه را اجرا کرد و نباید آنها را منتظرگذاشت. بنابراین همروندی تراکنش ها، کارایی سیستم را بالا می برد.
- افزایش گذردهی (throughput): اجرای 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 اجرای همروند تراکنش ها



الف) تغییرات گم شده ب) دستیابی به داده نهایی نشده ج) بازیابی ناهمگام

مشکلات اجرای همروند تراکنش ها

❖ **الف) تغییرات گم شده (lost updates)**

تراکنش T_1 در نقطه ای از زمان داده Q را می نویسد (مثلا ۷۰). پس از آن تراکنش T_2 همان داده را تغییر می دهد (مثلا ۱۰۰). آنگاه تراکنش اولی مقدار Q را می خواند و انتظار همان مقداری را که نوشته دارد. تراکنش ها برنامه های مستقلی هستند و از تاثیر روی یکدیگر بی خبرند.

❖ **ب) دستیابی به داده نهایی نشده (uncommitted-data access یا dirty-read)**

تراکنش T_1در نقطه ای از زمان داده را تغییر می دهد. سپس تراکنش T_2 همان داده را می خواند ولی بعدا T_1 به دلیلی ساقط می شود و داده به حال اولیه بر می گردد. T_2 بی خبر از همه جا از آن داده استفاده می کند و به راه خود ادامه می دهد و دچار خطا می شود.

مشکلات اجرای همروند تراکنش ها

❖ **ج) بازیابی ناهمگام(inconsistent retrieval یا phantom problem)**

تراکنش T_1 روی داده هایی به ترتیب کار می کند و در پایان نتیجه ای را که به همه داده ها مربوط می شود اعلام می کند. در حالیکه پس از گذشتن از محدوده ای، داده جدیدی در آن محدوده توسط تراکنش T_2 ایجاد می گردد.

تراکنش T_1 جمع موجودی تعدادی حساب بانکی را محاسبه می کند که این حساب ها از حساب شماره ۱ تا شماره ۶ هستند. اما در این میان تراکنش دیگری، حساب جدیدی با نام شماره ۳ را باز می کند و مبلغی را به حساب آن می ریزد. جمع موجودی حساب های ۱ تا ۶ که توسط تراکنش ۱ محاسبه شده درست نیست.

پی در پی پذیری Serializability

- معروف ترین متد کنترل درستی اجرای همروند پی در پی پذیری است.
- یک تراکنش اگر خاصیت ACID داشته باشد نتیجه اش درست است و نیز اگر چند تراکنش داشته باشیم که پی در پی باشند، این زمانبندی که به آن زمانبندی پی در پی گفته می شود نیز نتیجه اش درست است.
- پی درپی پذیر یعنی معادل پی درپی؛ یعنی اگر زمانبندی های همروندی داشته باشیم که پی درپی نیستند اما معادل پی در پی هستند، در این صورت نیز نتیجه کار درست خواهد بود.
- دو روش اصلی پی درپی پذیری عبارتند از :
  - پی درپی پذیری در برخورد یا CSR (Conflict Serializability)
  - پی درپی پذیری در دید یا VSR (View Serializability)
- دستورات محاسباتی تأثیری در پی درپی پذیری ندارند، در زمانبندی ها فقط دستورات read و write در نظر گرفته خواهد شد.

پی در پی پذیری Serializability

- علائم استفاده شده در تراکنش ها عبارتند از:
- $r_i(Q)$: تراکنش i داده Q را می خواند (read)
- $w_i(Q)$: تراکنش i روی داده Q می نویسید (write)
- c_i: تراکنش i به مرحله تثبیت می رسد (commit)
- a_i: تراکنش i ساقط می شود (abort)
- اولین دستور تراکنش شروع آن و آخرین دستور آن نیز پایان آن است.

پی در پی پذیری Serializability

- اگر دستورات خواندن و نوشتن مربوط به تراکنش های مختلف به صورت هم روند اجرا شوند، بعضاً با هم برخورد دارند و بعضاً ندارند. برخورد داشتن یعنی تأثیر مخرب روی هم گذاشتن. دستوراتی را که با هم برخورد ندارند می توان همروند اجرا کرد و آنهایی را که برخورد دارند، باید به صورت پی درپی اجرا نمود.
- دستورات خواندن بین تراکنش های مختلف با هم مشکلی ندارند و چندین تراکنش می توانند همزمان یک داده را بخوانند.
- درصورتی که یکی یا هر دو تراکنش بخواهند روی یک داده بنویسند در این حالت ترکنش ها برخورد دارند.

تعریف برخورد

- چنانچه p_i و q_i به ترتیب عملگرهای (دستورات) تراکنشهای t_i و t_j باشند، گویند p_i و q_i با هم برخورد دارند اگر و تنها اگر:
  - این دو عملگر مربوط به تراکنش های متمایز باشند ($t_i \neq t_j$)
  - هر دو عملگر به یک داده دسترسی داشته باشند
  - حداقل یکی از این دو عملگر، عملگر نوشتن write باشد.

T_i / T_j	$r_i(Q)$	$w_i(Q)$
rj(Q)	بی برخورد	برخورددار
$w_j(Q)$	برخورددار	برخورددار

جدول برخورد بین دستورات

تعریف: زمانبندی پی در پی

❖ زمانبندی S را پی درپی گوییم اگر برای هر دو تراکنش، پایان یکی قبل از شروع دیگری باشد.

❖ مثال : یک زمانبندی پی درپی

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

T_1	r(A) w(A) a
T_2	r(A) w(B) c

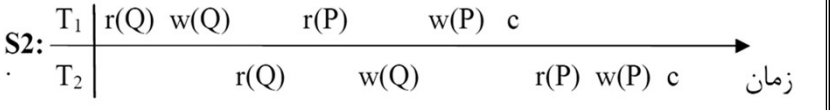
❖ ترتیب اجرای پی در پی آنها T_1 و سپس T_2 است:

❖ $S_1 : T_1 < T_2$

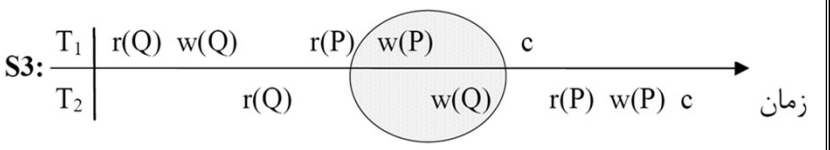
تعریف: زمانبندی معادل در برخورد (conflict equivalent)

❖ زمانبندی های S و 'S معادل در برخورد هستند، اگر هر دو روی یک مجموعه از دستورات و تراکنش ها کار کنند و با جابه جا کردن دستورات بدون برخورد در زمانبند S بتوانیم زمانبندی 'S را تولید کنیم.

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



❖ حل:



❖ دراین مثال فقط ترتیب زمانی $w_1(P)$ و $w_2(Q)$ عوض شده. هر دو دستور نوشتن هستند ولی برخورد ندارند چون روی دو داده مختلف عمل می کنند.

معادل در برخورد

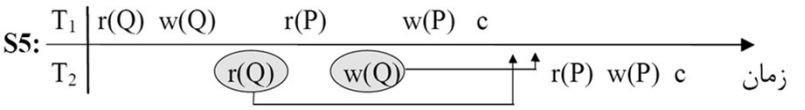
- نکته مهم: به هیچ عنوان نمی توان ترتیب دستورات یک تراکنش را عوض کرد چون منطق (semantic) آن تراکنش عوض می شود.
- مثال : " معادل در برخورد" با زمانبندی S_1

T_1	r(A) w(A) a	
T_2		w(B)r(A) c

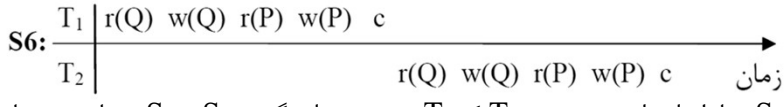
- غلط است زیرا ترتیب زمانی دستورهای دو یا چند تراکنش عوض نشده بلکه جای دستورها در درون یک تراکنش عوض شده است.

تعریف: پی در پی پذیر در برخورد

- زمانبندی S پی درپی پذیر در برخورد است اگر معادل در برخورد با یک زمانبندی پی درپی باشد. زمانبندی های همروند به شرطی که معادل در برخورد با یکی از ترتیبهای (زمانبندی های) پی درپی باشند، مشکل همروندی ندارند. مثال:



- دستور $w_2(Q)$ را می توان از $w_1(P)$ و c_1 گذراند زیرا با آنها برخورد ندارد. همچنین می توان $r_2(Q)$ را از $r_1(P)$ و $w_1(P)$ و c_1 گذراند. حاصل این عمل ها، زمانبندی S_6 است که پی در پی و معادل S_5 است.



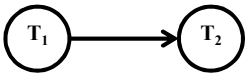
- S_6 معادل اجرای پی درپی $T_1 < T_2$ پس می توان گفت S_3 و S_5 زمانبندی های پی در پی پذیر در برخورد هستند.

تشخیص پی در پی پذیری دربرخورد(Conflict Serializability)

- راه‌حل گراف پی‌درپی پذیری برای تشخیص پی‌درپی پذیری ارائه شده‌است. این گراف مرتباً با تغییر مجموعه دستورات تراکنش‌ها در زمانبندی بهنگام می‌شود.
- گراف ها به طور مرتب تغییر می کنند؛ یعنی به محض اینکه یک دستور خواندن و یا نوشتن وارد سیستم می شود، ممکن است یک یال را به گراف اضافه کند و زمانی که تراکنشی ساقط و یا تثبیت شده و از سیستم خارج می شود، یالهایی و نیز گره ای از گراف حذف خواهد شد.
- نگهداری این گراف بسیار ساده است.
- در صورتیکه در گراف حلقه (cycle) مشاهده شود، تراکنش یا تراکنش ها ساقط می گردند. در غیر اینصورت به کار خود ادامه می دهند.
- هرگاه در گراف پی در پی پذیری حلقه (cycle) وجود داشته باشد؛ آنگاه زمانبندی آن پی در پی پذیر نیست و بالعکس.

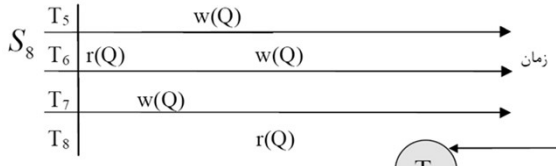
تشخیص پی در پی پذیری دربرخورد(Conflict Serializability)

- 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.
- دو روش وجود دارد:
  - هرگاه تراکنشی درخواستی برای خواندن یا نوشتن داد، بررسی شود اگر به آن اجازه داده شود آیا گراف دچار حلقه می شود؟ اگر دچار حلقه می شود اجازه داده نشود.
  - اجازه داده شود تراکنش ها به طور طبیعی کار خودشان را ادامه بدهند. فرض کنیم حلقه ای ایجاد نمی شود یا تعداد حلقه هایی که ایجاد می شود بسیار اندک است. اگر حلقه ایجاد شد ساقط شود و اگر حلقه ایجاد نشد تثبیت شود.
- گراف پی در پی پذیری در S_1

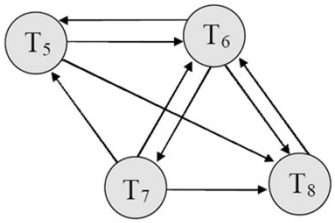


تشخیص پی در پی پذیری دربرخورد

❖ گراف پی درپی پذیری مربوط به زمانبندی S_8 زیر را رسم کنید و تشخیص دهید پی درپی پذیر است یا خیر؟ (توجه شود که تقدّم زمانی $r_8(Q)$ و $w_6(Q)$ مشخص نیست.)



❖ حل:



$T_6 \longrightarrow$ abort

❖ در این گراف تعدادی حلقه وجود دارد مانند (T_6,T_7,T_6) یا $(T_7,T_5,T_6,T_5,T_8,T_6)$. اگر تراکنش T_6 ساقط شود زمانبندی پی در پی پذیر $T_7<T_5<T_8$ به دست می آید که به این مساله پیدا کردن قربانی می گویند.

تمرین

❖ آیا زمانبندی زیر پی درپی پذیر در برخورد می باشد یا خیر؟ اگر جواب مثبت است معادل پی درپی آن چه خواهد بود؟

T_1	r(A) w(A)		r(B) w(B)	
T_2		r(A) w(A)		r(B) w(B)

❖ در زمانبند زیر، بعد از جابجایی چند دستور و کدامیک از دستورها می توان به اجرای پی در پی رسید؟

T_1	**r(A) w(A)**	**r(B) w(B) c**	
T_2		**r(A) w(A)**	**r(B) w(B) c**

تمرین

- کدامیک از زمانبندهای زیر پی در پی پذیر در برخورد هستند؟ (شماره تراکنش انجام دهنده دستور با اندیس نشان داده شده است)
- S_1: $R_1(x), R_3(x), W_1(x), R_2(x), W_3(x)$.
- S_2: $R_1(x), R_3(x), W_3(x) W_1(x), R_2(x)$.
- S_3: $R_3(x), R_2(x), W_3(x), R_1(x), W_1(x)$.

- با رسم گراف برای زمانبند داده شده مشخص کنید که آیا این زمانبند پی در پی پذیر در برخورد است یاخیر؟
- S_4: $R_1(x), R_2(z), R_1(z), R_3(x), W_1(x), W_3(y), R_2(y)$.

پی در پی پذیر در دید

- آیا روش های دیگری هم غیر از پی در پی پذیری در برخورد برای پی درپی پذیری وجود دارد؟ بلی
  - Commit-Serializability یا پی‌درپی پذیری در تثبیت
  - FSR (Final-state SeRializability) یا پی‌درپی‌پذیری درحالت نهایی
  - VSR (View Serializability)
- آیا روشی بهتر از پی در پی پذیری در برخورد وجود دارد؟ خیر.
- مزایا و معایب پی در پی پذیری در دید:
- مزایا:
- بازتر از روش پی درپی پذیری دربرخورد عمل می کند، یعنی زمانبندی را می پذیرد که پی درپی پذیری دربرخورد رد می کند.
- معایب:
- روش تشخیص آن زمانبر و طولانی می باشد.

پی در پی پذیر در دید

- 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 توسط یک تراکنش هم باید مقادیری که آن تراکنش می خواند یکسان باشند.
- تعریف:
- تراکنش T_j از تراکنش T_i می خواند اگر T_j داده ای را که آخرین بار T_i در آن نوشته است بخواند و T_i ساقط نشده باشد.

پی در پی پذیر در دید

- تعریف:
- زمانبندی های 's و s معادل در دید هستند اگر تراکنشها و مجموعه عملگرهای 'sو s یکسان باشد و سه شرط زیر برقرار باشد:
  - برای هر داده Q، اگر تراکنش T_i در s مقدار اولیه (مقدار قبل از اولین نوشتن) داده Q را می خواند، آنگاه در 's نیز مقدار اولیه Q را بخواند.
  - برای هر داده Q، اگر T_i در s داده Q را از T_j می خواند، در 's نیز داده Q را از T_j بخواند.
  - برای هر داده Q، آخرین تراکنشی از زمانبندی s که روی Q می نویسد، همان تراکنشی باشد که در زمانبندی 's آخرین بار روی Q می نویسد.
- با اعمال این شرط ها تضمین می شود:
- هر تراکنش در هر دو زمانبندی مقادیر یکسانی را بخواند.
- با تضمین یکسان بودن مقادیر نوشته شده روی هر داده اطمینان حاصل می شود که وضعیت نهایی بانک اطلاعات در هردو زمانبندی یکسان است.

تعریف: پی در پی پذیر در دید

❖ زمانبندی s پی در پی پذیر در دید است اگر معادل در دید با یک زمانبندی پی در پی باشد.

❖ مثال: آیا زمانبندی زیر پی درپی پذیر در برخورد می باشد؟ پی درپی پذیر در دید چه طور؟

$\underline{S_7}$

T_3	r(Q) w(Q) c
T_4	w(Q) c
T_5	w(Q) c

❖ حل: S_7 پی درپی پذیر در برخورد نیست ولی پی درپی پذیر در دید می باشد. ترتیب پی درپی آن $\underline{T_3<T_4<T_5}$ خواهد بود. CSR نیست اما VSR هست. برای تشخیص پی درپی پذیر بودن در دید، فقط یک داده به نام Q داریم که:

- ❖ در هر دو زمانبندی T_3 مقدار اولیه Q را می خواند.
- ❖ هیچ تراکنشی مقداری برای Q از تراکنش دیگر نمی خواند.
- ❖ در هر دو زمانبندی T_5 آخرین عمل نوشتن روی Q را انجام می دهد.

تعریف: پی در پی پذیر در دید

❖ هر زمانبندی که CSR نباشد اما VSR باشد نوشتن کورکورانه (blind write) انجام داده است یعنی بدون اینکه قبلاً داده مربوطه را خوانده باشد در آن می نویسد (در اینجا T_4,T_5)

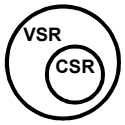
❖ هر زمانبندی پی در پی پذیر در برخورد حتماً پی در پی پذیر در دید است. عکس این موضوع الزاما برقرار نیست.

❖ تمرین: آیا زمانبندی زیر CSR است؟ VSR چطور؟

❖ S_8: $r_1(A)\ r_3(B)\ r_2(A)\ w_1(A)\ w_1(C)\ c_1\ w_2(C)\ w_2(D)\ c_2\ w_3(C)\ c_3$

تعریف: پی در پی پذیر در دید

- الگوریتم شناسایی پی در پی پذیری در دید از پیچیدگی بالایی برخوردار است. به همین دلیل بکارگیری روش CSR بسیار فراگیر شده است.
- VSR قدرتمندتر از CSR است و محدوده وسیع تری از زمانبندیهای درست را می پذیرد.
- تشخیص پی درپی پذیری در دید مسئله ای است که با پیچیدگی زمانی چند جمله‌ای قابل حل نیست (Np-complete) و تشخیص پی درپی پذیری در برخورد با پیچیدگی زمانی n^2 (n تعداد تراکنش ها) قابل قبول است.



محدوده عملکرد دو روش

تشخیص پی در پی پذیر در دید

- از نماد Read From (خواندن از) به صورت (T_j,x,T_i) استفاده می شود که تراکنش T_i داده x را از تراکنش T_j می خواند. مجموعه خواندن از زمانبندی s $RF(s)$ را برای همه داده های s تولید می کنیم با این فرض اولیه که قبل از شروع زمانبندی تراکنش T_0 تمام داده ها را نوشته است و پس از اتمام زمانبندی نیز تراکنش T_∞ همه داده ها را خواهد خواند.
- اگر 'S پی در پی باشد و $RF(S)=RF(S')$ باشد، آنگاه زمانبندی S پی در پی پذیر در دید است.
- آیا پی درپی پذیر بودن به معنی درستی یک زمانبندی است؟ یعنی جامعیت بانک اطلاعات را تضمین می کند؟خیر.
- جامعیت بانک اطلاعات دو شرط دارد:

۱ - از لحاظ کنترل همروندی مشکل نداشته باشد (پی درپی پذیری یکی از روش های کنترل همروندی است).

۲ - ترمیم پذیر

تشخیص پی در پی پذیر در دید

- آیا زمانبندی زیر پی در پی پذیر در دید است؟
- $S= r_2(x)\ w_2(x)\ r_1(x)\ r_1(y)\ r_2(y)\ w_2(y)\ c_1\ c_2$

- حل: مجموعه خواندن از زمانبندی های فوق به صورت زیر خواهد بود.
- $RF(S)=\{(T_0,x,T_2), (T_0,y,T_2), (T_2,x,T_1), (T_0,y,T_1),(T_2,x,T_:),(T_2,y,T_:)\}$
- اگر مجموعه خواندن را برای هر یک از ترتیب های پی در پی ممکن این تراکنش ها به دست آوریم خواهیم دید که هیچ یک معادل مجموعه $RF(S)$ نیستند:
- $RF(T_1<T_2)=\{(T_0,x,T_1), (T_0,y,T_1), (T_0,x,T_2), (T_0,y,T_2),(T_2,x,T_:),(T_2,y,T_:)\}$
- $RF(T_2<T_1)=\{(T_0,x,T_2), (T_0,y,T_2), (T_2,x,T_1), (T_2,y,T_1),(T_2,x,T_:),(T_2,y,T_:)\}$
- زمانبندی S پی در پی پذیر در دید نیست.
- تمرین: آیا زمانبندی زیر پی در پی پذیر در دید است؟
- $S= r_1(x), r_2(y), w_1(y), r_2(x), w_2(x)$

ترمیم پذیری

- در سناریو هایی نظیر دسترسی به داده تثبیت نشده (abort) تکلیف تراکنشی که با داده غیر معتبر درگیر شده است چیست؟ تا اینجا امکان اینکه تراکنشی دچار خرابی (failure) شود نادیده گرفته شده بود.
- برای درستی فرآیند همزمانی، یک زمانبندی علاوه بر پی درپی پذیری باید ترمیم پذیر نیز باشد.
- فرض کنید در زمانبندی S_9 تراکنش T_9 به محض انجام دستور خواندن، تثبیت شود. از طرفی اگر T_8 پس از تثبیت شدن T_9، مجبور به سقوط شود، T_9 به داده های تثبیت نشده دسترسی داشته است که صحت بانک اطلاعاتی را خدشه دار می نماید. بنابراین زمانبندی برای درست بودن علاوه بر پی درپی پذیری باید ترمیم پذیر نیز باشد.

S_9

T_8	r(A) w(A) r(B) a
T_9	r(A) c

- زمانبندی را ترمیم پذیر (Recoverable-RC) گویند اگر برای تمام T_j هاکه از T_i ها می خوانند، تثبیت تراکنش T_i قبل از تثبیت تراکنش T_j صورت گیرد.

ترمیم پذیری

- اشکالات:
- زمانبندی ترمیم پذیر ممکن است سبب ساقط شدن تراکنش های دیگر به صورت آبشاری (cascading aborts) گردد. این مشکل می تواند سبب از دست رفتن حجم قابل توجهی از کارهایی که تا به حال انجام شده اند گردد.
- مثال: در زمانبندی S_{10} چنانچه T_{10} دچار خرابی شود، T_{11} و T_{12} نیز مجبور به سقوط خواهند شد.

S_{10}

T_{10}	r(A) r(B) w(A)
T_{11}	r(A) w(A)
T_{12}	r(A)

- زمانبندی فاقد سقوط های آبشاری (Avoiding Cascading Aborts-ACA) است چنانچه برای هر دو تراکنش T_iو T_j، اگر تراکنش T_j از T_i بخواند، آنگاه تراکنش T_i قبل از خواندن تراکنش T_j تثبیت گردد.

ترمیم پذیری

- زمانبندی فاقد سقوط های آبشاری بهتر از زمانبندی های ترمیم پذیر هستند.
- هر زمانبندی فاقد سقوط های آبشاری (ACA) ترمیم پذیر (RC) هم هست.
- مشکل سقوط های آبشاری زمانبندی S_{10} در زمانبندی S_{11} رفع شده است.

S_{11}

T_{10}	r(A) r(B) w(A) c
T_{11}	r(A) w(A) c
T_{12}	r(A)

- زمانبندی را محض (سختگیر) (Strict-ST) گویند اگر برای هر دو تراکنش T_iو T_j، اگر تراکنش T_j داده ای را پس از نوشتن T_i بخواند یا بنویسد، این عمل T_j بعد از خاتمه (تثبیت یا سقوط) T_i اجرا شود.
- از لحاظ خواندن معادل فاقد سقوط آبشاری است ولی مفهوم نوشتن را هم در نظر می گیرد. ACA فقط به خواندن می پردازد. به عبارت دیگر زمانبندی سخت گیر، اجازه خواندن یا نوشتن هیچ داده ای را نمی دهد مگر آنکه تراکنشی که روی آن داده نوشته است، خاتمه یافته باشد (انجام یا ساقط شده باشد).

نیاز به پروتکل کنترل همروندی و ترمیم

- زمانبندی محض (سختگیر) به طور همزمان خاصیت پی درپی پذیری و ترمیم پذیری را دارد بنابراین جامعیت بانک اطلاعات را یکجا تضمین می کند.
- نکته: در کنترل هایی که انجام می شود، زمانبندی ها اگر غلط بودند abort یا ساقط می شدند و این کار درست نیست. ساقط کردن تراکنش ها باعث می شود مقدار زیادی از کارهایی که انجام دادند از بین برود.
- آیا می توان از ساقط کردن جلوگیری کرد؟
- بله. می توان پروتکل هایی را برای کنترل همروندی و ترمیم پذیری تراکنش ها تعریف کرد که اگر زمانبندی ها با اجرای این پروتکل ها به پیش بروند، همواره درست باشند و صحت و جامعیت بانک اطلاعات را به مخاطره نیاندازند و جز در موارد استثنایی نیازی به ساقط کردن تراکنش ها نیست.
- بنابراین پروتکل هایی در آینده بیان می شوند.

پروتکل های کنترل همروندی

- **رویکردهای زمانبندی تراکنش ها**
- خوش بینانه (optimistic)
  - اجرای آزادانه تراکنش ها- بررسی صحت عملکرد- نوشتن روی رسانه و انعکاس تغییرات در بانک اطلاعات
- بدبینانه (pessimistic)
  - اجازه اجرا به شرط حفظ صحت و جامعیت بانک اطلاعات-نوشتن روی رسانه
- **زمانبند در مواجه با هر عملگر یک انتخاب از سه انتخاب زیر را دارد:**

1. اجرای عملگر (execute)
2. به تأخیر اندازی اجراء عملگر (قرار دهی آن در صف) (delay)
3. عدم پذیرش عملگر که منجر به ساقط شدن تراکنش می گردد (reject)

پروتکل های کنترل همروندی

- **انواع زمانبند ها**
- زمانبند محافظه کار (conservative)
- اگر لازم باشد اجرای دستورات تراکنش را به تعویق می‌اندازد و محتاطانه و محافظه کارانه اجرای دستورات را پی می‌گیرد تا حتی الامکان تراکنش‌ها ساقط نشوند.
- زمانبند بی پروا (aggressive)
- هدف پرهیز از تأخیر در اجرای دستورات است. در این زمانبندی دستورات فوراً اجرا می شوند و درصورت بروز مشکلی مجبور به ساقط کردن برخی تراکنش ها می گردد.
- **پروتکل های کنترل همروندی**
- پروتکل های مبتنی بر قفل (lock-based)
- پروتکل های مبتنی بر گراف (graph-based)
- پروتکل های مبتنی بر مهر زمانی (timestamp-based)

پروتکل های مبتنی بر قفل (lock-based)

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

پروتکل های مبتنی بر قفل (lock-based)

- **سازگاری قفل ها**
- 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-eXecuted)
- 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.

پروتکل های مبتنی بر قفل (lock-based)

- 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.

$i \neq j$	s_j	x_j
s_i	سازگار	ناسازگار
x_i	ناسازگار	ناسازگار

پروتکل های مبتنی بر قفل (lock-based)

❖ **نکات قفل گذاری**

❖ هر تراکنش پیش از هر دستور 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 (wait) 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 روی آن داده بزند.

❖ درخواست آزاد شدن قفل روی داده Q با $u_i(Q)$ نمایش داده می شود.

❖ 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.

پروتکل های مبتنی بر قفل (lock-based)

❖ **نکات قفل گذاری**

❖ قفل گذاری داده ها می تواند با دانه بندی های (granularity) مختلف از نظر اندازه داده صورت پذیرد. قفل روی جدول، سطر یا بخشی از جدول.

❖ قفل‌های با دانه‌بندی ریزتر (fine grain) باعث افزایش درجه همروندی شده اما سربار نیز افزایش می‌یابد و قفل‌های با دانه‌بندی درشت‌تر (coarse grain) منجر به کاهش همروندی و نیز سربار می گردد.

❖ سیستم مدیریت بانک اطلاعات بسیاری از فعالیت ها از جمله قفل گذاری را به سیستم عامل می دهد. معمولا واحد قفل گذاری صفحه (page) است.

❖ از جدول قفل (lock table) برای پیاده سازی واحد مدیر قفل استفاده می شود. این جدول در هر لحظه وضعیت استفاده تراکنش ها از داده ها، نوع قفل ها، اشتراک داده ها و ... را در بر دارد. جدول قفل مشخص می کند هر داده ای توسط چه تراکنش هایی قفل زده شده یا باز می شود.

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

پروتکل های مبتنی بر قفل (lock-based)

❖ مثال: معادل زمانبندی S_1 را با استفاده از قفل‌های 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) a u(A)
T_2	x(A) w(A) x(B) w(B) u(A) u(B) c

پروتکل های مبتنی بر قفل (lock-based)

❖ مثال: معادل زمانبندی S_3 را با استفاده از قفل‌های اشتراکی و انحصاری بنویسید.

S_3

T_3		w(Q)	
T_4	r(Q)		w(Q)
T_5		w(Q)	
T_6			r(Q)

❖ $S_4: s_4(Q)\ r_4(Q)\ x_5(Q)\ x_3(Q)\ x_4(Q)\ w_4(Q)\ u_4(Q)\ a_4$ اجابت $w_5(Q)\ u_5(Q)$

c_5 اجابت $w_3(Q)\ c_3\ u_3(Q)\ s_6(Q)\ r_6(Q)\ u_6(Q)\ c_6$

پروتکل های مبتنی بر قفل (lock-based)

❖ **چند نکته**

❖ تثبیت یا سقوط تراکنش ها به عواملی بستگی دارد که بعضا ارتباطی با قفل گذاری ندارد و از کنترل ما خارج است (باز نشدن فایل درخواستی).

❖ در دو مثال قبل برای باز کردن قفل ها محدودیتی اعمال نگردید که در این حالت ممکن است مشکل بازیابی ناهمگام رخ دهد.

❖ یک تراکنش باید در صورت لزوم قفل خود را تبدیل کند.

❖ اگر تراکنشی قفل s(Q) دارد و بعداً عمل w(Q) می خواهد انجام دهد باید تقاضای x(Q) بدهد. (این درخواست ممکن اجابت نشود اگر تراکنش های دیگر هم Q را قفل اشتراکی کرده باشند)(افزایش درجه قفل upgrade)

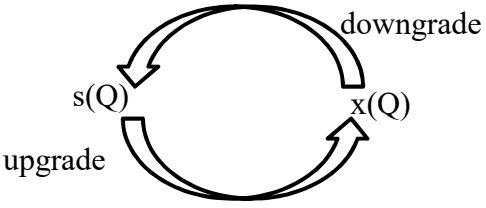
پروتکل های مبتنی بر قفل (lock-based)

❖ **چند نکته**

❖ قفل x(Q) برای خواندن بعدی باید به قفل s(Q) تبدیل شود تا دیگران هم بتوانند از این داده استفاده کنند. (کاهش درجه قفل downgrade)

❖ کاهش درجه قفل باعث افزایش درجه همروندی می شود.

❖ اگر قفل x را آزاد کنیم و بعد تقاضای قفل s بدهیم ممکن است تراکنش دیگری آن داده را از نوع x قفل کند و نتوانیم قفل مجدد از نوع s بزنیم.



بن بست (deadlock)– قحطی (starvation)

❖ زمانبندی زیر را در نظر بگیرید:

T_7	x(B)w(B)	x(A) → wait
T_8	s(A)r(A)s(B) →wait	

❖ هر یک از تراکنش های T_7 و T_8 منتظر دیگری هستند تا منبع مورد نیازشان را آزاد کند. (درخواست قفل ناسازگار)

❖ یک انتظار چرخشی میان تراکنشها بوجود آمده است که هیچکدام نمی توانند کار خود را انجام دهند.

❖ به این انتظار چرخشی بن بست گفته می شود.

❖ 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 تراکنشها بماند که همگی می خواهند قفل s روی همان داده بزنند و تراکنش مورد نظر تا زمان نامعلومی باید منتظر بماند. تراکنش درخواست دهنده قفل x دچار قحطی می شود.

پروتکل های قفل دو مرحله ای (2PL(Two-Phase Locking

❖ 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:

S_5	T_9	x(A) dec(A,amnt) w(A) u(A)
	T_{10}	s(A) r(A) s(B) r(B)
ادامه	T_9	x(B) inc(B,amnt) w(B) u(B)
	T_{10}	disp(A+B) u(A) u(B)

❖ با وجود رعایت همه قوانین قفل گذاری ولی حاصل نهایی T_{10} صحیح نیست.

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

پروتکل های قفل دو مرحله ای (2PL(Two-Phase Locking

- 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.
- تراکنش‌ها اجازه ندارند به محض اتمام کارشان با یک داده قفل آن را باز کنند.
- پروتکل قفل دو مرحله ای 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:
- مرحله اول (مرحله رشد- growing)
- در این مرحله تراکنش فقط می تواند قفل بگیرد و با داده ها کار کند ولی نمی تواند قفلی را آزاد کند.
- مرحله دوم (مرحله نقصان یا عقب نشینی shrinking)
- در این مرحله تراکنش فقط می تواند قفل آزاد کند و با داده ها کار کند ولی نمی تواند قفل جدیدی را روی داده ای بگذارد.

پروتکل B2PL (Basic Two-Phase Locking)

- در پروتکل قفل دو مرحله ای پایه 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.
- به محض 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، انجام عملیات، اتمام تراکنش وجود ندارد و به مشکل بن بست هم نمی پردازیم.
- اجرای 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.

پروتکل B2PL (Basic Two-Phase Locking)

❖ مثال: معادل زمانبندی S_5 با رعایت قفل دو مرحله ای پایه به صورت زیر است:

S_6	T_9	x(A) dec(A,amnt) w(A) x(B) inc(B,amnt) u(A) w(B) u(B)c
	T_{10}	s(A) r(A)
ادامه	T_9	
	T_{10}	s(B) r(B) disp(A+B) u(A) u(B) c

❖ تراکنش T_9 قفل داده A را باز نمی کند تا قفل B به او داده شود. بنابراین کار T_{10} تا رها شدن B به تعویق می افتد.

❖ با این روش دو تراکنش به صورت پی در پی پذیر کار کرده اند و نتیجه نیز صحیح است.

❖ در پروتکل B2PL امکان وقوع بن بست وجود دارد. برای رفع این مشکل پروتکل C2PL معرفی شده است.

پروتکل C2PL (Conservative Two-Phase Locking)

❖ پروتکل قفل دو مرحله ای محافظه کارانه پیش از شروع اجراء اولین دستور تراکنش باید همه قفلهای مورد نیاز آن تراکنش گرفته شده باشد.

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

❖ مهمترین مزیت: تضمین می کند بن بست رخ نخواهد داد. زیرا حالتی پیش نمی آید که درخواست قفل داده ای که در اختیار دیگری است را داشته باشد درحالیکه خودش برخی از قفل ها را در اختیار دارد.

❖ مشکلات اصلی در این پروتکل: کاهش سطح همروندی است و نیاز به دانستن مجموعه قفل های مورد نیاز هر تراکنش پیش از شروع اجرا آن است.

❖ در دنیای واقعی احتمال بروز بن بست خیلی زیاد نیست و در اینجا به این بهانه کارایی سیستم بسیار پایین می آید.

پروتکل C2PL (Conservative Two-Phase Locking)

❖ مثال: معادل C2PL زمانبندی S_5 به صورت زیر است:

S_7	T_9	x(A) x(B)dec(A,amnt) w(A) inc(B,amnt) u(A) w(B) u(B)c
	T_{10}	s(A) s(B)

ادامه	T_9	
	T_{10}	r(A) u(A) r(B) disp(A+B) u(B) c

پروتکل S2PL (Strict Two-Phase Locking)

❖ 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 ها abort یا commit به تعویق می افتد. (برای تضمین عدم وقوع سقوط های آبشاری)

❖ قفل های s می توانند کمی زودتر (بعد از آخرین دستور تراکنش و قبل از abort یا commit) باز شوند.

❖ در سایر موارد عملکرد این پروتکل با B2PL یکسان است.

پروتکل S2PL (Strict Two-Phase Locking)

- این پروتکل سخت گیرانه عمل می کند و ممکن است که بسیاری از زمانبندی های درست را نپذیرد.
- به عنوان یکی از بهترین گزینه ها در اکثر سیستم های مدیریت بانک اطلاعات مورد استفاده قرار گرفته 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 پیام های بازکردن قفل ندارد.
- پروتکل S2PL تمام شرا یط صحت و جامعیت بانک اطلاعات را یکجا دارد.
- جلوگیری از ساقط شدن تراکنش ها، جلوگیری از بازیابی نابهنگام، پی در پی پذیری، ترمیم پذیری

پروتکل S2PL (Strict Two-Phase Locking)

- مثال: معادل S2PL زمانبندی S_5 به صورت زیر است:

S_8	T_9	x(A) dec(A,amnt) w(A) x(B) inc(B,amnt) w(B) c u(A) u(B)
	T_{10}	s(A)

ادامه	T_9	
	T_{10}	r(A) s(B) r(B) disp(A+B) c u(A) u(B)

- در برخی پروتکل ها به نام SS2PL (Strong Strict Two-Phase Locking) یا R2PL (Rigorous Two-Phase Locking) 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.

پروتکل SC2PL (Strict Two-Phase Locking)

- این پروتکل مزایای هر دو پروتکل C2PLو S2PL را در کنار هم داراست.
- از قوانین C2PL برای قفل کردن داده ها و از قوانین S2PL برای آزاد سازی قفل ها استفاده می شود.
- مثال: معادل SC2PL زمانبندی S_5 به صورت زیر است:

S_9	T_9	x(A) x(B) dec(A,amnt) w(A) inc(B,amnt) w(B) c u(A) u(B)
	T_{10}	s(A)

ادامه	T_9	
	T_{10}	s(B) r(A) r(B) disp(A+B) c u(A) u(B)

پروتکل های مبتنی بر قفل (lock-based)

- معادل زمانبندی زیر را یک بار با قفل باینری و یکبار با قفل s/x و رعایت پروتکل B2PL بنویسید.

S_{10}

T_1	r(Q)		w(Q)	
T_2		r(A)		r(Q)
T_3		r(Q)	r(P)	w(P)
T_4			w(Q)	w(A)

- روش باینری:
- $l_1(Q)\ r_1(Q)\ l_2(A)\ r_2(A)\ l_3(Q)\ l_4(Q)\ w_1(Q)\ u_1(Q)\ c_1$ اجابت $r_3(Q)\ l_3(P)$

$r_3(P)\ l_2(Q)\ w_3(P)\ u_3(Q)\ u_3(P)\ c_3$ اجابت $w_4(Q)\ l_4(A)$ بن بست

انتظار ← $r_3(P)\ l_2(Q)$

انتظار ← $w_4(Q)\ l_4(A)$

پروتکل های مبتنی بر قفل (lock-based)

❖ روش s/x:

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

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

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

❖ تراکنش های T_2 و T_3 و نیز T_1 و T_3 دستورهای همزمان دارند که می توان هر یک را ابتدا انجام داد. در این مثال تغییر ترتیب این دستورها تاثیر عمده ای در نتیجه ندارد اما به طور کلی می تواند تاثیر داشته باشد و بن بست رخ دهد.

پروتکل های مبتنی بر قفل (lock-based)

❖ تمرین: معادل زمانبندی های زیر را با پروتکل S2PL بنویسید.

S_1 r1(A),r2(B),r1(B),w1(B),r3(C),w2(A)

S_2

T1	R(A)		w(A)	
T2		R(B)		R(A)
T3		R(A)	R(C)	w(C)
T4		W(A)		W(B)

پروتکل های مبتنی بر قفل (lock-based)

❖ تمرین: زمانبندی زیر را در نظر بگیرید.

S_3

T1	r(A)	w(B)	
T2	r(B)		w(C)
T3		r(C)	w(A)

❖ معادل C2PL آنرا بنویسید.

❖ نشان دهید که در B2PL دچار بن بست می شود یا خیر.

پروتکل های مبتنی بر گراف (graph-based)

❖ پروتکل های مبتنی بر گراف خیلی مرسوم نیستند. در این پروتکل ها مجموعه تمامی داده های مورد نظر $D=\{d_1,d_2,\ldots,d_n\}$ به صورت یک گراف جهت دار بدون حلقه (DAG) در می آیند. به این گراف، گراف بانک اطلاعات گفته می شود.

❖ در این گراف اگر 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، آن وقت هر تراکنش که می خواهد از داده d_j استفاده نماید، باید قبل از d_j داده d_i را نیز قفل کند(یعنی کل مسیر مورد نیاز را قفل نماید).

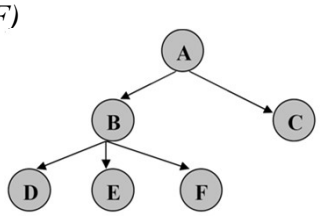
❖ نوع خاصی از این پروتکل ها که پرکاربردتر است، پروتکل درختی (tree protocol) است که در بانک اطلاعات سلسله مراتبی کاربرد داشته است. در این پروتکل ها از قفل باینری یا s/x استفاده می شود.

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

پروتکل های مبتنی بر گراف (graph-based)

- هر تراکنش اولین قفلش را روی هر داده ای می تواند بزند اما از آن پس داده ای مثل Q فقط به شرطی می تواند قفل شود که پدر داده Q توسط همان تراکنش قفل شده باشد.
- آزاد کردن قفل ها در هر زمانی مجاز است که امتیاز بزرگی محسوب می شود.
- برای هر تراکنش در شروع گرفتن قفل داشتن قفل برای پدر ضروری نیست.
- در درخت زیر زمانبندی S_{11} قفل های باینری را به صورت زمانبندی S_{12} اخذ می نماید.
- $S_{11} : w_1(E), w_2(D), w_1(C), w_2(E), w_2(F)$

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



پروتکل های مبتنی بر گراف (graph-based)

- پروتکل های درختی، پی در پی پذیری در برخورد و فاقد بن بست بودن را تضمین می کنند.
- یکی از مشکلات این دسته پروتکل ها این است که بسیاری از داده هایی را قفل می کنند که نیاز به دسترسی به آنها نیست. البته زمانی کاربرد دارند که داده ها ارتباط درختی با یکدیگر داشته باشند که معمولاً در بانک اطلاعات معمولی، داده ها ارتباط درختی با یکدیگر ندارند. علاوه بر افزایش سربار اضافی، سطح همروندی را نیز کاهش می دهد. مشکل سقوط آبشاری به قوت خود باقی است.
- پروتکل درختی نسبت به قفل دو مرحله ای قفلها را زودتر آزاد می نمایند که تا حدودی همروندی را بهبود می بخشد.
- مجموعه زمانبندی های قابل قبول پروتکل درختی با مجموعه زمانبندی های قابل قبول پروتکل قفل دومرحله ای دقیقاً یکسان نیست، یعنی زمانبندی هایی وجود دارند که در پروتکل های درختی معتبرند ولی در قفل دو مرحله ای قابل قبول نمی باشند و بالعکس.

پروتکل های مبتنی بر مهر زمانی (timestamp-based)

- 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 T_i با $TS(T_i)$ نمایش داده می شود. برای دو تراکنش T_iو T_j در صورتی که T_j دیرتر وارد سیستم شده باشد، $TS(T_i) < TS(T_j)$ 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 سیستم جهت تخصیص مهر زمانی استفاده می شود که Timestamp تصاعدی خواهد بود. در سیستم های نامتمرکز می توان از یک سایت یا زوج مرتب مهر زمانی محلی و ID سایت استفاده کرد.

پروتکل های مبتنی بر مهر زمانی (timestamp-based)

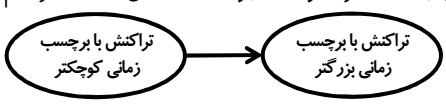
- برای هر داده Q مهر زمانی خواندن و نوشتن به این صورت تعریف می شود:
- W-TS(Q): مهر زمانی نوشتن داده Q، برابر است با بزرگترین مهر زمانی تراکنشی (که به صورت موفقیت آمیز) روی Qنوشته است.
- R-TS(Q): مهر زمانی خواندن داده Q، برابر است با بزرگترین مهر زمانی تراکنشی (که به صورت موفقیت آمیز) Q را خوانده است.
- نکته: منظور از بزرگترین، آخرین تراکنش و یا جدید ترین تراکنش می باشد.
- به طور موفقیت آمیز به این معنی است که تراکنش ها commit شده اند.

پروتکل های مبتنی بر مهر زمانی (timestamp-based)

- 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 T_i شامل یک دستور read(Q) است.
- اگر $TS(T_i) < W\text{-}TS(Q)$ باشد آنگاه تراکنش T_i 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 $TS(T_i) >= W\text{-}TS(Q)$ باشد، آنگاه دستور خواندن تراکنش T_i اجرا می شود و مهر زمانی خواندن Q، با ماکزیمم مهر زمانی تراکنش T_i و مهر زمانی خواندن Q مقدار دهی می شود.
- تذکر: ردشدن (reject)، به معنای در انتظار ماندن یا سقوط و شروع مجدد می باشد.

پروتکل های مبتنی بر مهر زمانی (timestamp-based)

- ۲. قواعد نوشتن:
- فرض کنید تراکنش T_i شامل یک دستور write(Q) باشد.
- اگر $TS(T_i) < R\text{-}TS(Q)$ یا $TS(T_i) < W\text{-}TS(Q)$ باشد آنگاه با دستور نوشتن تراکنش موافقت نمی شود و تراکنش T_i، رد می شود.
- در غیر این صورت نوشتن اجرا می شود و مهر زمانی نوشتن Q، با $TS(T_i)$ مقدار دهی می شود.
- پروتکل مبتنی بر مهر زمانی مطمئناً فاقد حلقه بوده و زمانبندی آن پی‌درپی‌پذیر است.
- اگر هیچ تراکنشی به حالت انتظار نرود (یعنی رد شدن تراکنش به معنی شروع مجدد باشد) این پروتکل ها فاقد بن بست نیز خواهند بود اما ممکن است ترمیم پذیر نباشندکه راه حل هایی برای برقراری این شرط نیز وجود دارد.



پروتکل های مبتنی بر مهر زمانی (timestamp-based)

❖ زمانبندی زیر را با پروتکل S2PL و مهر زمانی بنویسید:

r1(A),r2(B),r1(B),w1(B),r3(C),w2(A)

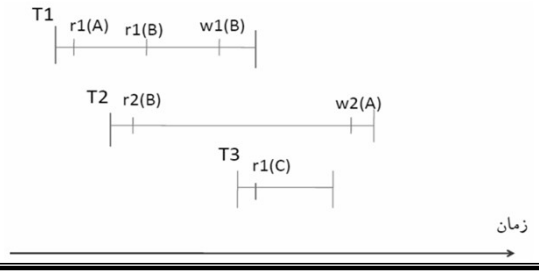
❖ S2PL: $s_1(A), r_1(A), s_2(B), r_2(B), s_1(B), r_1(B), x_1(B), s_3(C), r_3(C), c_3, u_3(C), x_2(A)$

انتظار

بن بست

انتظار

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



پروتکل های مبتنی بر مهر زمانی (timestamp-based)

❖ زمانبندی زیر را با پروتکل مهر زمانی بنویسید:

T_1	r(D)	w(D)
T_2	r(D)	w(D)

عمل نوشتن توسط T_1در لحظه t4 رد می شود

چرا که $TS(T_1)= 150 < W\text{-}TS(D) = 160$

$TS(T_1)=150$

$TS(T_2)=160$

$R\text{-}TS(D)=0$

$W\text{-}TS(D)=0$

زمان	T_1	T_2	D
	$TS(T_1)=150$	$TS(T_2)=160$	R-TS(D)=0 W-TS(D)=0
t1	r(D)		R-TS(D)=150
t2		r(D)	R-TS(D)=160
t3		w(D)	W-TS(D)=160
t4	w(D)		

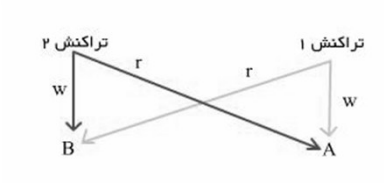
Rules:

Read: IF TS(Ti)<W-TS(Q) Then Reject Else Accept

Write: IF (TS(Ti)<W-TS(Q)) Or (TS(Ti)<R-TS(Q)) Then Reject Else Accept

روش های مدیریت بن بست

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



❖ 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.

روش های مدیریت بن بست

❖ 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. بن بست را می شگند، روشی ساده است ولی امکان بروز قحطی وجود دارد.

❖ در این روش قبل از شروع تراکنش ها به آنها یک فرصت یا مهلت زمانی اختصاص داده می شود.

❖ مزایا:

❖ باعث می شود اگر تراکنش ها در بن بست قرار گیرند ساقط شوندو بن بست ها به طور خودکار شکسته شود.

❖ افراد و کاربران مختلف نمی توانند وقت سیستم را به طور نامحدود بگیرند.

❖ معایب:

❖ به سادگی نمی توان میزان فرصت یا timeout را تعیین کرد.

❖ امکان بروز قحطی

روش های مدیریت بن بست

- 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. یکی از روشهای زیر را می توان اعمال کرد:

روش های مدیریت بن بست

- روش wait-die: تراکنش پیرتر(با مهرزمانی‌کمتر) منتظر تراکنش جوانتر می‌ماند تا قفل مربوطه را آزاد کند. در مقابل تراکنش جوانتر هرگز منتظر نمی‌ماند، بلکه ساقط می شود (می میرد). ممکن است یک تراکنش چندین بار بمیرد.
- روش wound-wait: تراکنش پیرتر به جای انتظار، تراکنش جوانتر را می کشد (قبضه‌ای یا preemptive نام دارد). تراکنش جوانتر منتظر تراکنش پیرتر می ماند. در این روش تراکنش پیرتر 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.
- در هر دو روش فوق، تراکنشی که ساقط می شود با همان بر چسب زمانی مخصوص بن بست شروع به کار مجدد می کنند که همین سبب می‌شود تراکنش جوان پس از مدتی تبدیل به یک تراکنش پیر می شود.

روش های مدیریت بن بست

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

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

روش های مدیریت بن بست

❖ تشخیص و رفع بن بست

❖ برای تشخیص بن بست گرافی به نام گراف انتظار Wait-For Graph وجود دارد که گره های آن تراکنش ها هستند و لبه $T_i \rightarrow T_j$ در صورتی وجود دارد که تراکنش T_i منتظر T_j باشد. چنانچه در این گراف حلقه ای وجود داشته باشد، در این صورت بن بست به وجود آمده است.

❖ گراف انتظار را رسم کرده و هر عملی که انجام شد با توجه به لبه های به وجود آمده گراف به روز می شود و در زمانهای مختلف (چند ثانیه یا دقیقه ای یا هر چند دقیقه یکبار) بررسی می شود که آیا در گراف دور (cycle) وجود دارد یا خیر. اگر حلقه وجود داشت سعی می شود با ساقط کردن (abort) یک یا چند تراکنش، حلقه را از بین برد.

❖ معمولا گره‌ای به عنوان گره قربانی استفاده می شود که ساقط کردن آن، کمترین هزینه را داشته باشد.

روش های مدیریت بن بست

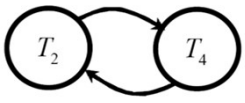
- تراکنشی که باید ساقط شوند قربانی victim گفته می شود:
- مهمترین معیارهای انتخاب قربانی عبارتند از:
  - تعداد حلقه هایی از گراف انتظار که این تراکنش در آنها شرکت دارد حداکثر باشد و با ساقط شدن آن، این حلقه ها نیز شکسته خواهد شد.
  - حجم کاری که تا کنون در آن تراکنش صورت گرفته است کمتر باشد.
  - تعداد به روزرسانی هایی که تراکنش انجام داده کمتر باشد.
  - حجم کارباقی مانده تراکنش بیشتر باشد.
- 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 ساقط می شود.

روش های مدیریت بن بست

- مثال: گراف انتظار زمانبندی S_{10} با قفل باینری به شکل زیر می باشد:

S_{10}

T_1	r(Q)		w(Q)	
T_2		r(A)		r(Q)
T_3		r(Q)	r(P)	w(P)
T_4			w(Q)	w(A)



- یکی از تراکنش ها به عنوان قربانی ساقط می شود.

سطوح جامعیت

- آنچه تا به اینجا به عنوان شرط درستی زمانبندی ها، به نام پی در پی پذیری و ترمیم پذیری بررسی شد، بالا ترین سطح از اطمینان در مورد صحت و جامعیت بانک اطلاعات را برای ما به ارمغان می آورد.
- سطوح پایین تری از جامعیت نیز وجود دارند.
- در مواردی ممکن است دقت و صحت نتیجه تراکنش برای ما چندان اهمیت نداشته باشد که سرعت اجرا (درجه همروندی بالاتر): پرس و جو در مورد تعداد تاپل های جدول
- 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:
- بالاترین سطح جامعیت و آنچه تا به حال معرفی شد که پیش فرض نیز همین سطح بوده است.

سطوح جامعیت

- سطح خواندن قابل تکرار (repeatable read):
- فقط داده های نهایی شده را می توان خواند. در این سطح اگر یک داده را چندین بار هم بخوانیم، همیشه مقدار یکسانی برای آن دریافت خواهیم کرد.
- سطح خواندن قابل تثبیت شده (read committed):
- فقط داده های نهایی شده را می‌توان خواند اما تضمینی نیست که با خواندن های بعدی هم همین مقدار را بخوانیم.
- سطح خواندن تثبیت نشده (read uncommitted):
- اجازه خواندن داده های نهایی نشده را نیز می دهد.

مدیریت ترمیم (recovery management)

- در مدیریت تراکنشها، این امکان وجود دارد که در هر لحظه از اجرای تراکنش، یک خرابی (failure) اتفاق بیفتد و تراکنش نتواند ادامه یابد. به عبارتی برخی از دستورات آن اجرا شده و بقیه به دلیل وقوع خرابی متوقف شوند. همچنین زمانی که تراکنشی انجام (commit) می شود، بعداً اتفاقی بیفتد که تأثیر آن را از بین ببرد. این دو نوع اتفاق به عنوان failureها مطرح هستند و باید آنها را ترمیم کرد؛ یعنی باید کاری کردکه اگر چنین اتفاقاتی افتاد، تأثیر مخربی روی بانک اطلاعات نگذارند.
- طبق خواص پایایی (durability) و یکپارچگی (atomicity) 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.

مدیریت ترمیم (recovery management)

- انواع خرابی (failure):

1. 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

مدیریت ترمیم (recovery management)

❖ انواع خرابی (failure):

2. خرابی سیستم (crash system): خرابی سخت افزاری یا نرم افزاری که سبب از کار افتادن سیستم (downشدن) 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.
3. 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.
4. خرابی ارتباطات: این دسته از خرابی ها مختص بانک های نامتمرکز می باشند.

مدیریت ترمیم (recovery management)

❖ انواع رسانه های ذخیره سازی:

❖ از جهت قابلیت نگهداری اطلاعات در صورت وقوع خرابی، رسانه های ذخیره سازی به سه دسته تقسیم می شوند:

❖ رسانه فرار(volatile storage):

❖ رسانه ای که در صورت وقوع خرابی سیستم، اطلاعات آن از بین می رود، مثل حافظه اصلی، حافظه نهان (cache) و ثبات (register).

❖ رسانه غیرفرار (non-volatile storage):

❖ خرابی سیستم را تحمل می کنند و اطلاعات آن ها حتی با وقوع خرابی سیستم، قابل بازیابی است مانند دیسک، نوار مغناطیسی، حافظه flash، RAMی که دارای باتری پشتیبان است.

مدیریت ترمیم (recovery management)

- رسانه پایدار (stable storage):
- رسانه ای که ایده آل است و در برابر تمام خرابی ها مصون می باشد. این نوع رسانه در واقع یک مفهوم منطقی است نه فیزیکی و هنوز چنین رسانه ای وجود خارجی ندارد، بلکه سعی می شود با راهکارهایی که از رسانه های فیزیکی موجود استفاده می کنند به این هدف ایده آل نزدیک شویم.
- متداولترین راهکار، استفاده از چندین کپی از داده ها روی رسانه های مختلف است (backup). با این کار سعی می شود احتمال از دست رفتن اطلاعات در صورت وقوع خرابی را به صفر هر چه نزدیکتر کنیم. هر چه تعداد کپی ها، ناهمگونی رسانه ها، درجه اطمینان آنها و ... بیشتر باشد، احتمال از دست رفتن کامل اطلاعات کمتر است. اما کپی کردن ها، به روز نگه داشتن داده ها و مدیریت آنها پیچیدگی های زیادی را به دنبال دارد.
- RAID یا Redundant Array of Independent Disks یکی از سیستمهای رایج در این راستا است.

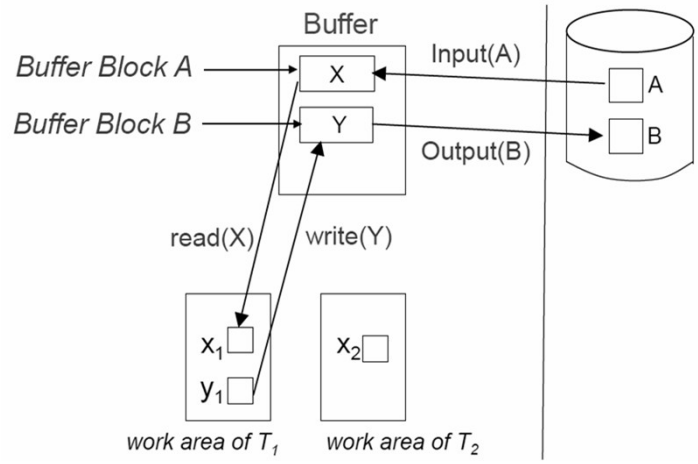
مدیریت ترمیم (recovery management)

- روال دسترسی به داده ها بر ای انجام تراکنش
- داده‌های بانک اطلاعات روی رسانه‌ها قرار دارند که معمولترین آنها دیسک می‌باشد. عمده‌ترین عامل سرعت در پردازش اطلاعات، تعداد دفعات دستیابی به رسانه است. در بانک‌های اطلاعات نامتمرکز علاوه بر این عامل، پهنای باند نیز اهمیت دارد.
- برای کاهش تعداد دفعات مراجعه به دیسک، رکوردها باهم بلوک‌بندی می‌شوند و در هر بار مراجعه به دیسک برای خواندن یا نوشتن، به جای یک رکورد، یک بلوک خوانده می‌شود. پس ازخواندن یک بلوک، اطلاعات آن بلوک به بخشی از حافظه اصلی منتقل می‌شوند که بافر(buffer) نامیده می‌شود. سپس تراکنش‌ها ازاین بافرها که به صورت RAM هستندو سرعت خیلی بالایی دارند، رکوردها را برداشته و عملیات‌ها را انجام می‌دهند. خروجی برعکس این فرایند است؛ یعنی تراکنش‌ها از ناحیه‌کاری(work area) خود، رکوردها را روی بافرها نوشته و باپرشدن بافر به‌یکباره به‌دیسک منتقل می‌شوند.

مدیریت ترمیم (recovery management)

- مرحله اول این کار یعنی انتقال اطلاعات از دیسک به بافر و بالعکس input()و output() نام دارد و مرحله دوم یعنی انتقال اطلاعات از بافرها به ناحیه های کاری تراکنش ها و بالعکس به نام read() و write() مرسوم است.
- سیستم های مختلف تعدادی بافر دارند که دارای سایز مشخصی هستند. بنابراین سایز بلوک هایی که تعیین می کنیم، یعنی تعداد رکوردهایی که در بلوک ها قرار می دهیم با توجه به سایز بافر می باشد.
- اگر سایز بلوک ها از سایز بافرها کمتر باشد در این صورت فضای بافر را از دست داده ایم و اگر سایز بلوک ها از سایز بافرها بیشتر باشد، در این صورت باید بافرها را به هم چسباند و آنها را خواند که کار بسیار سختی است. بنابراین سایز بلوک ها باید هر چه نزدیکتر به سایز بافرها باشد و نه بیشتر.

مدیریت ترمیم (recovery management)



مدیریت ترمیم (recovery management)

- ثابت شده که برای هر نوع IO بهتر است دو بافر اختصاص دهیم.
- یک بافر توسط IO Proccessor پر می شود؛ یعنی بلوک در آنجا قرار می گیرد که عملی طولانی است. بافر دیگر در صورتی که پر شده باشد به طور همزمان توسط cpu خالی می شود؛ یعنی به ترتیب رکوردهای آن در اختیار ناحیه کاری تراکنش قرار می گیرند و زمانی بافر پر و دیگری خالی شد، نقش آنها عوض می شود.
- به ازای هر IOداشتن دو بافر بهترین حالت ممکن است.

مدیریت ترمیم (recovery management)

- 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 شود که عموماً شامل دو مرحله زیر می باشند:

1. مرحله اول: در حین عملکرد عادی سیستم، اطلاعاتی که برای انجام ترمیم (پس از وقوع خرابی) به آنها نیاز داریم در جایی ثبت شوند. به این مکان log گویند.
2. مرحله دوم: پس از وقوع خرابی (وقتی سیستم دوباره بالا می آید)، با استفاده از اطلاعات ثبت شده در مرحله قبل و الگوریتم هایی که از پیش تهیه شده اند خواص یکپارچگی و پایایی تراکنش ها اعمال گردند و سیستم را به حالت اول برگردانیم.

مدیریت ترمیم (recovery management)

- الگوریتم های ترمیم
- 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 از الگوریتم ترمیم، هر تراکنشی که با موفقیت انجام (commit) شده است را تکرار (redo) کرده، یعنی دوباره اجرا می کنیم تا اثراتش روی بانک اطلاعات دائمی شود و هر تراکنشی که ساقط شده را خنثی (undo) می کنیم. فرض می کنیم تراکنش ها سریالی هستند.
- الگوریتم های ترمیم به دو رویکرد کلی تقسیم می شوند: رویکرد کارنامه (log-based) و رویکرد رونوشت (shadow paging)

مدیریت ترمیم (recovery management)

- رویکرد کارنامه (log-based)
- 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 T_i، رکورد $< T_i,start >$
- برای دستور write(x) از تراکنش T_i، در حالت کلی رکورد $<T_i,x,V_1,V_2>$ که V_1 و V_2 به ترتیب مقادیر قبل و بعد از انجام نوشتن x می باشند.
- با اجرای آخرین دستور تراکنش T_i، رکورد $<T_i,Commit>$ در کارنامه ثبت می شود.

مدیریت ترمیم (recovery management)

- رویکرد کارنامه (log-based)
- 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

(deferred database modification)

- انعکاس فوری تغییرات در بانک اطلاعات

(immediate database modification)

مدیریت ترمیم (recovery management)

- انعکاس معوق تغییرات در بانک اطلاعات
- تمامی رکوردهای کارنامه مربوط به انجام تغییرات، در کارنامه ثبت می شود اما انعکاس این تغییرات روی رسانه (یعنی اجرای واقعی writeها) تا زمان پس از انجام جزیی (partial commit) اجرای آخرین تراکنش به تعویق می افتد.
- در رکورد های کارنامه مربوط به دستور نوشتن تراکنش T_i، 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 رکوردهای $<T_i,start>$ و هم رکوردهای $<T_i,Commit>$ برای آنها موجوداست) را تکرار می‌کنیم (استفاده از log ها).
- تکرارشدن یک تراکنش یعنی تمامی داده‌هایی که تراکنش روی آنها نوشته است، بامقدار جدید موجود در رکورد کارنامه دستور نوشتن، مقداردهی می‌شوند.

مدیریت ترمیم (recovery management)

- انعکاس معوق تغییرات در بانک اطلاعات
- نکته: چنانچه از روش انعکاس معوق تغییرات استفاده کنیم، در انجام ترمیم نیازی به خنثی کردن (undo) نداریم، زیرا:
- تا تراکنشی تثبیت نشده باشد، نمی تواند محتوای رسانه را تغییر دهد (زیرا تراکنش هایی که تثبیت نشده اند ، به مرحله انجام جزئی نرسیده اند)
- و اگر هم نیمه کاره ساقط شود چون تاثیری روی بانک اطلاعاتی نگذاشته، نیازی به خنثی کردن نیست.
- برای همین رکورد نوشتن در این روش به صورت در حالت کلی رکورد $<T_i,x,V>$ است و فاقد مقدار قدیمی x است.

مدیریت ترمیم (recovery management)

- 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 بیفتد و نیاز به خنثی کردن تراکنش های نیمه کاره داشته باشیم پس ساختار رکورد کارنامه دستور نوشتن به صورت عمومی $<T_i,x,V_1,V_2>$ است یعنی هم مقدار قدیمی و هم مقدار جدید x را ثبت می نماییم.

مدیریت ترمیم (recovery management)

- آیا برای یک دستور نوشتن، ترتیب اجرای نوشتن روی رسانه و ثبت رکورد دستور نوشتن روی کارنامه تاثیر و اهمیت دارد؟
- 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 ثبت شده می توان عمل ترمیم را انجام داد. این قاعده پروتکل WAL (Write-Ahead Log) نام دارد.

مدیریت ترمیم (recovery management)

- در مرحله دوم از الگوریتم ترمیم که تغییرات را فوراً روی دیسک منعکس می کند:
- تمامی تراکنش هایی مثل T_i که انجام شده اند (هم رکورد های $<T_i,start>$ و هم رکورد های $<T_i,commit>$ 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.

مدیریت ترمیم (recovery management)

- 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 idempotent)

مدیریت ترمیم (recovery management)

- 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 (checkpoint) استفاده می شود.

مدیریت ترمیم (recovery management)

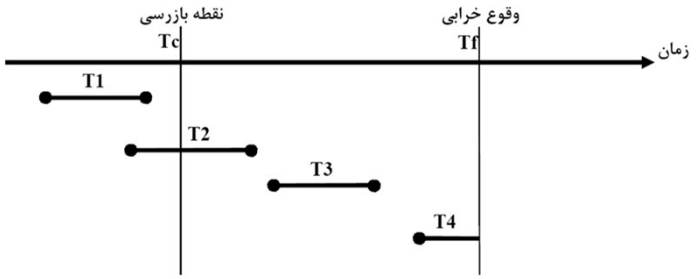
- نقاط بازرسی (checkpoints)
- در این روش به طور متناوب در برهه های زمانی معینی از عملکرد عادی سیستم، کارهای انجام شده روی بانک اطلاعات (تا آن زمان) را قطعی و نهایی می کنیم که به آن نقاط بازرسی گوییم.
- بنابراین اگر اتفاقی در سیستم رخ بدهد، نباید تا ابتدای log را بررسی کنیم، بلکه کافیست تا اولین نقطه بازرسی این کار را انجام دهیم.
- کافیست log را از آخر شروع کنیم، اطلاعات آن را مورد استفاده قرار دهیم، سیستم را ترمیم کنیم و زمانی که به اولین نقطه بازرسی رسیدیم متوقف شویم.

مدیریت ترمیم (recovery management)

- نقاط بازرسی (checkpoints)
- 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.

مدیریت ترمیم (recovery management)

- مثال: در این شکل پس از وقوع خرابی به صورت زیر عمل می شود:



- تراکنش T_1 در زمان T_c نهایی شده و نیازی به ترمیم ندارد.
- تراکنش T_2 و T_3 را بنا به خاصیت پایایی باید تکرار نمود تا اثر آنها در سیستم نهایی و دائمی شود. (برای T_2 فقط بخشی از دستورات که پس از T_c اجرا شده اند را تکرار می کنیم.)
- تراکنش T_4 که نیمه کاره مانده است را بنا به خاصیت یکپارچگی، خنثی می کنیم.

مدیریت ترمیم (recovery management)

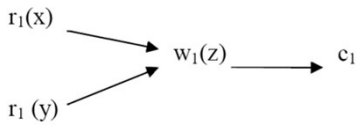
- رویکرد رونوشت
- رویکرد رونوشت خیلی معمول نیست. مهمترین دلیل این است که برای پیاده سازی رویکرد رونوشت، ما به کارنامه نیاز داریم. اگر قرار است که کارنامه و الگوریتم های پردازش کارنامه را داشته باشیم تا رویکرد رونوشت را پیاده ساری کنیم، بهتر است که رویکرد مبتنی بر کارنامه را انجام دهیم.
- قبل از انجام تغییرات (قبل از شروع به اجرای تراکنش) یک کپی از صفحات مورد نیاز بانک اطلاعات تهیه (نسخه سایه shadow) و در رسانه غیر فرار ذخیره می نماییم. نسخه جاری (current)، نسخه ای است که تغییرات مورد نظر روی آن انجام می شود.
- چنانچه تراکنش به انجام رسید، این نسخه جاری 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.

مدیریت ترمیم (recovery management)

- رویکرد رو نوشت
- مزا یای رویکرد رو نوشت:
  - سربار مربوط به نوشتن رکوردهای کارنامه را ندارد.
  - انجام عملیات ترمیم بسیار ساده و ناچیز است.
- معایب رویکرد رونوشت:
  - برای پیاده سازی آن نیاز به کارنامه داریم.
  - کپی کردن اطلاعات مربوطه از بانک بسیار پر هزینه است (حتی با وجود بهینه ساز ی های انجام شده در این زمینه).
  - توسعه این روش برای حالتی که تراکنشها بتوانند همروند اجرا شوند بسیار مشکل است (بر خلاف روش کارنامه).
  - سربار انجام عملیات تثبیت زیاد است.
- نتیجه: در سیستم های بانک اطلاعات رویکرد کارنامه از رویکرد رونوشت بهتر است و عموماً از روش کارنامه برای انجام ترمیم استفاده می شود.

تعاریف و قضیه ها (بیان تئوریک)

- بیان مفاهیم سابق به صورت ریاضی
- مثال: شکل زیر نشان می دهد که تراکنش T_1، به طور همزمان داده های y و x خوانده و پس از این دو رویداد، روی داده z نوشته و بعداً به انجام رسیده است.



- لبه‌هایی که بر اساس خاصیت انتقال قابل استنتاج هستند ($r_1(x) \rightarrow c_1$) رسم نمی کنیم.
- تراکنش یک partial order است. partial order یا ترتیب جزئی یعنی یک مجموعه ای که بعضی یا تعدادی از اعضای آن ترتیب خاصی دارند و اگر در یک مجموع های همه اعضاء ترتیب مشخصی داشته باشند به آن total order یا ترتیب کلی می گوییم. یک تراکنش به تنهایی در صورتی یک partial order است که بیش از یک CPU یا IO processor در اختیار داشته باشیم.

تعاریف و قضیه ها (بیان تئوریک)

- زمانبندی را با یک گراف فاقد حلقه (DAG) نمایش داد که در آن هر گره یکی از دستورات تراکنش که با $o_i(x)$ را نشان می دهد (o می تواند r یا w باشد). هر لبه $p \rightarrow q$ به معنی آن است که عملگر p قبل از عملگر q اجرا می شود.
- تعریف: دو عملگر با هم برخورد conflict دارند اگر، مربوط به تراکنش های متمایز باشند، رو ی یک داده کار کنند و حداقل یکی از آنها عملگر نوشتن باشد:

$$p_i(x) \asymp q_j(y) \Leftrightarrow (i \neq j \wedge x = y \wedge (p = w \vee q = w))$$

- برا ی دو عملگری که با هم برخورد دارند، هم در تراکنش و هم در زمانبندی ها حتماً باید ترتیب اجرای آنها را نسبت به هم تعیین کنیم، زیرا:
- در مورد $r_i(x)$ و $w_j(x)$: نتیجه خواندن داده، قبل و بعد از نوشتن روی آن داده ممکن است متفاوت باشد و مقداری که خوانده می شود بستگی به این دارد که خواندن، قبل از نوشتن اجرا می شود یا بعد از آن.
- در مورد $w_i(x)$ و $w_j(x)$: مقدار نهایی نوشته شده در داده x بستگی به این دارد که کدام عملگر w آخر اجرا شود.

مدیریت تراکنش

تعاریف و قضیه ها (بیان تئوریک)

- تعریف: تراکنش T_i ترتیب جزئی با $<_i$ است و دارای خواص زیر می باشند:
- T_i شامل مجموعه عملگرهای خواندن و نوشتن داده‌ها و تثبیت یا ساقط می‌باشد.

$$T_i \subseteq \{r_i(x), w_i(x) \mid x \text{ داده است}\} \cup \{a_i, c_i\}$$

- یکی از دو عملگر تثبیت یا ساقط باید وجود داشته باشد و نه هردوی آنها.

$$a_i \in T_i \Leftrightarrow c_i \notin T_i$$

- عملگر تثبیت یا ساقط آخرین عملگر است.

$$(t = c_i \vee t = a_i) \Rightarrow \forall p \in T_i,\ p <_i t$$

- ترتیب اجرای خواندن و نوشتن تراکنش روی یک داده حتما باید مشخص شده باشد.

$$\forall r_i(x), w_i(x) \in T_i \Rightarrow r_i(x) <_i w_i(x) \vee w_i(x) <_i r_i(x)$$

تعاریف و قضیه ها (بیان تئوریک)

❖ زمانبندی کامل (complete schedule)

❖ تعریف: اگر $T=\{T_1,T_2,\ldots, T_n\}$ مجموعه تراکنش ها باشد، آنگاه زمانبندی کامل H روی T، ترتیب جزئی را با رابطه $<_H$ است که:

۱- H شامل تمام تراکنش ها می باشد:

$$H = \cup_{i=1}^{n} T_i$$

۲- ترتیب اجرای دستورات تراکنش ها حفظ می گردد و ترتیب های دیگری نیز ممکن است اضافه شوند:

$$<_H \supseteq \cup_{i=1}^{n} <_i$$

۳- برای هر دو عملگر دارای برخورد، باید ترتیب آنها مشخص شود:

$$\forall\ p_i(x), q_j(y) \in T: p_i(x) \asymp q_j(y) \Rightarrow p_i(x) <_H q_j(y) \vee q_j(y) <_H p_i(x)$$

❖ تعریف: زمانبندی، پیشوندی (prefix) از یک زمانبندی کامل می باشد. پیشوند یعنی از ابتدا شروع کنید و تا یک جایی پیش بروید. آنچه که در واقعیت بانک های اطلاعاتی وجود دارد زمانبندی است. زمانبندی یک مفهوم پویاست در حالی که زمانبندی کامل یک مفهوم ایستا می باشد.

تعاریف و قضیه ها (بیان تئوریک)

❖ مثال: تراکنش های زیر را در نظر بگیرید:

$T_1 = r_1(x) \rightarrow r_1(y) \rightarrow w_1(x) \rightarrow c_1$

$T_2 = r_2(x) \rightarrow w_2(y) \rightarrow w_2(x) \rightarrow c_2$

$T_3 = r_3(y) \rightarrow w_3(x) \rightarrow w_3(y) \rightarrow c_3$

❖ زمانبندی کامل H_1 روی مجموعه تراکنش های بالا به صورت زیر میباشد:

$$H_1 = \begin{array}{l} r_1(x) \rightarrow r_1(y) \rightarrow w_1(x) \rightarrow c_1 \\ \quad\quad\quad\quad\ \downarrow \quad\quad\quad\quad\ \uparrow \\ r_2(x) \rightarrow w_2(y) \rightarrow w_2(x) \rightarrow c_2 \\ \quad\quad\quad\quad\ \downarrow \quad\quad\quad\quad\ \uparrow \\ \quad\quad\quad\quad r_3(y) \rightarrow w_3(x) \rightarrow w_3(y) \rightarrow c_3 \end{array}$$

❖ زمانبندی H_1' پیشوندی از H_1 است:

$$H'_1 = \begin{array}{l} r_1(x) \rightarrow r_1(y) \rightarrow w_1(x) \\ \quad\quad\quad\quad\ \downarrow \quad\quad\quad\quad\ \uparrow \\ r_2(x) \rightarrow w_2(y) \rightarrow w_2(x) \rightarrow c_2 \\ \quad\quad\quad\quad\ \downarrow \quad\quad\quad\quad\ \uparrow \\ \quad\quad\quad\quad r_3(y) \rightarrow w_3(x) \end{array}$$

تعاریف و قضیه ها (بیان تئوریک)

- تعریف: تراکنش T_i در H تثبیت (یا ساقط) شده است اگر $a_i \varepsilon H$ ($c_i \varepsilon H$) در غیر این صورت T_i فعال است.
- نکته: زمانبندی کامل، تراکنش فعال ندارد.
- تعریف: پرتو ثابت (committed projection) زمانبندی H که با C(H) نمایش داده می شود با حذف تمام عملگرهای تراکنش های تثبیت نشده حاصل می گردد.

$$C(H)=\{\, p_i(x) \mid 1 \le i \le n,\quad p_i(x) \in H \wedge c_i \in H \,\}$$

- نکته: پرتو ثابت، یک زمانبندی کامل روی مجموعه تراکنش های تثبیت شده می باشد که نه تراکنش فعال دارد و نه تراکنش ساقط شده.
- باید توجه داشت که بر روی تراکنش های فعال نمی توان حساب کرد، زیرا تراکنش های فعال ممکن است به دلیل وقوع خرابی ساقط شوند و تراکنش های ساقط شده نیز تأثیری روی بانک اطلاعات ندارند.

تعاریف و قضیه ها (بیان تئوریک)

- اگر طرح همروند را طوری تغییر دهیم که به یکی از طرح های پی در پی برسیم به شرطی که دستورات با هم برخورد نداشته باشند به آن پی در پی پذیر در برخورد گفته می شود.
- تعریف: دو زمانبندی H´ و H معادل در برخورد هستند اگر و تنها اگر:
- ۱- مجموعه تراکنش ها و مجموعه عملگرهایشان یکسان باشد.
- ۲- ترتیب عملگرهای دارای برخورد تراکنش های ساقط نشده، در هر دو، یکسان باشد:

$$\forall p_i(x), q_j(y) : p_i(x) \asymp q_j(y) \wedge$$
$$a_i, a_j \notin H \wedge a_i, a_j \notin H' (p_i(x) <_H q_j(y) \Leftrightarrow p_i(x) <_{H'} q_j(y))$$

تعاریف و قضیه ها (بیان تئوریک)

❖ مثال: در زمانبندی های زیر H_2 و H_3 معادل هستند اما H_4 و H_5 با هیچ کدام معادل نیستند.

✓ همه خواص زمانبندی را دارند. ✓ H_4 شرط ۱ و H_5 شرط ۲ را ندارند.

H_2: $r_1(x) \rightarrow w_1(y) \rightarrow w_1(z)$
$\downarrow \quad \downarrow$
$w_2(y) \rightarrow r_2(z) \rightarrow c_2$

H_3: $r_1(x) \rightarrow w_1(y) \rightarrow w_1(z)$
$\downarrow$
$w_2(y) \rightarrow r_2(z) \rightarrow c_2$

H_4: $r_1(x) \rightarrow w_1(y) \rightarrow w_1(z) \rightarrow w_1(t) \rightarrow c_1$
$\downarrow$
$w_2(y) \rightarrow r_2(z) \rightarrow c_2$

H_5: $r_1(x) \rightarrow w_1(y) \rightarrow w_1(z)$
$\downarrow \quad \uparrow$
$w_2(y) \rightarrow r_2(z) \rightarrow c_2$

تعاریف و قضیه ها (بیان تئوریک)

❖ تعریف: تراکنش T_i در زمانبندی H قبل از T_j اجرا می شود اگر و تنها اگر تمام عملگرهای T_i قبل از تمام عملگرهای T_j اجرا شود.

$$T_i <_H T_j \Leftrightarrow \forall p_i(x) \in T_i, \forall q_j(y) \in T_j (p_i(x) <_H q_j(y))$$

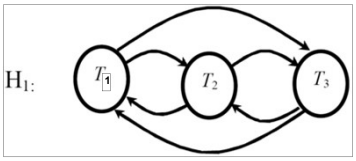
❖ زمانبندی ها را نمی توانیم به صورت پی درپی تعریف کنیم به دلیل اینکه تراکنش فعال دارند و در زمانبندی های پی درپی فقط تراکنش آخر می تواند فعال باشد. بنابراین به صورت مقدماتی زمانبندی کامل به صورت پی درپی تعریف می شود.

❖ تعریف: زمانبندی کامل H را پی درپی گو ییم اگر بر ای هر دو تراکنش T_i و T_j در H، تراکنش T_i قبل از T_j اجرا شود یا بالعکس.

$$H \in SER \Leftrightarrow \forall T_i, T_j \in H (T_i <_H T_j \vee T_j <_H T_i)$$

تعاریف و قضیه ها (بیان تئوریک)

- پیش تعریف: زمانبندی H پی درپی پذیر در برخورد است اگر معادل در برخورد با یک زمانبندی پی در پی باشد.
- زمانبندی H در صورتی پی درپی پذیر است که کامل باشد، زیرا باید همه تراکنش‌های آن خاتمه یافته باشند. پس باید به راهکار جدیدی بیاندیشیم.
- تعریف: زمانبندی H پی درپی پذیر در برخورد(CSR) است اگر پرتو ثابت آن معادل در برخورد با زمانبندی پی در پی H_s باشد.
- تعریف: در گراف پی درپی پذیری H که آن را با SG(H) نمایش می دهیم، گره ها تراکنش ها هستند و لبه $T_i \rightarrow T_j$ $(i \neq j)$ در صورتی وجود دارد که در H عملگری ناسازگار از T_i قبل از عملگری ناسازگار از T_j آمده باشد.
- مثال: گراف پی درپی پذیری زمانبندی H_1 بالا به صورت زیر می باشد:

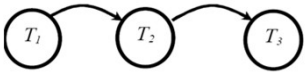


تعاریف و قضیه ها (بیان تئوریک)

- قضیه ۱: زمانبندی H پی در پی پذیر در برخورد است اگر و فقط اگر SG(H) فاقد حلقه باشد.
- اثبات:
- ۱) اگر H پی در پی پذیر در برخورد باشد آنگاه می توان دستورات بدون برخورد تراکنش ها را نسبت به هم جابه جا کردو به یک معادل پی در پی رسید. به این معنی که دو دستور ناسازگار در دو تراکنش یافت نمی شود که در جایی یکی قبل از دیگری و در جایی دیگر برعکس باشد یعنی در SG(H) حلقه وجود ندارد.
- ۲) در صورتیکه در SG(H) حلقه وجود نداشته باشد همواره می توان دستورهای ناسازگار بین دو تراکنش را از هم عبور داد تا یکی از دو تراکنش پس از دیگری قرار گیرد. یعنی می توان به معادل پی در پی رسید.

تعاریف و قضیه ها (بیان تئوریک)

- قضیه ۲: در گراف پی درپی پذیری خاصیت انتقال وجود ندارد.
- در گراف پی درپی پذیری زمانبندی $r_1(x)\ w_2(x)\ r_2(y)\ w_3(y)$ گره های T_1، T_2 و T_3 و لبه های $T_1 \rightarrow T_2$ و $T_2 \rightarrow T_3$ وجود دارند. نکته قابل توجه این است که $T_1 \rightarrow T_2$ با داده x به وجود آمده است ولی $T_2 \rightarrow T_3$ با داده y و از آنجا که این لبه ها روی دو داده مختلف ایجاد شده‌اند، لبه $T_1 \rightarrow T_3$ وجود ندارد.



- در زمانبندی H تراکنش T_i داده x را از تراکنش T_j می خواند (read-from)، اگر T_j آخرین عمل نوشتن روی x را انجام داده باشد و قبل از خواندن T_i ساقط نشده باشد.
- در زمانبندی H تراکنش T_i از تراکنش T_j می‌خواند اگر T_i داده‌ای را از T_j بخواند.
- در زمانبندی H تراکنش T_i آخرین نوشتن (final write) داده x را انجام می دهد اگر T_i ساقط نشده باشد و هر تراکنش T_j دیگری که روی آن داده می نویسد، یا قبل از T_i بنویسد یا ساقط شده باشد.

تعاریف و قضیه ها (بیان تئوریک)

- برای آنکه زمانبندی صحیح باشد، یعنی جامعیت بانک اطلاعات را حفظ کند، هم باید از نظر همروندی مشکلی نداشته باشد و هم از نظر ترمیم.
- همروندی: امکان استفاده تراکنش از داده های مشترک بانک اطلاعات بدون اینکه روی سایر تراکنش ها تاثیر مخرب بگذارد.
- ترمیم پذیری: اثر دائمی تراکنش هایی که پایان یافته اند و خنثی کردن اثر تراکنش هایی که پایان نیافته اند.
- تعریف: زمامنبندی H را ترمیم پذیر (Recoverable) می نامیم اگر، هرگاه T_i از T_j $(i \neq j)$ بخواند، تثبیت T_j قبل از تثبیت T_i باشد.

$$H \in RC \Leftrightarrow \forall T_i, T_j \in H, i \neq j \left(T_j \xrightarrow[H]{*} T_i \Rightarrow c_j <_H c_i\right)$$

- نکته: اشکال عمده زمانبندی های ترمیم پذیر سقوط های آبشاری است.
- تعریف :زمانبندی H را فاقد سقوط آبشاری (ACA) می نامیم اگر، هرگاه T_i از T_j $(i \neq j)$ بخواند، آنگاه تثبیت T_j قبل از خواندن، T_i باشد.

$$H \in ACA \Leftrightarrow \forall T_i, T_j \in H, i \neq j, \forall x \left(T_j \xrightarrow[H]{x} T_i \Rightarrow C_j <_H r_i(x)\right)$$

تعاریف و قضیه ها (بیان تئوریک)

- زمانبندی های سقوط آبشاری و ترمیم پذیر فقط به مفهوم read from می پردازند و به عملگر write کاری ندارند و این عملگرهای write نیز ممکن است مشکل ایجاد کنند. بنابراین زمانبندی بهتری را که کامل نیز است معرفی می کنیم به نام زمانبندی محض یا سخت گیر.

- تعریف: زمانبندی H را محض(سخت گیر) می نامیم هرگاه هر عمل $o_i(x)$ که بعد از $w_j(x)$ انجام می شود، حتما پس از خاتمه T_jباشد.

- به طور خلاصه هر کس از داده ای تأثیری می پذیرد، تکلیف تأثیرگذار باید قبلاً مشخص شده باشد.

$$H \in ST \Leftrightarrow \forall o_i(x) : (w_j(x) <_H o_i(x), i \neq j \Rightarrow c_j <_H o_i(x) \vee a_j <_H o_i(x))$$

تعاریف و قضیه ها (بیان تئوریک)

- مثال: تراکنش ها و زمانبندی های زیر را در نظر بگیرید:

T_1: $r_1(x)\ w_1(y)\ w_1(z)$
T_2: $r_2(u)\ w_2(y)\ r_2(z)$

H_8: $r_1(x)\ w_1(y)\ r_2(u)\ w_2(y)\ w_1(z)\ r_2(z)\ c_2\ c_1$
H_9: $r_1(x)\ w_1(y)\ r_2(u)\ w_2(y)\ w_1(z)\ r_2(z)\ c_1 c_2$
H_{10}: $r_1(x)\ w_1(y)\ r_2(u)\ w_2(y)\ w_1(z)\ c_1 r_2(z)\ c_2$
H_{11}: $r_1(x)\ w_1(y)\ r_2(u)\ w_1(z)\ c_1\ w_2(y)\ r_2(z)\ c_2$
H_{12}: $r_1(x)\ r_2(u)\ w_1(y)\ a_1\ w_2(y)\ r_2(z)\ c_2$

- نکته: اگر زمانبندی محض باشد، یقینا فاقد سقوط های آبشاری نیز هست و اگر فاقد سقوط های آبشاری باشد، ترمیم پذیر نیز می باشد. عکس این ادعا همیشه صحیح نیست.

تعاریف و قضیه ها (بیان تئوریک)

❖ مثال: تراکنش ها و زمانبندی های زیر را در نظر بگیرید:

	RC	ACA	ST
H_8	×	×	×
H_9	√	×	×
H_{10}	√	√	×
H_{11}, H_{12}	√	√	√

❖ زمانبندی H_8 ترمیم پذیر نیست زیرا T_2 داده z را از T_1 می خواند اما $c_2<c_1$.

❖ زمانبندی H_9 اگر چه ترمیم پذیر است اما فاقد سقوط های آبشاری نیست زیرا T_2 داده z را قبل از تثبیت T_1 از آن می خواند.

❖ زمانبندی H_{10} فاقد سقوط های آبشاری است اما سخت گیر نیست زیرا عمل نوشتن T_2 در y بعد از نوشتن T_1 در y و قبل از اتمام T_1 انجام شده است.

❖ زمانبندیهای H_{11} و H_{12} محض می باشند.

تعاریف و قضیه ها (بیان تئوریک)

❖ قضیه ۳: $ST \subset ACA \subset RC$

❖ در تعریف زیرمجموعه محض اگر $A \subset B$ باشد، اولاً A زیرمجموعه B است و ثانیاً این مساوی نیستند (در B چیزی است که در A نیست)

❖ الف- $ST \subseteq ACA$

❖ فرض کنید که در زمان بندی محض H، تراکنش T_i داده x را از تراکنش T_j $(i\neq j)$ بخواند، پس داریم: $w_j(x)<_H r_i(x)$ و $c_j<_H r_i(x)$. از آنجا که $r_i(x) <_H c_i$ می باشد در نتیجه $w_j(x)<_H c_i$، بنابراین H فاقد سقوطهای آبشاری نیز هست.

❖ اما با توجه به مثال فوق، زمان بندی مانند H_{10} وجود دارد که ACA هست ولی ST نیست، پس ACA≠ ST و لذا $ST \subset ACA$.

❖ ب- $ACA \subseteq RC$

❖ فرض کنید H، یک زمان بندی فاقد سقوط های آبشاری باشد و تراکنش تراکنش T_i داده x را از تراکنش T_j $(i \neq j)$ بخواند، و T_i تثبیت شده باشد، پس داریم: $w_j(x)<_H c_j<_H r_i(x)$ وچون c_i Hε و $r_i(x) <_H c_i$ لذا $c_j <_H c_i$ بنابراین H ترمیم پذیر نیز هست.

❖ اما با توجه به مثال فوق، زمان بندی مانند H_9 وجود دارد که RC هست ولی ACA نیست، پس ACA≠ RC و لذا $ACA \subset RC$.

تعاریف و قضیه ها (بیان تئوریک)

- قضیه ۴: زمان بندی قفل دومرحله ای پایه دارای خاصیت CSR می باشد؛ یعنی پی درپی پذیری در برخورد را تضمین می کند.
- اثبات: در گراف پی درپی پذیری یک زمان بندی قفل دومرحله ای حلقه ای وجود ندارد. لذا طبق قضیه ۱زمان بندی قفل دومرحله ای پایه B2PL، پی درپی پذیر در برخورد می باشد.
- قضیه ۵: پروتکل قفل دومرحله ای پایه تضمین می کند که مشکل بازیابی ناهمگام پیش نیاید.
- اثبات: مشکل بازیابی ناهمگام زمانی پیش می آید که داده های مورد دستیابی بعضاً به هم وابستگی داشته باشند. از آنجا که این وابستگی ها قابل شناسایی نیستند، مجبوریم فرض کنیم که همه داده های مورد دستیابی به هم وابسته اند. نگهداری قفل همه داده ها و جداکردن فاز گرفتن قفل از آزاد کردن قفل، این فرض را برآورده می سازد.
- نتیجه: پروتکل قفل دومرحله ای پایه، هر سه مشکل همروندی را حل می کند.

تعاریف و قضیه ها (بیان تئوریک)

- $ol_i(x)$ برای درخواست قفل، $ou_i(x)$ برای آزاد نمودن قفل عملگر o.
- قضیه ۶: در گراف انتظار یک زمانبندی C2PL حلقه وجود ندارد و بن بست رخ نخواهد.
- اثبات: در این زمانبندی، اگر تراکنشی مثل T_i منتظر قفلی باشد که در اختیار T_j قرار دارد، پس T_i قبل از گرفتن قفل، شروع به اجرا نموده است و منتظر مانده که خلاف فرض است، بنابراین هیچ تراکنشی منتظر آزاد کردن قفل نمی ماند.
- قضیه ۷: قفل دومرحله ای محض، اجرای محض زمانبندی را تضمین می کند.
- اثبات: اگر H یک زمانبندی قفل دو مرحله ای محض باشد و $w_i(x) <_H o_j(x)$ آنگاه داریم:

$$wl_i(X) <_H w_i(X) <_H wu_i(X) \text{ و } ol_j(X) <_H o_j(X) <_H ou_j(X)$$

- با توجه به اینکه $ol_j(x)$ و $wl_i(x)$ با هم برخورد دارند، باید داشته باشیم: $wu_i(x) <_H ol_j(x)$ یا $ou_j(x) <_H wl_i(x)$ که با فرض $w_i(x) <_H o_j(x)$ در تناقض است. و لذا $wu_i(x) <_H ol_j(x)$.

تعاریف و قضیه ها (بیان تئوریک)

- اما از آنجا که H از نوع S2PL است باید یکی از دو شرط زیر برقرار باشد:
- $c_i <_H wu_i(x)$ یا $a_i <_H wu_i(x)$
- از مجموعه قواعد حاصله برمی آید که یا $c_i <_H o_j(x)$ و یا $a_i <_H o_j(x)$ یعنی Hیک اجرای محض است.
- این خاصیت برای ما بسیار مهم است زیرا به طور ضمنی، ترمیم پذیر بودن و فاقدسقوط های آبشار ی بودن را نیز تضمین می نما ید. $ST \subset ACA \subset RC$
- ویژگی فوق را شاید بتوان مظهری از اوج نبوغ یک متخصص بانک اطلاعات دانست که ضمن حل مسئله پی درپی پذیری، مشکل ترمیم پذیری را نیز حل کرده و به قول معروف با یک تیر، دو نشان زده است.
- همچنین در قفل دومرحله ای محض، به ویژه به دلیل آزاد کردن قفل ها در خاتمه تراکنش، نیاز به ارسال پیام آزاد کردن قفل از بین رفته و حجم مراودات کاهش چشمگیری می یابد.

امنیت در بانک اطلاعات

- امروزه سازمان ها بیش از پیش وابسته به اطلاعاتی هستند که افراد، منابع و امور سازمان را مدیریت می کنند. بنابراین تخلف در امنیت اطلاعات ممکن است کل سیستم را به خطر بیاندازد. فقط کافی است تصور کنید که چه اتفاقی ممکن است بیفتد اگر اطلاعات مربوط به بیماران یک بیمارستان به صورت نادرست تغییر داده شود.
- تعریف: امنیت (Security) یعنی محافظت از تلاش‌های تبهکارانه برای سرقت Disclosure، تغییرAlternation و یا تخریبDestruction داده‌های بانک اطلاعات.
- تهدیدات امنیت به دو صورت عمدی و غیرعمدی هستند. دسته اول شامل تمام تهدیداتی است که نتیجه تخلف عمدی کاربران و یا برنامه ها است. این کاربران می توانند کاربران مجاز سیستم و یا کاربران خارجی (که می توانند به صورت غیر مجاز به سیستم و منابع آن دسترسی داشته باشند) باشند. برنامه ها نیز می توانند هر برنامه محلی و یا هر برنامه از راه دور (شامل ویروس ها یا اسب های تراوا و ...) باشند. دسته دوم شامل تهدیداتی است که بر اثر ناآگاهی یا عدم دقت و کوتاهی افراد و نارسایی و کیفیت پایین برنامه ها رخ می دهد.

امنیت در بانک اطلاعات

❖ هنگامی که صحبت از یک بانک اطلاعات امن به میان می آید معمولاً سه هدف زیر در رابطه با آن مطرح می شود:

۱- محرمانگی (security): محرمانگی به صورت عدم دسترسی کابران غیر مجاز به اطلاعات تعریف می شود. به عنوان مثال یک دانشجو نباید اجازه مشاهده نمره سایردانشجویان را داشته باشد.

۲- جامعیت (integrity): فقط کاربران مجاز می توانند داده های مربوطه را تغییر دهند. به عنوان مثال دانشجویان می توانند نمرات خود را ببینند اما اجازه تغییر آن را ندارند. جامعیت به صورت جلوگیری از تغییر، حذف و یا دخالت ناخواسته و نادرست در اطلاعات نیز تعریف می شود.

۳- دسترس پذیری (availability): مجوز کاربران مجاز نباید به طور ناخواسته قطع شود. به عنوان مثال استاد درسی که اجازه تغییر نمرات را دارد، همواره باید بتواند این عمل را انجام دهد.

امنیت در بانک اطلاعات

❖ برای رسیدن به سه هدف فوق باید سیاستهای امنیتی واضح و مشخصی تدوین گردد. به عبارت دیگر باید به طور کامل و صریح روشن گردد که چه بخش یا بخش هایی از داده ها باید محافظت شوند و چه کاربرانی اجازه دسترسی به چه قسمت هایی از داده ها را دارند و نیز کاربران چه اعمالی مجازند انجام دهند.

❖ این نکته بسیار مهم است که اغلب کاربران ما فقط به بخش کوچکی از داده های بانک اطلاعات دسترسی دارند. بنابراین اینکه اجازه بدهیم همه به کل بانک اطلاعات دسترسی داشته باشند کار بسیار اشتباهی است.

❖ برای برقراری امنیت در سیستم بانک اطلاعات:

✓ ابتدا به کمک مکانیزم تصدیق هویت کاربر (کلمه عبور) اطمینان حاصل می شود که کاربر وارد شده مجاز است.

✓ سپس با مکانیزم کنترل دسترسی فقط مجوزهای دسترسی به داده های به خصوص که می تواند به آنها دسترسی داشت باشد به او تخصیص داده می شود.

✓ کاربر مجاز ممکن است روی داده های مجاز خود تغییرات غیر مجاز بدهدکه با قرار دادن قیود جامعیت از داده ها مراقبت می شود.

امنیت در بانک اطلاعات

- کنترل دسترسی (access control)
- کنترل دسترسی باعث می شود که وقتی درخواست هایی برای دستیابی به بانک اطلاعات مطرح می شود، سیستم آنها را ارزیابی کند که قبول و یا رد کند.
- در این رابطه باید دو مورد را از هم تفکیک کرد؛ سیاست ها و مکانیزم ها.
- سیاست به راهبردهای سطح بالایی گفته می شود که چگونگی انجام کارها و قواعد را معین می نماید.
- مکانیزم به توابع نرم افزاری و راهکارهای سخت افزاری سطح پایینی گفته می شود که چگونگی پیاده سازی سیاست ها را بیان می کنند.

امنیت در بانک اطلاعات

- جداسازی سیاست از مکانیزم چندین فایده دارد:
  ۱. امکان مقایسه سیاست های مختلف و ارزیابی خصوصیات آنها را می دهد، بدون آن که نیازی به چگونگی پیاده سازی آنها باشد.
  ۲. امکان ایجاد مکانیزم‌های جدید که سیاست‌های جدیدرا اجرا می‌کنند، به وجود می‌آید به طوری که در تغییر یک سیاست احتیاج به تغییر کل‌پیاده‌سازی نیست.
  ۳. وجود مکانیزم‌های مختلف که بخش‌هایی از چندین سیاست را در یک زمان اجرا می‌کنند، این امکان را به کاربران می‌دهد که بهترین سیاست را انتخاب کنند.
- سیاست کنترل دسترسی به روش های زیر تقسیم می شود:
  - ✓ کنترل دسترسی محتاطانه DAC (Discretionary Access Control)
  - ✓ کنترل دسترسی الزامی MAC (Mandatory Access Control)
  - ✓ کنترل دسترسی مبتنی بر نقش RBAC (Role Based Access Control)
- کلمه دیگری نیز برای اینها به‌کار رفته با نام مدل که عبارت است از مجموعه‌ای از سیاست ها که کار دسترسی به یک بانک اطلاعات را مشخص می‌کند.

امنیت در بانک اطلاعات

- کنترل دستر سی محتاطانه (Discretionary Access Control)
- این سیاست ها، دسترسی کاربران به بانک اطلاعات را بر اساس هویت کاربران (Id) و قوانینی به نام اجازه ورود (مجوز) کنترل می کنند. در اینجا برای هر کاربر (یا گروهی ازکاربران) نوع دسترسی های آنها به هر شیء موجود در بانک اطلاعات مشخص می شود.
- نوع این اشیاء بستگی به بانک اطلاعات دارد. در مدل رابطه ای، اشیاء می توانند رابطه ها (جداول)، دیدها، صفت ها و سطرهای جدول باشند. در مدل شیءگرایی می توانند شامل کلاس ها، اشیاء و متدها باشند.
- کنترل های قابل اجرا بر روی اشیاء در مدل رابطه ای معمولاً در دو سطح صورت می گیرد:

۱) سطح account: امتیازهای مختلف توسط مدیر بانک اطلاعات، تشخیص و به کاربران مختلف اعطا می‌شود و در همه رابطه‌ها قابل اعمال می‌باشد.

امنیت در بانک اطلاعات

۲) سطح جدول: امتیاز های اعطا شده فقط برای بخش خاصی از جدول می باشد. معمولاً انواع مجوزهایی که به یک کاربر داده می‌شود به صورت زیر است.

- در سطح داده ها یعنی اینکه کاربر به چه داده هایی می تواند دسترسی داشته باشد و یا تغییر بدهد.
  - مجوز read: اجازه خواندن داده‌ها را می دهد ولی تغییر داده‌ها را نمی دهد.
  - مجوز insert: اجازه درج داده جدید را می دهد.
  - مجوز update: اجازه تغییر داده ها را می دهد.
  - مجوز delete: اجازه حذف داده ها را می دهد.
- در سطح شما (schema) یعنی اینکه آیا کاربر می تواند شکل و شمایل جدول را تغییر دهد یا حذف یا اضافه کند.
  - مجوز index: اجازه ایجاد یا حذف شاخص ها را می دهد.
  - مجوز resource: اجازه ا یجاد رابطه های جدید را می دهد.
  - مجوز alteration: اجازه اضافه یا حذف صفات رابطه را می دهد.
  - مجوز drop: اجازه حذف رابطه ها را می دهد.

امنیت در بانک اطلاعات

- سیاست های دسترسی نیز به دو گونه زیر تعریف می شوند:
- ۱. سیاست های بسته: در این نوع سیاست ها دسترسی هایی اجازه داده می شود که مجوز صریح آنها موجود باشد و تصمیم پیش فرض این است که دسترسی رد شود. اکثر سیستم ها از سیاست های بسته پشتیبانی می کنند.
- ۲. سیاست های باز: در این سیاست ها، مجوز منفی داده می شود و در صورت وجود آن، اجازه دسترسی داده نمی شود و تصمیم پیش فرض (در صورت عدم وجود مجوز منفی) این است که دسترسی قبول شود. سیاست باز فقط در سیستم هایی استفاده می شود که به حفاظت محدود نیاز دارند و اکثر دسترسی ها اجازه داده می شود.
- تعیین مجوز بر ای هر کاربر، هر مد دسترسی، و هر شیء، بار اجرایی زیادی را بر سیستم مدیریت بانک اطلاعات می گذارد. با گروه بندی کاربران، مدها و اشیاء این مجوزها برای گروهی از کاربران، دسته ای از مدها و یا مجموعه ای از اشیاء نگهداری می شود.

امنیت در بانک اطلاعات

- ممکن است یک گروه از کاربران دارای مجوز مثبت و چند استثناء با مجوز منفی برای چند کاربر باشد. در این حالت برخوردهایی پیش می آید. مثلاً فرض کنید یک کاربر به دو گروه تعلق داشته باشد و یکی از گروه ها برای یک دسترسی، دارای مجوز مثبت و دیگری برای همین دسترسی دارای مجوز منفی باشد. راه حل های مختلفی برای برخورد دستر سی ها وجود دارد:
  - ✓ مجوزهای منفی حفظ می شوند (اولویت با رد شده ها)
  - ✓ برخورد ممکن است از طریق رابطه های ممکن بین گروه ها رفع شود. مثلاً اگر یکی از گروه ها عضو دیگری باشد ممکن است مجوز تعیین شده برای گروه اول نگهداری شود.
  - ✓ تعیین اولویت های صریح (در حالت برخورد) به طوریکه مجوز با اولویت بیشتر نگهداری شود.

امنیت در بانک اطلاعات

- نمایش و اعمال مجوز
- ما چگونه می توانیم در سیستم خود، این روش ها و سیاست های مختلف را پیاده سازی کنیم؟ یعنی چگونه می توانیم این مجوزها را نمایش دهیم و آنها را اعمال کنیم؟
- یکی از راه های نمایش این مجوزها استفاده از ماتریس دسترسی (access matrix) است. در این ماتریس:
- سطرها نشان دهنده کاربران و فرایندها (subjects) می باشند.
- ستون ها نشان دهنده اشیای موجود (objects) می باشند.
- درایه ها مد دسترسی (mode) کاربر به آن شی ء مربوطه می باشد.
- متأسفانه ماتریس دسترسی ممکن است بسیار بزرگ و پراکنده باشد. همچنین ذخیره مجوزها در این ماتریس ممکن است باعث ناکارایی گردد.
- اما راه حل معمولی که در اکثر بانک های اطلاعات قرار می گیرد، گراف مجوز کاربر است.

امنیت در بانک اطلاعات

- سه روش نشان دادن ماتریس: لیست کنترل دسترسی (access control list): ماتریس براساس ستونها ذخیره می شود. به همراه هر شیء یک لیست وجود داردکه شامل کاربران و مد دسترسی آنها به این شیء می باشد.

  بررسی سریع مجوزها برای یک شیء امکان پذیر است ولی برای بدست آوردن همه مجوزهای یک کاربر باید تمام لیست های دسترسی بررسی شود.
- قابلیت (capability): ماتریس براساس سطرها ذخیره می شود. به همراه هر کاربر یک لیست وجود دارد که برای هر شیء موجود در بانک اطلاعات دسترسی کاربر به این شیء را نشان می دهد.

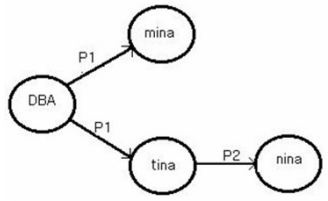
  تعیین سریع امتیازهای کاربر امکان پذیر است ولی به دست آوردن همه دسترسی های ممکن به یک شیء نیاز به بررسی کل قابلیتهای موجود دارد.
- جدول مجوز (authorization table): هر درایه غیرتهی از ماتریس در یک جدول سه ستونی که صفات آن کاربران، اشیا و مدل های دسترسی است نشان داده می شود.

امنیت در بانک اطلاعات

- گراف مجوز کاربر (authorization graph)
- مدیر بانک اطلاعات (DBA) ریشه گراف است.
- هر یک از کاربران گره دیگری از این گراف هستند.
- هر ضلعی از گره U به W با برچسب P مشخص می شود به این معنی که U امتیاز P را به W واگذار کرده است.
- مثال در SQL:
- DBA دستور زیر را اجرا می کند.
- GRANT SELECT ON Student TO tina, mina WITH GRANT OPTION;
- کاربر tina دستور زیر را اجرا می کند.
- GRANT SELECT ON Student TO nina;

امنیت در بانک اطلاعات

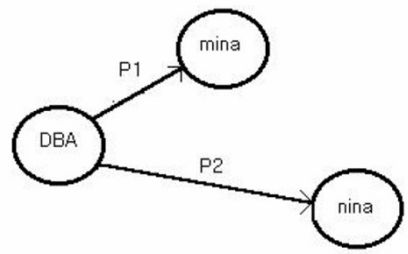
- گراف مجوز این مثال به صورت زیر خواهد بود:



- P1: SELECT ON Student WITH GRANT OPTION;
- P2: SELECT ON Student
- با استفاده از گراف مجوز کاربر، اگر مسیری از DBA به U وجود داشته باشد به طوری که تمام اضلاع این مسیر بر چسب P را داشته باشند، آنگاه U امتیاز P را دارد. در مثال بالا mina، nina، tina می توانند روی جدول Student، SELECT کنند اما فقط tina و mina حق واگذاری به غیر را دارند یعنی حق واگذاری به غیر به nina واگذار نشده است.

امنیت در بانک اطلاعات

- اگر برای پس گرفتن امتیاز DBA دستور زیر را اجرا کند:
- REVOKE SELECT ON Student FROM tina;
- گراف مجوز به صورت زیر در می آید:



- اگر به دستور فوق قید CASCADE اضافه شود آنگاه فقط mina می تواند از جدول Student انتخاب SELECT کند.

امنیت در بانک اطلاعات

- مدیریت مجوزها:
- روش های مدیریت واگذاری و پس گرفتن امتیاز کاربران به کاربران دیگر:
- متمرکز(centralized): کاربر یا گروهی از کاربران حق واگذاری یا پس گرفتن امتیاز خود به دیگران را دارند.
- مالکیت (ownership): فقط مالک شیء (کسی که آن را ایجاد کرده است) می تواند مجوز دسترسی به شیء را به دیگر کاربران واگذار کرده و یا پس بگیرد.
- نامتمرکز (decentralized): ترکیبی از دو روش قبلی است. مالک شیء می تواند حق واگذاری و یا پس گرفتن امتیاز را به کاربران بدهد یا پس بگیرد و آنها به دیگران واگذار کنند یا پس بگیرند.
- یکی از مشکلات این روش رسیدگی به مجوزهای واگذار شده توسط کاربرانی است که امتیاز آنها پس گرفته شده است.

امنیت در بانک اطلاعات

- مسئله اسب های تراوا (trojan horses)
- سیاست های محتاطانه، از کاستی هایی برخوردارند که سبب دسترسی های غیرمجاز کاربران به اطلاعات محرمانه سیستم می شود. یکی از مهمترین این کاستی ها، آسیب پذیری در مقابل اسب های تراوا است.
- از طریق روش اسب های تراوا، یک کاربر غیر مجاز می تواند از طریق کاربران مجاز به اطلاعات محرمانه دسترسی داشته باشد بدون اینکه سوء استفاده او کاملاً واضح و مشخص باشد.
- به عنوان مثال فرض کنید دانشجوی x می خواهد به لیست نمرات استاد y که در جدول grades ذخیره شده است دسترسی داشته باشد. برای این منظور، دانشجو ی x اعمال زیر را انجام می دهد:
- جدول جدیدی به نام sample ایجاد کرده و اجازه عمل insert در این جدول را به استاد y می دهد (استاد y از این موضوع بی اطلاع است)

امنیت در بانک اطلاعات

- سپس برخی توابع DBMS را که استاد y همواره استفاده می کند، چنان تغییر می دهد که در هنگام کار او با سیستم، اطلاعات از جدول grades خوانده شده و در جدول sample کپی شوند (اسب تراوا).
- دانشجو x سپس منتظر کار استاد با سیستم شده و پس از آنکه نمرات از جدول grades به sample کپی شدند، توابع تغییر داده شده را به حالت اول بر می گرداند تا مدیر بانک اطلاعات، از آن مطلع نشود.
- بر خلاف سیاست های سیستم که دسترسی به جدول grades را تنها به استاد y داده بود، کاربر دیگری توانست اطلاعات سیستم را کند. بنابراین نیاز به سیاست های دسترسی امن تری داریم که امنیت کامل دسترسی ها را برقرار کرده و در مقابل حملات امنیتی (از جمله روش اسب های تراوا) در امان باشد.
- مزیت روش محتاطانه انعطاف پذیری است. عیب آن این است که کاربر غیرمجاز ازطریق کاربر مجاز می تواند امتیازاتی را کسب کند. کاربر می تواند امتیاز خود را به کاربر دیگر اعطا کند، (انتشار امتیاز) در این صورت عملیات سلب امتیاز با مشکلاتی مواجه می شود.

امنیت در بانک اطلاعات

- کنترل دسترسی مبتنی بر نقش (Role Based Access Control)
- ابتدا نقش های مطرح در سازمان را تعریف، مجوزهای هر یک از نقش ها تعیین و به آن اختصاص داده می شود. سپس افراد را به نقشهای مربوطه (یا نقشها را به نقشهای دیگر) نسبت می دهیم.
- در صورت تغییر شغل سازمانی به منظور تغییر مجوزهای فرد، کافیست نقش او را عوض کنیم. می توان پرسنل جدید را به راحتی به نقش مربوطه نسبت داد.
- مثال:

```
create role r1
create role r2
grant select on sec to r1
grant update(avg) on stud to r2
grant all privilege on crs to r2
grant r1 to r2
grant r1 to Ali, Reza
grant r2 to Hamid
```


امنیت در بانک اطلاعات

- کنترل دسترسی الزامی (Mandatory Access Control)
- معروفترین مدل کنترل دسترسی الزامی، مدل Bell-LaPadula است. دراین مدل پدیده های سیستم به چهار دسته زیر تقسیم می شوند:
- ۱- شیء (object): پدیده های فعالی که اطلاعات را نگهداری می کنند. (جداول، رکورد، اشیا، دیدها، صفت ها، کلاس ها)
- ۲- فرایند (subject): پدیده هایی که درخواست های دسترسی به اشیا را می دهند. (کاربران و برنامه ها)
- ۳- کلاس های امنیتی (security classes): سطوح امنیتی دسترسی به هر شیء، به صورت زیر تعریف می شوند: خیلی محرمانه (TS)، محرمانه (S)، سری (C)، طبقه بندی نشده (U) و ترتیب آنها به صورت TS>S>C>U است.
- ۴- حساسیت (clearance): حساسیت هر فرایند در رابطه با یک کلاس امنیتی را نمایش می دهد. کلاس دسترسی را به صورت زوج مرتب هایی از سطوح امنیتی و مجموعه گروهها نمایش می دهند (S,GI).

امنیت در بانک اطلاعات

- کلاس دسترسی C_1 پایین تر از کلاس دسترسی C_2 است ($C_1>=C_2$) اگر و فقط اگر سطح امنیت C_1 بزرگتر یا مساوی سطح امنیت C_2 و گروه های C_1 شامل همه گروه های C_2 باشند.
- دو کلاس C_1و C_2 را غیر قابل مقایسه گویند، اگر نه $C_1>=C_2$ و نه $C_2>=C_1$باشد.
- سطح امنیت از کلاس دسترسی یک شئ نشان دهنده حساسیت اطلاعات آن شئ می باشد. همچنین نشان دهنده پتانسیل خرابی است که از یک دسترسی غیر مجاز به اطلاعات می تواند نتیجه شود.
- سطح امنیت کلاس دسترسی یک کاربر نشان دهنده قابل اعتماد بودن کاربر است که اطلاعات حساس را برای کاربران غیرمجاز فاش نمی کند.
- کاربران می توانند تراکنشی با کلاس دسترسی خود یا کمتر از آن به سیستم بدهند. مثلا کاربر یا کلاس (S,0) می تواند تراکنش هایی با کلاس های دسترسی (S,0)، (C,0) و (U,0) به سیستم بدهد.

امنیت در بانک اطلاعات

- مدل Bell-LaPadula دو محدودیت زیر را در تمام دسترسی های به اشیاء بانک اطلاعات قائل است:
- فرایند S اجازه دسترسی خواندن به شی ء O را دارد اگر
- Class(S) >= Class(O)
- به عنوان مثال کاربر X با حساسیت TS می تواند جدولی با حساسیت C را بخواند. اما کاربرY با حساسیت C اجازه خواندن از جدولی با حساسیت TSرا ندارد.
- فرایند S اجازه نوشتن روی شی ء O را دارد اگر
- Class(O) >= Class(S)
- به عنوان مثال کاربر X با حساسیت S اجازه نوشتن در جدولی با حساسیت S یا TS را دارد.
- مثلا یک سروان می تواند اطلاعات فوق سری تولید کند اما نمی تواند به همه اطلاعات فوق سری دسترسی داشته باشد.

امنیت در بانک اطلاعات

- کلاس های دسترسی برای جامعیت (integrity) شامل سطح جامعیت و مجموعه گروه ها است.
- سطح جامعیت برای کاربر نشان دهنده قابل اعتماد بودن کاربر برای ورود، تغییر و یا حذف داده ها است.
- سطح جامعیت برای شی، هم نشان دهنده درجه اطمینانی است که می توان به اطلاعات ذخیره شده در آن داشت و هم نشان دهنده پتانسیل خرابی است که می توان از یک تغییر غیرمجاز اطلاعات نتیجه شود.
- نمونه ای از سطوح امنیت در این حالت می تواند از سه نوع بسیار مهم(C)، مهم (I) و نامشخص (U) باشد.
- در رابطه با جامعیت نیز کنترل دسترسی، طبق دو قانون زیر اعمال می شود:
- فرایند S اجازه خواندن از شی O را دارد اگر
- Class (O) >= Class (S)
- فرایند S اجازه نوشتن برروی شی O را دارد اگر
- Class (O) <= Class (S)

امنیت در بانک اطلاعات

- سیاست های امنیت اجازه جریان داده را فقط از سطوح پایین تر(امنیت) به سطوح بالاتر می دهد، در حالیکه سیاست های جامعیت اجازه جریان داده را فقط از سطوح بالاتر به سطوح پایین تر می دهد.
- اگر نیاز به کنترل هر دو مقوله امنیت و جامیت باشد لازم است که کلاس دسترسی متفاوتی برای امنیت . جامعیت به کار رود.
- ضعف اصلی سیاست های الزامی دشواری کنترل آنهاست، زیرا احتیاج به تعریف استفاده طبقه بندی اشیاء، کاربران و برنامه ها دارند. که ممکن است همیشه میسر نباشد. همچنین دسترسی های تعیین شده فقط براساس طبقه بندی اشیاء و تراکنش های موجود در سیستم است و هیچ امکانی به کاربران برای واگذاری یا پس گرفتن مجوز به دیگر کاربران داده نمی شود.

امنیت در بانک اطلاعات

- رمزنگاری
- یکی دیگر از روش های برقراری امنیت در سیستمهای بانک اطلاعات، رمزنگاری است. این روش امنیت کانال های ارتباطی را برقرار می کند. ایده اصلی رمزنگاری داده ها استفاده از الگوریتم های رمزنگاری و نیز یک کلید رمزنگاری مخصوص مدیر بانک اطلاعات است که به صورت امن نگاه داشته می شود.
- روش های رمز نگاری به دو دسته کلی زیر تقسیم می شوند:
- متقارن: در روش رمزنگاری متقارن، فرستنده و گیرنده از یک کلید سری مشترک برای رمزنگاری و رمزگشایی استفاده می کنند. روش Data Encryption Standard (DES) مثالی از رمزنگاری متقارن است. بدیهی است که برا ی دو طرف ناشناس توافق بر روی یک کلید سری مشترک دشوار و ناامن است. بنابراین این روش مورد استفاده کمتری دارد.

امنیت در بانک اطلاعات

- رمزنگاری
- نامتقارن: روش دیگر، روش رمزنگاری نامتقارن است. در این روش هر فرد دو کلید در اختیار دارد: کلید عمومی که آزادانه منتشر می شود و کلید خصوصی که به صورت خصوصی و کاملاً محرمانه نگهداری می شود.

  در این روش، فرستنده، داده ها را با کلید عمومی رمز کرده و به گیرنده ارسال می کند. داده های رمز شده تنها با کلید خصوصی گیرنده قابل رمزگشایی هستند و از آنجایی که کلید خصوصی هر شخص منحصر به فرد است و به طور کاملاً امن نگاه داشته می شود، شخص دیگری نمی تواند این اطلاعات را رمزگشایی کرده و یا از آنها استفاده نماید.

  این روش از امنیت بالایی برخوردار است و مورد استفاده فراوانی دارد.

بحث های تکمیلی امنیت

- نقش مدیر بانک اطلاعات (DBA)
- مدیر بانک اطلاعات (DBA) نقش بسیار مهمی در تعیین و تبیین سیاست های امنیتی سیستم دارد. معمولاً مدیر بانک اطلاعات دارای حساب ویژه‌ای در DBMS است که با این حساب می‌تواند سیستم را کنترل کرده و امنیت آن را برقرار نماید.
- وظایف مدیر بانک اطلاعات در قبال امنیت سیستم، به شرح زیر است:
- ۱) ایجاد حساب (account) برای کاربران: همه کاربران و گروه ها برای استفاده از بانک اطلاعات نیازمند حساب کاربری و کلمه عبور می باشند. بدیهی است برنامه هایی که با بانک اطلاعات در تعامل هستند نیز باید دارای حساب ویژه ای در سیستمDBMS باشند.

بحث های تکمیلی امنیت

- ۲) کنترل سیاست های الزامی (mandatory): در صورتی که سیستم مدیریت بانک اطلاعات، از روش کنترل الزامی برخوردار باشد مدیر بانک اطلاعات باید کلاس های امنیتی مربوط به اشیاء بانک اطلاعات، کاربران، گروه ها و همچنین روابط بین آنها را مشخص کند.
- ۳) پیگیری اجازه ها و مدیریت کارنامه (log): مسئولیت دیگر مدیربانک اطلاعات، مدیریت و کنترل کارنامه سیستم است. اینکه هر کاربر چه دستوراتی را اجرا کرده و یا به چه داده هایی دسترسی دارد و یا چه دستوراتی به سیستم ارسال کرده است، می تواند در بررسی مشکلات سیستم و یافتن مشکلات امنیتی آن، مؤثر و مفید باشد.

بحث های تکمیلی امنیت

- امنیت در بانک های اطلاعات آماری (statistical databases)
- بانک اطلاعات آماری، بانکی است که فقط به پرسش و پاسخ های آماری پاسخ می دهد. به عنوان مثال اگر یک بانک اطلاعات آماری از اطلاعات دانشجویان نگهداری کنیم، تنها پرسش و پاسخ های آماری از قبیل میانگین نمرات، بالاترین سن، کمترین تعداد واحد ها و ... برای این بانک مجاز خواهد بود و پاسخگویی به پرسش هایی در مورد تک تک دانشجو ها مجاز نخواهد بود. امنیت در چنین بانک اطلاعاتی نیازمند در نظر گرفتن مسایل جدیدی است.
- مشکل اساسی بانک های اطلاعات آماری عمدتاً به این صورت است که کاربر با ارسال چند پرسش آماری بتواند اطلاعات محرمانه ای در مورد تک تک اعضا و داده های آن به دست آورد.
- مثلاً با سناریوی زیر، یک کاربر می تواند حقوق مسن ترین و جوان ترین استاد را تشخیص دهد:

بحث های تکمیلی امنیت

- پرسش اول: چند استاد وجود دارد که سن آنها از X بزرگتر باشد؟ کاربر، مقدار X را مرتباً عوض می کند تا پاسخ دریافتی از سیستم عدد یک باشد.
- پرسش دوم: بیشترین حقوق افرادی که سنی بالاتر از X دارند چند است؟ جواب این پرسش حقوق پیرترین فرد دانشگاه را مشخص می کند. به همین ترتیب می توان حقوق جوانترین استاد دانشگاه را پیدا کرد.
- عمده ترین روش های حل مشکل فوق این است که:

۱- تنها به پرسش هایی پاسخ داده شود که برای پاسخگویی آنها نیاز به حداقل N سطر از جداول باشد. با در نظر گرفتن عدد مناسبی برای N (بسته به سیستم) می توان جلوی مشکلات فوق را تا حدی گرفت. مثلاً با در نظر گرفتن N>1 در مثال قبل می توان جلوی پاسخ دهی به سؤال دوم مبنی بر اینکه " بیشترین حقوق افراد با سن بالاتر از X چند است؟ " را گرفت.

۲- محدودیتی برای اشتراک ردیف هایی که در پرسش های متوالی که توسط یک شخص داده می شود در نظر بگیریم.

۳- تعداد پرسشهای یک کاربر را محدود کرد.

- با اعمال هر۳ شرط هنوز امنیت در سیستم برقرار نیست زیرا دو یا چند کاربر می توانند تبانی کنند. در عمل از ترکیبی از سه روش فوق و کنترل log سیستم استفاده می شود.

بانک اطلاعات نامتمرکز (توزیع شده)

- کاربردهای کنونی جهان در بانک اطلاعات اکثراً نامتمرکز هستند.
- فواید بانک اطلاعات نامتمرکز:
- فراهم نمودن امکان اشتراک منابع
- قابلیت اطمینان بالاتر
- عدم وابستگی به یک سایت و توزیع امکانات
- بالا بودن سطح امنیت

- بانک اطلاعات نامتمرکز مجموعه ای از چند بانک اطلاعات، منطقاً مرتبط به هم می باشد که روی یک شبکه کامپیوتری توزیع شده اند؛ سیستم مدیریت بانک اطلاعات نامتمرکز (DDBMS یا Distributed DBMS) نرم افزاری است که امکان مدیریت بانک های اطلاعات نامتمرکز را فراهم می نماید و نامتمرکز بودن سیستم را از دید کاربران پنهان می سازد.

بانک اطلاعات نامتمرکز (توزیع شده)

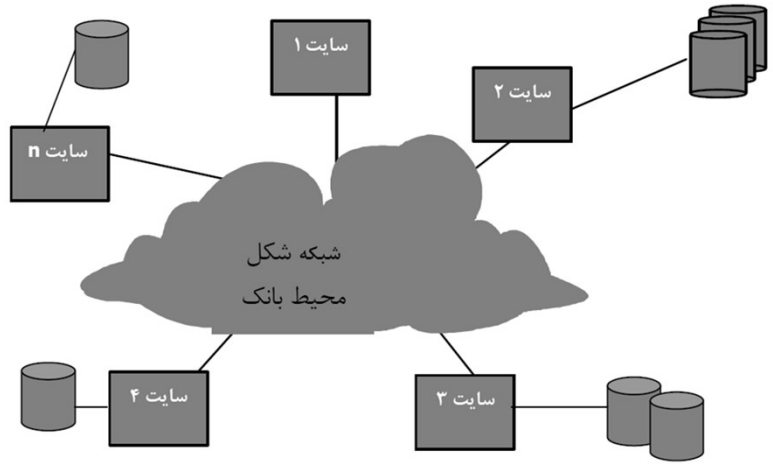
- داده های بانک اطلاعات نامتمرکز توزیع شده اند. جهت توزیع کردن:
  ۱ - داده و ۲ - پردازش مورد نیاز است.
- سیستمی که در آن کل اطلاعات در یک سایت قرار دارد و سایت های دیگر از طریق شبکه از آن بانک اشتفاده می کنند یک بانک نامتمرکز نیست (زیرا اطلاعات توزیع نشده است). چنین سیستمی فقط بانک اطلاعات متمرکز روی شبکه است.
- بانک اطلاعات نامتمرکز در واقع اجتماع «بانک اطلاعات» و «شبکه های کامپیوتری» است.
- تقسیم بندی جغرافیایی:
- LAD یا Local Area Database محلی
- MAD یا Metropolitan Area Database منطقه ای
- WAD یا Wide Area Database گسترده

بانک اطلاعات نامتمرکز (توزیع شده)

- هر چه بر گستردگی جغرافیایی بانک اطلاعات نامتمرکز افزوده می گردد، تعداد بانک های آن کم می شود. به عبارت دیگر، در مقیاس جهانی، تعداد بانک های نامتمرکز محلی بسیار زیاد، تعداد بانک های نامتمرکز منطقه ای نه چندان زیاد و تعداد بانک های نامتمرکز گسترده نسبتاً کم است.
- بانک های محلی از یک یا چند کامپیوتر اصلی و تعدادی پایانه (terminal) مختلف وکامپیوتر شخصی تشکیل شده که از طریق سیم مخصوص به هم وصل شده اند و در درون یک ساختمان یا حداکثر یک مجموعه واقع هستند و مدیریت آنها نسبتا ساده تر است.
- بانک های منطقه ای از طریق خطوط تلفن (عادی و همراه) به هم متصلند. کامپیوترهای پرتابل از درون اتومبیل با استفاده از تلفن همراه به بانک اطلاعات منطقه ای وصل می گردند.
- بانک های گسترده نیز از کامپیوترها و پایانه های متفاوتی تشکیل شده اند که ممکن است در شبکه های محلی و منطقه ای فعال باشند و به یک شبکه سراسری کشوری یا جهانی نیز متصل گردند.

بانک اطلاعات نامتمرکز (توزیع شده)

- محیط بانک اطلاعات نامتمرکز



بانک اطلاعات نامتمرکز (توزیع شده)

- مثال: بانک اطلاعات آموزش کشور
- این بانک حداقل شامل بخش های زیر است.

۱. هر یک از دانشگاه ها یک شبکه محلی دارند.

۲. مجموعه ای از دانشگاه های هر شهر یا منطقه در یک شبکه منطقه ای قرار می گیرند.

۳. هر یک از ادارات کل آموزش و پرورش شهرها یا مناطق، یک شبکه محلی دارند که مدارس را نیز شامل می شود.

۴. شبکه های محلی آموزش و پرورش به همان شبکه منطقه ای منطقه خود که دانشگاه ها را در بر می گیرد متصلند.

بانک اطلاعات نامتمرکز (توزیع شده)

بانک اطلاعات نامتمرکز (توزیع شده)

۵. هر یک از وزارتخانه های آموزش عالی، آموزش و پرورش، بهداشت و آموزش پزشکی دارای شبکه های محلی و شبکه های منطقه ای هستند. مثلاً در وزارت علوم، هر یک از معاونت های اداری و مالی، دانشجوئی، آموزشی، پژوهشی و نیز سازمان سنجش و آموزش کشور به طور جداگانه شبکه ای محلی تشکیل می دهند و روی هم به شبکه شهر تهران متصلند.

۶. سایر دانشگاه ها و موسسات آموزش عالی شبکه های محلی و منطقه ای ویژه خود را دارند.

۷. از اتصال شبکه های منطقه ای به یکدیگر شبکه سراسری آموزش کشور تشکیل می شود.

مفاهیم و اهداف

ناهمگون heterogeneous و همگون homogeneous

- ناهمگون heterogeneous و همگون homogeneous
- بانک های اطلاعات در ساده ترین شکل کاملاً همگون هستند و در پیچیده ترین شکل کاملاً ناهمگون می باشند.
- در بانک همگون، تمام سایتها دارای ویژگی های سخت افزاری و نرم افزاری یکسان می باشند و برای پردازش درخواست کاربر با دیگران مشارکت دارند.
- در بانک ناهمگون، سایتهای مختلف می توانند از نرم افزار، سخت افزار و یا شمای متمایزی استفاده و امکاناتی را برای مشارکت در انجام کارهای سایتهای دیگر ارائه نمایند.
- نکته: هر چقدر درجه ناهمگونی بالاتر باشد، کار سخت تر می باشد.

مفاهیم و اهداف

ناهمگون heterogeneous و همگون homogeneous

- در همگونی از جهت نرم افزار دو جنبه اصلی عبارتند از: سیستم عامل و سیستم مدیریت بانک اطلاعات.
- ساده ترین شکل زمانی ممکن است که همه بانک ها همگونی داشته باشند (strict homogeneity) یعنی هر کدام نسخه ای از نرم افزار یکسان را روی کامپیوتر سازگار اجرا می کنند.
- حالت دیگر حالتی است که همه چیز یکسان باشد به جز سیستم مدیریت بانک اطلاعات که از یک مدل خاص هستند ولی متفاوت باشند. مثلا همگی از نوع رابطه ای باشند ولی یکی oracle و دیگری DB2 و ... باشد.
- پیچیده ترین حالت زمانی است که همه چیز حتی سیستم مدیریت بانک اطلاعات متفاوت باشد. مثلا یک سایت O_2 (شی گرا) اجرا کند، دیگری oracle و .... در این حالت که درحال حاضر در جهان دارد، همه سیستم ها از واسط (interface) یکسانی پیروی می کنندو قادرند با یکدیگر ارتباط برقرار کنند.

مفاهیم و اهداف

توزیع داده ها- تکرار داده ها

- توزیع distribution داده ها
- شامل دو بخش fragmentation (شکستن بانک اطلاعات) و allocation (تخصیص تکه ها به سایت ها) است.
- ابتدا داده ها را تقسیم کنیم سپس تصمیم بگیریم هر بخش را روی کدام یک ازکامپیوترها قرار دهیم که ممکن است به صورت top-down باشد؛ یعنی ابتدا بانک را در اختیار داشته باشیم و سپس تصمیم بگیریم داده ها را در کجا قرار دهیم یا به صورت bottom-up باشد؛ یعنی ابتدا تعدادی بانک نامتمرکز موجود هستند و سعی کنیم از روی آنها یک سیستم بانک متمرکز بسازیم. پس باید تعدادی داده اضافی که metadata گفته می شود در این سایت ها قرار دهیم.
- تکرار داده ها (replication)
- برای سرعت بالاتر در بازیابی اطلاعات، قابلیت اطمینان، تحمل پذیری خطاو .. نیاز به نگهداری داده در بیش از یک سایت است. این کار تکرار داده نام دارد.
- تکرار جزئی (partial): از هر داده تعدادی نسخه در جاهای مختلف وجود دارد.
- تکرار کامل (full): از این داده روی تمام سایت ها یک نسخه وجود دارد.

مفاهیم و اهداف

تکرار داده ها (replication)

- مهمترین مزایای تکرار داده ها عبارتند از:
- در دسترس بودن (availability): خرابی یک سایت که داده مورد نظر را در اختیاردارد، منجر به عدم در دسترس بودن آن داده نمی شود.
- موازی سازی (parallelism): پرس وجوهایی که می خواهند از یک داده (مثلاً یک رابطه) استفاده کنند، می توانند به طور موازی روی چندین سایت انجام شوند.
- کاهش میزان انتقال داده ها: داده هایی که در یک سایت قرار دارند، به عنوان داده های محلی آن سایت تلقی می شوند و لازم نیست از سایت دیگری به آنجا منتقل شوند.
- مهمترین معایب تکرار داده ها عبارتند از:
- افزایش هزینه به روز نگهداشتن تکرارها: هزینه: سرعت و فضای حافظه کامپیوتر
- افزایش پیچیدگی کنترل همروندی

مفاهیم و اهداف

شفافیت (transparency)

- شفافیت (transparency)
- شفافیت به این معنی است که پیچیدگی ها و جنبه های مختلف مربوط به توزیع جغرافیایی بانک اطلاعات حتی الامکان از دید کاربران مخفی باشد و آنها را آزار ندهد. وضع ایده آل این است که کاربران در بازیابی اطلاعات (پرس وجو) هیچ کاری با نامتمرکز بودن بانک نداشته باشند، یعنی از دید آنها با بانک متمرکز تفاوتی نداشته باشد. شفافیت جنبه های مختلفی دارد:
- شفافیت خرابی (failure transparency):
  جایگزینی خودکار بانکی با بانک دیگری که دچار خرابی می شود.
- شفافیت کارایی (performance transparency):
  انتخاب خودکار کم هزینه ترین و سریعترین راه حل.
- شفافیت ناهمگونی (heterogeneity transparency):
  حل خودکار تفاوت ها.

مفاهیم و اهداف

شفافیت (transparency)

- شفافیت تکرار داده ها (replication transparency):
  تکرار داده در سایت های مختلف باعث آزار کاربر نشود و کاربر تصور کند با یک نسخه از داده سروکار دارد.
- شفافیت تقسیم (fragmentation transparency):
  کاربر نباید درگیر این مسئله شودکه آیا داده مورد نظر تقسیم و توزیع شده است یا خیر. بلکه باید تصور کند که هر داده ای که او نیاز دارد، داده ای محلی (local data) در همان سایت می باشد. سیستم مدیریت بانک اطلاعات نامتمرکز باید یک بانک اطلاعات متمرکز را برای کاربر شبیه سازی کند. جزئیات مربوط به توزیع بانک اطلاعات برعهده سیستم مدیریت بانک اطلاعات نامتمرکز می باشد و خارج از وظایف کاربر می باشد.
- شفافیت محل داده ها (location transparency):
  کاربران تصور کنند داده ها در یکجا هستند.

مفاهیم و اهداف

قابلیت اطمینان- بهبود کارایی

- قابلیت اطمینان (reliability)
- منظور از قابلیت اطمینان توانایی سیستم در پاسخ به درخواستها در صورت وقوع خرابی می باشد. با داشتن عناصر (سخت افزاری، نرم افزاری، داده و ...) متعدد در سایت های گوناگون، این امکان بالا می رود و اگر سایتی از کار افتاد سایت دیگری را می توان جایگزین نمود.

- بهبود کارایی
- بانک اطلاعات نامتمرکز باید سرعت عمل و کارآیی خود را بالا ببرد. بانک اطلاعات متمرکز در مقایسه با بانک اطلاعات نامتمرکزی که همان امکانات و داده ها را دارد سرعت کمتری خواهد داشت.

مفاهیم و اهداف

بهبود کارایی

- مهمترین جنبه هایی که به بهبود کارایی در بانک های اطلاعات نامتمرکز کمک می کنند عبارتند از:
- موازی سازی
  - هم درون یک پرس وجو و هم بین چند پرس و جو می توان موازی سازی انجام داد. مثلاً یک پرس وجو را به چند زیرپرس و جو تقسیم کرده، آنها را به طور موازی در سایت های مختلف اجرا می نماییم و نتیجه آنها را ادغام کنیم.

- محلی کردن داده ها (data localization)
  - محلی کردن داده ها بهبود کارآیی را در بر دارد.

مفاهیم و اهداف

بهبود کارایی

- محلی کردن داده ها
- در بانک اطلاعات نامتمرکز حتی الامکان داده ها در نزد صاحب آنها نگهداری می شوند.
- نزدیک کردن داده ها به محل استفاده آنها سه مزیت عمده دارد:

۱. از آنجا که هر سایت فقط به بخشی از بانک دسترسی دارد، حجم عملیات در آن کاهش می یابد.

۲. تأخیر مربوط به دسترسی از راه دور (انتقال داده ها) کم می شود.

۳. هر سایت مسئول داده های خود است، بنابراین به روزرسانی و کنترل آنها ساده تر و قابل اعتمادتر می باشد.

مفاهیم و اهداف

توسعه ساده تر سیستم

- توسعه ساده تر سیستم
- اگر داده ها و ابعاد بانک اطلاعات گسترش پیدا کند دو حالت زیر ممکن می باشد:
- بانک اطلاعات متمرکز: از کامپیوترها، امکانات و نرم افزارهای قویتر استفاده کنیم و پردازش های قبلی را تبدیل کنیم و یک سیستم جدید راه اندازی کنیم.
- بانک اطلاعات نامتمرکز: کافی است تعدادی کامپیوتر به سیستم اضافه کنیم و قدرت پردازش داده ها را افزایش دهیم. بانک اطلاعات نامتمرکز توسعه پذیرتر است.

مفاهیم و اهداف

خودمختار ی محلی (local autonomy)

- خودمختار ی محلی (local autonomy)
- بانک های اطلاعاتی که در یک شبکه با یکدیگر همکاری می کنند، تمایل به از دست دادن آزادی عمل ندارند.
- به طور مثال:
- اگر چند بانک اطلاعات با یکدیگر تعامل داشته باشند، هیچ کدام از این بانک ها حاضر نیستند عملیات داخلی و امنیت داده های خود را زیر سؤال ببرند و تمایل دارند کارها مانند گذشته انجام بگیرد و امکان کار سراسری نیز برای آنها فراهم شود.
- ممکن است مایل نباشند دیگران داده های آنها را تغییر دهند یا همهٔ وقت کامپیوتر آنها را بگیرند و صرف کارهای سراسری (global) کنند. رسیدن به همه این هدف ها غیرممکن است و بانک هایی که عضو یک بانک نامتمرکز می شوند باید طبق ضوابط و مقرراتی امکانات خود را در اختیار دیگران بگذارند.

بانک اطلاعات نامتمرکز (توزیع شده)

مفاهیم و اهداف

خودمختار ی محلی (local autonomy)

- مهمترین اصولی که بانک ها معمولاً برای خود محفوظ نگه می دارند عبارتند از:
  - مالکیت و مدیریت داخلی داده ها.
  - عدم وابستگی به بانک های دیگر در انجام امور داخلی.
  - کنترل داخلی عملیات و آزادی در اعمال روش های دلخواه امنیت و جامعیت.
- نتیجه: وجود یک سایت مرکزی و وابستگی بانک های دیگر به آن مطرود است زیرا در صورت خرابی سایت مرکزی سایر بانکها نیز مختل می مانند.

مفاهیم و اهداف

خودمختار ی محلی (local autonomy)

- در ذیل برخی از نیازهای خود مختاری معرفی شده اند:
- عملکرد سیستم های مدیریت بانک اطلاعات تحت تأثیر سایت های دیگر قرار نگیرد.
- نحوه پردازش پرس وجوهای داخلی نباید تحت تأثیر اجرای پرس وجو های سراسری که به چندین بانک اطلاعات دسترسی دارند قرار گیرد.
- عملکرد سیستم نباید با اضافه یا کم شدن یک سایت تغییر کند.

بانک اطلاعات نامتمرکز (توزیع شده)

مفاهیم و اهداف

خودمختار ی محلی (local autonomy)

- عمده ترین ابعاد خودمختاری عبارتند از:

۱. خود مختاری در طراحی: هیچ گونه تغییری در نرم افزارهای بانک های عضو لازم نیست داده شود.

۲. خود مختار ی در اجرا: هر عضو، کنترل صددرصد روی تراکنش هایی که در آن بانک اجرا می شود دارد. یک عضو می تواند، در شرایط اضطراری، زیر تراکنش ها را که از تراکنش های سراسری می آیند، در هر وضعیتی که باشند (حتی در شرف نهایی شدن) ساقط کند و تراکنش های داخلی خود را بدون هیچ گونه محدودیتی به اجرا در آورد.

۳. خود مختاری در ارتباط: بانک های عضو هیچ گونه اطلاعاتی در مورد کنترل های اعمال شده روی تراکنش ها در اختیار یکدیگر یا نرم افزار سراسری، قرار نمی دهند. به عبارت دیگر، کنترل تراکنش سراسری و زیر تراکنش های آن باید مستقلاً توسط نرم افزار سراسری انجام شود و از نرم افزارهای داخلی انتظار همکاری نداشته باشد.

مفاهیم و اهداف

تداوم فعالیت

- تداوم فعالیت.
- بانک های عضو نمی توانند به دلخواه سایت خود را تعطیل کنند یا به دلایلی غیر فعال نمایند، زیرا سایت های دیگر باید بتوانند به استفاده های مجاز ادامه دهند. این مورد نمی تواند بخشی از خودمختاری سیستم به حساب آید. اصولاً مشارکت در یک بانک اطلاعات نامتمرکز، با هدف دست یابی به اطلاعات وسیع تر و پشتیبانی دیگران صورت میگیرد و سایت ها باید بتوانند روی همکاری دیگران حساب کنند.
- سرعت قابل قبول، به روز درآوردن داده ها در سایت های دیگر و ارائه امکانات معمول بانک اطلاعات (مدیریت تراکنش و غیره) از جمله اصول این مشارکت به حساب می آیند.

مفاهیم و اهداف

- معماری سیستم های بانک اطلاعات نامتمرکز
- معماری یک سیستم یعنی ساختار آن از نظر اجزا (component) تشکیل دهنده، عملکرد هر یک از اجزا و ارتباط و تعامل آنها با یکدیگر.
- بانک اطلاعات نامتمرکز، مجموعه ای از بانک های اطلاعات مجزا است که با یکدیگر ارتباط برقرار می کنند و از طریق شبکه های کامپیوتری به هم متصل می شوند از لحاظ معماری به چند دسته مختلف تقسیم می شوند.
- عمده ترین ابعاد تأثیرگذار بر معماری:
- توزیع شدگی.
- ناهمگونی (heterogeneity).
- خود مختاری محلی (local autonomy).
- استفاده از بانک دانش(knowledge base).
- این موارد باعث می شود بانک نامتمرکز کاملاً متفاوت داشته باشیم. بانک هایی که خود مختار نیستند، نسبت به بانک های خود مختار ساده تر هستند.

مفاهیم و اهداف

- مهمترین معماری های بانک اطلاعات نامتمرکز عبارتند از:
- ۱. سیستم های بانک اطلاعات نامتمرکز ساده (ordinary distributed database systems)
- ۲. سیستم ها ی بانک اطلاعات چندگانه (multi-database systems)
- ۳. سیستم ها ی اطلاع رسانی همیار (cooperative information systems)

مفاهیم و اهداف

- علاوه بر تفاوت های ساختاری و مفهومی، معماری های مختلف بانک اطلاعات نامتمرکز در مدیریت تراکنش ها نیز تفاوت های ریشه ای و اساسی دارند:
- معماری های ساده روش هایی چون قفل گذاری دومرحله ای را تعمیم میدهند. دراین گونه معماری‌ها، وابستگی به یک یا چند سایت مرکزی همواره وجود دارد.
- معماری هایی که به خودمختاری محلی احترام می گذارند، باید به راه حل های جدیدی بیندیشند، زیرا سایت مرکزی وجود ندارد و نیز مدیران تراکنش محلی نمی توانند اطلاعات خود را (مثل قفل گذاری داده ها) در اختیار مدیر تراکنش سراسری و یا سایت های دیگر قرار دهند.
- معماری های پیچیده تر (سیستم های اطلاع رسانی همیار) که بر اساس بانک دانش بنا می شوند، باید از این هم فراتر روند و تراکنش های طولانی مطرح می شوند و نمی توانند قفل ها را در این مدت طولانی در اختیار بگیرند. خواهیم دید که نمیتوان تراکنش ها را تحت چهارچوب ACID اجرا کرد و چهارچوب ACID کارآیی خود را ندارد.

مفاهیم و اهداف

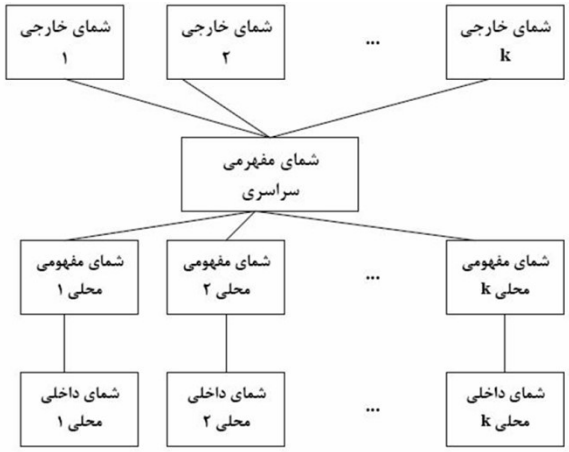
- سیستم های بانک اطلاعات نامتمرکز ساده
- اولین و ساده ترین معماری بانک اطلاعات نامتمرکز سیستمهای بانک اطلاعات نامتمرکز ساده هستند. مشخصه منحصر به فرد این سیستم ها، پیروی از یک شِمای مفهومی سراسری GCS یا Global Conceptual Schema است.
- از قبل بانک اطلاعاتی وجود ندارد و در ابتدا تعدادی بانک اطلاعات نامتمرکز ایجاد می کنیم و برای کاربردی که داریم، یک شمای مفهومی سراسری طراحی می کنیم.
- در هر یک از سایت ها بر مبنای شمای مفهومی سراسری یک شمای مفهومی محلی LCS یا Local Conceptual Schema درست می کنیم.
- طبیعی است که شمای مفهومی یک دانشگاه عام با یک دانشگاه صنعتی متفاوت خواهد شد.

مفاهیم و اهداف

- از روی شمای مفهومی محلی هر سایت، شمای داخلی محلی Local (LIS) Internal Schema آن ساخته می شود. این شما مربوط به نحوه ذخیره سازی فیزیکی هر یک از تکه ها (fragment) و تکرارهای (replica) داده های آن سایت است.
- بر مبنای شمای مفهومی، شمای خارجی محلی Local External (LES) Schema می سازیم. دسترسی هر کاربر (برنامه کاربردی) با استفاده از شمای خارجی محلی او انجام خواهد شد که از شمای خارجی سراسری به دست می آید زیرا کاربران هر سایت می توانند به داده های سایتهای دیگر نیز دسترسی داشته باشند.
- viewهای مورد نیاز را برای کاربران تعریف می‌کنیم. کاربران،کاربران سراسری هستند، یعنی شمای خارجی بر مبنای شمای سراسری می تواند تعریف شود.

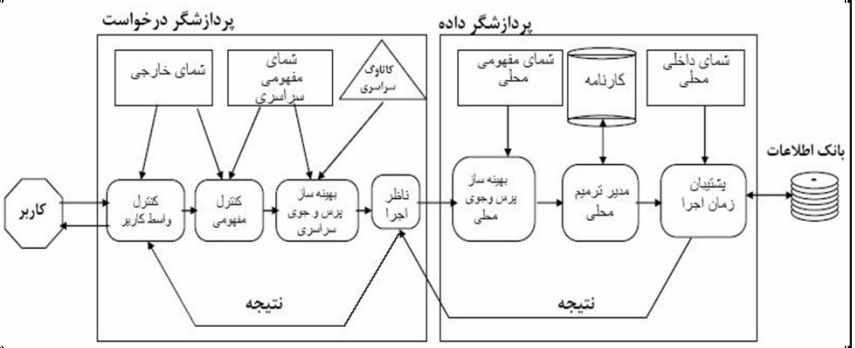
مفاهیم و اهداف

معمار ی سیستم های بانک اطلاعات نامتمرکز ساده (از نظر داده ای)



مفاهیم و اهداف

اجزای یک سیستم بانک اطلاعات نامتمرکز ساده (از نظر داده ای)



❖ البته این تقسیم بندی به واحدهای پردازشگر درخواست و پردازشگر داده تنها از جنبه ساختاری است و هیچ الزامی ندارد که این دو در سایتهای جداگانه ای قرار داشته باشند.

مفاهیم و اهداف

- اجزای واحد پردازشگر درخواست:
- کنترل واسط کاربر user interface handler
- دستورات کاربر را تقسیم نموده و داده هایی که باید به عنوان جواب به کاربر ارسال شوند را فرم دهی می کند.
- کنترل مفهومی semantic control
- براساس قیود جامعیت و مجوزهای دسترسی تعریف شده روی شمای مفهومی سراسری چک می کند که آیا پرس و جوی کاربر قابل پردازش است یا خیر.
- بهینه‌سازوتجزیه‌کننده پرس‌وجوی سراسریglobal query optimizer& decomposer
- استراتژی بهینه اجرای پرس و جوی سراسری را یافته و آنرا (با کمک شمای مفهومی محلی و سراسری به عنوان کاتالوگ سراسری) به پرس و جوی محلی تبدیل می کند.

مفاهیم و اهداف

- اجزای واحد پردازشگر درخواست:
- ناظر اجرا execution monitor
- این واحد که مدیر تراکنش نامتمرکز نامیده می شود وظیفه هماهنگی اجرای درخواستهای توزیع شده کاربر را برعهده دارد. در اجرای پرس و جوها به صورت توزیع شده، نظارت اجرا در سایتهای مختلف و ارتباط با دیگر سایتها توسط این واحد انجام می گیرد.
- کاتالوگ سراسری global catalogue
- این کاتالوگ شامل اطلاعاتی به ویژه در مورد مکان بخش های داده می باشدو به عبارتی خودش یک بانک اطلاعات حاوی دادگان (metadata) اضافی نسبت به کاتالوگ اصلی سیستم می باشد. این کاتالوگ ممکن است سراسری باشد یا محلی برای همان سایت. از نظر توزیع می تواند در یک سایت متمرکز باشد یا روی چندین سایت توزیع شود. همچنین ممکن است فقط یک نسخه از آن داشته باشیم یا اینکه چندین کپی از آن موجود باشد. انتخاب این موارد بستگی به شرایط طراحی بانک اطلاعات نامتمرکز دارد.

مفاهیم و اهداف

- اجزای واحد پردازشگر داده:
- بهینه ساز پرس و جوی محلی local query optimizer
- مسئول بهترین مسیر برای دسترسی به داده ها
- مدیر ترمیم محلی local recovery manager
- مسئول تضمین صحت و جامعیت بانک اطلاعات درصورت وقوع خرابی
- پشتیبان زمان اجرا run-time support
- دسترسی فیزیکی به بانک اطلاعات براساس دستورات سطح پایین تولید شده توسط واحد بهینه ساز پرس و جو از طریق این بخش صورت می گیرد. این واحد به عنوان واسطی برای سیستم عامل بوده و شامل مدیر بافر (cache) بانک اطلاعات نیز می باشد.

بانک اطلاعات نامتمرکز (توزیع شده)
سیستم های بانک اطلاعات نامتمرکز ساده

مفاهیم و اهداف

- نکته: سیستم هایی که به سیستم های چند مشتری/کارگزار معروف هستند، از این نوع می باشند.
- سیستم های چند مشتری/کارگزار:
- نوع گسترش یافته سیستم های بانک اطلاعات نامتمرکز ساده، سیستم های سیستم های بانک اطلاعات چند مشتری/کارگزار هستند.
- در این سیستم ها داده ها در چند کارگزار پخش شده اند و تعدادی مشتری به هر کدام دسترسی دارند.
- در این گونه سیستمها عملکرد سیستم به دو بخش کلی در سایت مشتری و سایت کارگزار تفکیک می شود.
- اغلب کارهای مدیریت داده مثل پردازش و بهینه سازی پرس و جو و مدیریت تراکنش ها توسط کارگزار صورت می گیرد و در سمت مربوط به مشتری فقط واسط کاربر و برنامه کاربردی مربوطه قرار دارد. فعالیتهای ارتباطی بین مشتری و کارگزار به اشتراک گذاشته می شود.

مفاهیم و اهداف

- دو رویکرد کلی زیر وجود دارد:
- هرمشتری فقط یک کارگزار خاص به نام کارگزار خانگی (home server) داشته باشد که درصورت نیاز، ارتباط مشتری با سایر کارگزارها را برقرار کند (Light client).
- هر مشتری مستقیما به کارگزار مناسب درخواست مورد نظرش وصل و مدیریت شود (Heavy client).

مفاهیم و اهداف

- سیستم های بانک اطلاعات چندگانه (multidatabase)
- ویژگی منحصر به فرد بانک اطلاعات چندگانه این است که عضوهای آن هم خود مختار و هم ناهمگون می باشند.
- هر عضو به دو نوع نیاز پاسخ می گوید:

1. پرس و جوها یا تراکنش های داخلی (local) که هیچ ارتباطی با مسائل نامتمرکز ندارند و کاربران آنها نوعاً از وجود سیستم نامتمرکز بی اطلاعند. در هر سایت، یک سیستم مدیریت بانک اطلاعات داخلی (local DBMS) مسئول پاسخگویی به این پرس وجوهاست. این سیستم ارتباطی با سیستم سراسری و تراکنش های سراسری ندارد. مشکلی که پیش می آید، تداخل زیرتراکنش های (sub transactions) سراسری با تراکنش های داخلی مخصوصا در زمینه کنترل همروندی می باشد.

مفاهیم و اهداف

2. پرس وجوها یا تراکنش های سراسری که هدف اصلی سیستم multidatabase هستند. پرس و جو ها با استفاده از داده های multidatabase که در سایت های داخلی آنها پراکنده هستند اجرا می شوند.

برای پاسخگویی به این نوع پرس وجو ها یا تراکنش ها، سیستم نرم افزاری قدرتمندی لازم است که ضمن احترام به خود مختاری سایت ها، یک سیستم کلی نامتمرکز که شفافیت های لازم را دارد به وجود می آورد.

کاربران سراسری با استفاده از این سیستم نامتمرکز باید قادر باشند به سادگی پرس وجو ها و تراکنش های خود را مطرح و اجرا کنند.

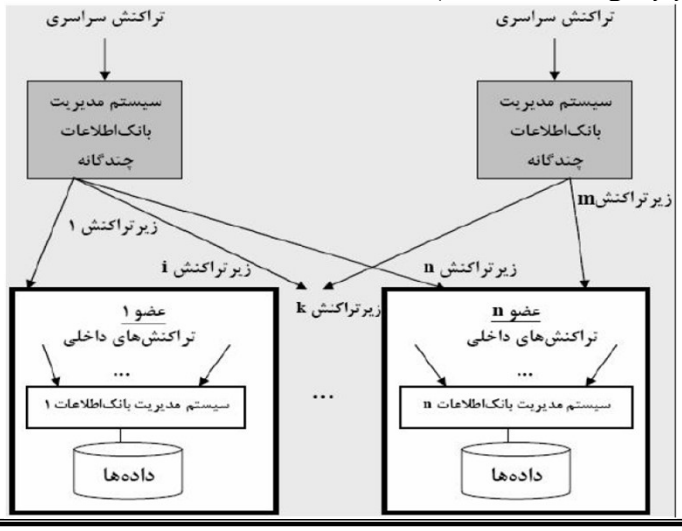
سیستم نرم افزاری سراسری باید تراکنش ها را ابتدا تجزیه کند و تعدادی زیرتراکنش بسازد که زیرتراکنش ها در سایت های عضو، همانند یا مشابه تراکنش های داخلی آنها اجرا می شوند و نتیجه آنها برمی گردد.

مفاهیم و اهداف

- به دلیل اینکه تمام سایت ها و بانک اطلاعات عضو به صورت همانندی نگریسته می شوند، از سیستم بانک اطلاعات سراسری که سیستم مدیریت بانک اطلاعات چندگانه نامیده می شود یک نسخه در هر یک از بانک های عضو قرار می دهیم. این نسخه مطابق با امکانات و وضعیت موجود هر یک از سایت ها تنظیم شده و کاربران می توانند با استفاده از این سیستم، پرس وجوها و تراکنش های خود را که بخش های مختلف آن در جاهای مختلف اجرا می شود درخواست کنند.
- شکل شمای کلی بانک اطلاعات چندگانه را نما یش می دهد.

مفاهیم و اهداف

❖ تصویر کلی بانک اطلاعات چندگانه



مفاهیم و اهداف

❖ خود مختاری و ناهمگونی به چه دلیل می باشد؟

❖ قبلاً ممکن است هر یک از بانک های عضو وجود داشته اند و برای اهداف متفاوتی طراحی شده اند و در آینده هر کدام در راستای زمان به سمت و سوی مختلفی بروند. مهمترین دلیل مقبولیت عام این مدل، احترام گزاردن به خود مختاری داخلی است، زیرا شرکتها و سازمان های مختلف، ضمن نیاز و علاقه به همکاری سراسری، حاضر نیستند ضرورت های داخلی خود را فدا کنند و احیاناً دچار وقفه یا مشکلات امنیتی شوند و یا تغییرات اساسی در کارها بدهند.

❖ خود مختاری محلی در شمای مفهومی سراسری نمود می یابد. هر یک از بانک های عضو شمای مفهومی منحصر به فرد خود را دارند و هر آنچه که لازم می باشد از روی شماهای مفهومی استخراج می کنیم و یک meta data به وجود می آوریم که بر مبنای آن بتوانیم Queryها را تجزیه کنیم و برای اجرا به سایت‌ها ارسال کنیم. از شمای مفهومی داخلی یک شمای مفهومی سراسری بوجود آوریم که این توانایی را بدهد که پرس وجوها را تجزیه و به بانک ها ارسال کنیم.

مفاهیم و اهداف

- در سیستم های بانک اطلاعات نامتمرکز ساده، شمای مفهومی بانک اطلاعات مربوطه از قبل وجود دارد ولی در سیستم های بانک اطلاعات چندگانه این شما از قبل وجود نداردو با استفاده از شمای مفهومی بانک های اطلاعات محلی که می خواهند بخش هایی را به اشتراک بگذارند ساخته می شود.
- بانک اطلاعات سراسری در این دو سیستم متفاوت است چرا که در بانک اطلاعات نامتمرکز ساده اجتماع تمام بانک های اطلاعات محلی است ولی در بانک چندگانه زیرمجموعه ای است از آن اجتماع (بخش هایی که به عملیات سراسری مربوط می شود).

مفاهیم و اهداف

- طراحی شما در سیستم های بانک اطلاعات نامتمرکز ساده، بالا به پایین و در سیستم هَای بانک اطلاعات چندگانه، پایین به بالا می باشد. در سیستم های بانک اطلاعات نامتمرکز ساده، یک بانک اطلاعات متمرکز را تقسیم و توزیع می نمایند. در سیستم های بانک اطلاعات چندگانه، بانک های اطلاعات مجزای موجود با یکدیگر مشارکت و همکاری می کنند تا یک سیستم بانک اطلاعات واحد و متمرکز را ایجاد نمایند.
- نکته: عدم الزام در داشتن شمای مفهومی سراسری یک مزیت عمده برای بانک اطلاعات چندگانه می باشدَ. به دلیل آنکه در بسیاری از کاربردهای بانک اطلاعات نامتمرکز تعدادی بانک از ابتدا وجود دارند و می خواهند با یکدیگر تعامل کنند و این واقعیت موجود زمان است.
- بانک اطلاعات چندگانه multidatabase با شرایط اکثر شرکت ها و مؤسسات کنونی جهان هماهنگ تر است.
- خود مختاری داخلی نمی تواند صددرصد باشد.

مفاهیم و اهداف

- سیستم مدیریت بانک اطلاعات چندگانه می بایست از مقرراتی پیروی کند تا نیاز عضوهای آن برآورده شود. مهمترین این مقررات را که با نام " اهداف بانک اطلاعات چندگانه " بیان شده اند در اینجا خلاصه می کنیم:
- انتقال اطلاعات از فرمی به فرم دیگر.
- پشتیبانی از زبان واسط واحد برای برنامه ساز ی بانک اطلاعات.
- شفافیت در زمینه های مختلف از قبیل نوع سخت افزار، سیستم عامل، و پروتکل شبکه.
- امکان خواندن و نیز به روز درآوردن داده ها در کل سیستم.
- فراهم آوردن همه امکانات معمول بانک اطلاعات از قبیل تعریف دید(view)، بهینه سازی پرس وجو، مدیریت تراکنش، کنترل همروندی و بازگرد، جامعیت، امنیت، انواع واسط کاربر و غیره.
- فراهم آوردن کارایی (سرعت) نزدیک به سرعت بانک های معمولی.

مفاهیم و اهداف

- سیستمهای اطلاع رسانی همیار (Cooperative Information Systems)
- سیستم های اطلاع رسانی همیار از سایت های مختلف و بانک های اطلاعات ناهمگون و خودمختاری شبیه بانک اطلاعات چندگانه تشکیل شده اند.
- تفاوت:
- در بانک اطلاعات چندگانه از مجموعهٔ conceptual schema داخلی می توان یک conceptual schema سراسری به وجود آورد که بتوان از طریق آن تراکنش ها را تجزیه کرده و بخش های مختلف آن را تشخیص داده و آنها را برای اجرا ارسال نمود.
- در بانک های اطلاعات چندگانه تراکنش ها در چهارچوب ACID حرکت می‌کنند. تراکنش ها قابل تجزیه هستند و به سایت های مختلف ارجاع می شوند و سایت ها هم آنها را اجرا می کنند و نتیجه را می فرستند.

مفاهیم و اهداف

- ممکن است درخواست ها کلی تر باشد. زمانی که وارد سیستم اطلاع رسانی همیار می شود، سیستم تلاش می کند که آنها را تجزیه کند و بخش هایی را که قادر به تجزیه است یا خود اجرا می کند و یا به سایت هایی که می شناسد ارسال می کند. بخش هایی را قادر به تجزیه آنها نیست یا ناشناس باشد، سعی می کند عضوهایی را بیابد که می توانند احتمالاً آن بخش را تجزیه کنند و یا اجرا کنند و بخش های ناشناخته را به آن عضو ها ارسال می کند. این پروسه در آن عضو نیز تکرار می شود و این روند تکرار می شود تا زمانی که همه بخش ها شناخته و اجرا شوند و یا بخش هایی به عنوان بخش های مبهم مشخص شود.
- تراکنش های سیستم های اطلاع رسانی همیار NON ACID هستند یعنی ممکن است بخش هایی از آنها اصلاً اجرا نشود، در حالی که نتیجه بخش های دیگر را به کاربر ارسال می کنیم. این وضعیت به خاصیت یکپارچگی یا Atomicity مربوط می شود.

بانک اطلاعات نامتمرکز (توزیع شده)
سیستمهای اطلاع رسانی همیار

مفاهیم و اهداف

- در این پروسه ممکن است حلقه تکرار اتفاق بیفتد، یعنی سایتی تراکنشی را دریافت و سپس آن را تجزیه می کند و به سایت دیگر ارسال می کند و ادامه پیدا می کند تا اینکه به سایت اول باز می گردد و اگر از آن جلوگیری نکنیم تکرار می شود. پس همراه هر زیر تراکنش شناسه سایت هایی را که از آن عبور کرده ارسال می کنیم تا اگر شناسه تکراری به وجود آمد پاسخ دهیم که این کار قابل انجام نیست.

مفاهیم و اهداف

- دو نگرش در مورد سیستم اطلاع رسانی وجود دارد:
- نگرش مدیریی (مربوط به رشته مدیریت)
- نگرش مهندسی نرم افزار (که از بانک اطلاعات نشئت میگیرد)
- سیستم اطلاع رسانی = {انسان، دستورالعمل، برنامه کاربردی، بانک دانش، بانک اطلاعات، نرم افزار، سخت افزار}
- پس از انسان که مهمترین رکن تشکیل دهنده این سیستم است، سایر عوامل وابستگی ماهوی به بانک اطلاعات دارند. نرم افزار، بانک دانش، و برنامه کاربردی برای استفاده بیشتر و بهتر از بانک اطلاعات به کار گرفته می شوند و دستورالعمل ها در رابطه با استخراج و استفاده از اطلاعات است.
- ناهمگونی بانک های عضو در سیستم های اطلاع رسانی همیار باعث پیچیدگی کل سیستم و عدم وجود پایگاه مشترک برای مراوده و رد و بدل کردن اطلاعات می شود. این کمبود را با استفاده از بانک دانش برطرف می کنیم. بانک های دانش حاوی اطلاعاتی هستند که می توان با استفاده از آنها در کل سیستم به دنبال راه حل مشکل گشت و پاسخ پرس و جوها را به دست آورد.

مفاهیم و اهداف

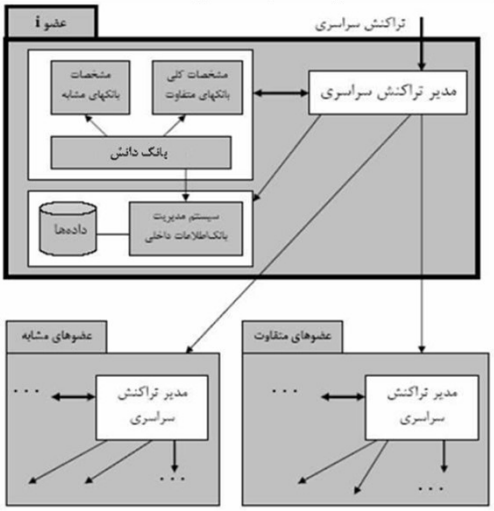
- در معماری همیار همه سایت ها از ساختار واحدی برخوردارند. علاوه بر بانک های عضو در هر سایت دو امکان نرم افزاری دیگر نیز پیش بینی شده است:
- ۱. بانک دانش که علاوه بر بانک اطلاعات داخلی، از دو منبع اطلاعاتی به شرح زیر تغذیه می شود:
- الف) مشخصات بانک های مشابه: در این منبع مشخصات ساختاری و اطلاعات کلی تمام بانک های عضو که در زمینه مشابهی فعالیت می کنند، آمده است. این مشخصات کلی به شناسایی عضوی که داده های لازم برای پاسخگویی به پرس و جوهای مشابه را دارد می انجامید.
- ب) مشخصات بانک های متفاوت: در این منبع مشخصات کلی تمام بانک های عضو که در زمینه های متفاوتی فعالیت می کنند آمده است. وقتی پرس و جوئی مطرح می شود که به زمینه کاری بانک مربوطه ارتباط ندارد، از طریق این منبع می تواند بانک های ذیربط را شناسایی و پرس و جو را به آنها واگذار کند و منتظر نتیجه بماند.

مفاهیم و اهداف

- ❖ ۲. مدیر تراکنش سراسری (GTM یا Global Transaction Manager) که درخواست های سراسری را دریافت، تجزیه و برای اجرا به عضوهای مختلف ارسال می نماید.
- ❖ نکاتی در رابطه با معماری نظاره یادآوری می شود:
- ❖ الف) این سیستم نسبت به بانک اطلاعات چندگانه پیچیده تر است و اهداف آن نیز متفاوت است.
- ❖ ب) داده های یک پرس و جو ممکن است در عضوهای مختلفی به صورت تکراری وجود داشته باشند.
- ❖ ج) مدیر تراکنش سراسری همواره می کوشد تا حد ممکن پرس و جوها را خود پاسخ دهد و در درجه دوم از عضوهای مشابه درخواست کند.
- ❖ د) همه عضوها از وضعیت مشابهی در رابطه با تراکنش های سراسری برخوردارند. یعنی همگی می توانند تراکنش ها را تجزیه، ارسال و پیگیری نمایند.

مفاهیم و اهداف

شکل معماری سیستم (نظام) اطلاع رسانی همیار [نظاره]



مفاهیم و اهداف

- عمده ترین تفاوت های بانک اطلاعات چندگانه با سیستم اطلاع رسانی همیار را می توان به صورت زیر خلاصه کرد:
- ۱ . در بانک اطلاعات چندگانه، همه عضوها در دامنه مشترکی فعالیت می کنند (مثلاً دانشگاه) و در ناهمگونترین شکل ممکن، دامنه های مشابهی را می پوشانند به طوری که می توان از بخش های شماهای مفهومی داخلی آنها که به عملیات سراسری مربوط می شوند، یک شمای مفهومی سراسری به دست آورد و تراکنش ها و پرس وجوها را با استفاده از شمای سراسری تجزیه کرد.
- در مقابل، در سیستم اطلاع رسانی همیار، هر عضو می تواند در دامنه کاملاً متفاوتی فعالیت کند.
- ۲. در بانک اطلاعات چندگانه، همهٔ عضوها "بانک اطلاعات" هستند و DBMS دارند و فعالیت های عمومی بانک های اطلاعات را ارائه می دهند. در مقابل، عضوهای سیستم اطلاع رسانی همیار می توانند حتی یک سیستم معمولی مبتنی بر پرونده یا بانک دانش باشند.

مفاهیم و اهداف

- ۳. درخواست های کاربران در بانک اطلاعات چندگانه از فرمت بسته و مشخصی پیروی می کنند و همواره به سادگی قابل تجزیه و برنامه ریزی هستند. در مقابل، در سیستم اطلاع رسانی همیار، درخواست ها که البته از فرمت مشخصی پیروی می کنند، ممکن است یا اصلاً قابل تجزیه و برنامه ریزی نباشد و یا پس از تجزیه بخش هایی از آنها قابل برنامه ریزی و انجام نباشد و حتی کل درخواست و یا بخش هایی از درخواست قابل شناسایی نباشد.
- ۴. تراکنش ها در بانک اطلاعات چندگانه به صورت تودرتوی بسته (closed nested) هستند، یعنی از خواص ACID پیروی می کنند و سقوط حتی یک زیرتراکنش کوچک باعث سقوط کل مجموعه می گردد. در مقابل، تراکنش ها در سیستم اطلاع رسانی همیار از نوع تودرتوی باز (open nested) هستند، یعنی با سقوط بخش هایی از آنها می توان کل مجموعه را تثبیت (commit) نمود و به انتها رساند.

مفاهیم و اهداف

- ❖ می‌توان امکاناتی فراهم کرد که در تراکنش‌های طولانی قفل حداقل برخی از داده‌ها را قبل از نهایی‌شدن و تثبیت تراکنش بازکنیم و داده‌ها را در اختیار تراکنش‌های دیگر قرار دهیم. XML بسترمناسبی برای پوشش ناهمگونی‌ها است.
- ❖ بانک اطلاعات نظیر - به - نظیر (peer-to-peer)
- ❖ تعدادی بانک اطلاعات هستند که آزادانه هر کدام از آنها با بانک های دیگر تعامل برقرار می کند (به طور انتخابی) و کار های سراسری انجام می دهد.
- ❖ مثال: نظام اطلاع رسانی آموزش کشور
- ❖ بانک اطلاعات آموزش کشور که قبلاً طراحی شد در واقع پاسخگوی نیازها و منطبق بر واقعیت های موجود نیست. اگر در قالب بانک اطلاعات نامتمرکز ساده انجام داده شود مرتباً دستخوش تغییر می شود.
- ❖ در قالب multidatabase معقول تر از حالت قبلی می باشد، زیرا خود مختاری سایت ها را می پذیریم. طراحی و ساخت یک نرم افزار multidatabase که به سادگی بتواند تمام این نیاز ها را پاسخگو باشد مشکل می باشد.