Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство
Ниво студија: Основне академске студије
Модул: Информатика у инжењерству
Предмет: Базе података
Број индекса: 2006/2015

Александар Б. Миладиновић

Трансакције – концепти и примена у SQL окружењу

завршни рад

Комисија за преглед и одбрану:

1. **Проф др Милан Ерић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру завршног рада са темом *“ Трансакције – концепти и примена у SQL окружењу“* **кандидат треба да:**

Проучи литературу и презентира основне концепте обраде трансакција. Завршни рад треба да обухвати уводна разматрања о развоју трансакционих процеса, основне карактеристике окружења у којем се трансакције одвијају и својства трансакционих процеса као и конкретан показни пример. На крају дати закључна разматрања и списак коришћене литературе.

Препоручена литература:

1. Лазаревић Б., …: **Базе података**, ФОН Београд, Београд 2003.
2. P. A. Bernstein, E. Newcomer: **Principles of Transaction Processing**, 2nd Edition, Elsevier, 2009.
3. J. Gray, A. Reuter : **Transaction Processing: Concepts and Techniques**, Elsevier, 1993,

Крагујевац, март 2018.

Ментор:

Prof. Dr Milan D. Erić

Изјава о самосталној изради рада

Изјављујем да сам овај завршни рад урадио самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

2006/2015 Александар Миладиновић

(потпис)

РЕЗИМЕ

Овај рад пружа увид у структуру апликације и система за обраду трансакција. Трансакција је извршење програма који извршава једну административну функцију приступајући дељеној бази података. Трансакција се може извршити онлајн док корисник чека или ван мреже (у batch моду) ако извршење траје дуже него што корисник може да чека резултат. Крајњи корисник захтева извршавање трансакционог програма слањем поруке захтева. Комуникација између страна укључених у пословну трансакцију често се одвија преко рачунарске мреже као што је интернет. Овај рад описује ACID својства трансакција, протокол двофазног извршавања, релацију између система за обраду трансакција са batch процесирањем, системима у реалном времену и системима складишта података. Важно својство трансакција је да су оне изоловане, што значи да се ће извршавање трансакција имати исти ефекат као да су извршене у низу једна за другом. Такво извршавање се назива серијабилно. Најпопуларнији механизам за одржавање серијабилности је закључавање. Закључавање је резервације коју свака трансакција има да би приступила подацима које користи. У раду је посебна пажња посвећена закључавању и протоколима закључавања.

Кључне речи: трансакција, обрада трансакција, ACID својства, серијабилност, закључавање

ABSTRACT

This paper work provides an overview of transaction processing application and system structure. A transaction is the execution of a program that performs an administrative function by accessing a shared database. Transaction ca execute online, while user is waiting, or off-line (in batch mode) if the execution takes longer than user can wait for result. The end user requests the execution of a transaction program by sending request message. Communication between the parties involved in business transaction is often done over a computer network, such as internet. This paper works describes the ACID properties of transaction, two-phase commit protocol and relation of transaction processing with batch processing, real-time and data warehousing systems. An important property of transaction is that they are isolated, which means that the execution of transaction has the same effect as running the transactions serially, one after another. Such an execution is called serializable. The most popular mechanism used to attain serializability is locking. A lock is the reservation that each transaction has to access data it uses. In this paper work special attention is given to the locking and locking protocols.

Key words: transaction, transaction processing, ACID properties, serializability, locking

САДРЖАЈ

1 УВОД

Улога информационих технологија постаје све видљивија у свакодневном животу људи који се полако све више ослањају на рачунаре. Све је више пословних трансакција које су аутоматизоване, на пример, поручивање књиге у онлајн књижари или трансфер новца на банковни рачун на неком другом крају света. Без обзира на тип трансакције која се обавља, желимо да она буде тачна и не желимо да имамо било какве недоумице око њеног исхода.

Трансакције морају да испрате савремене начине пословања и нове технологије и самим тим постају све комплексније. Паметни телефони нам омогућавају да спроведемо трансакцију у било које време на било ком месту. Технолошки напредак, у виду распрострањености интернета или напредак начина третирања пословних догађаја, омогућавају пословању да повећа динамику трансакција кроз инструменте који бележе те догађаје, анализирају повезане податке и активно комуницирају са клијентима да би унапредили корисничко искуство. Да би се прилагодили растућем броју и комплексности трансакција неопходно је континуирано праћење, процена и прилагођавање начина на који ИТ (информационе технологије) подржава обраду трансакција.

Без обзира на то којим се послом бавите неопходно је обезбедите да се ваше трансакције изврше на прави начин и са интегритетом. Погрешан или непотпун резултат трансакције може нарушити поверење купаца, утицати на профит компаније или довести до неочекиваних потраживања, тужби или новчаних казни. Компанија мора да буде у могућности да се ослони на рачунарски систем који је 100% поуздан и гарантује интегритет трансакција у сваком моменту. [1]

2 ОБРАДА ТРАНСАКЦИЈА

2.1 Трансакције

Пословна трансакција је једна интеракција у стварном свету, обично између предузећа и особе или неког другог предузећа, где долази до некакве размене. На пример, то може да укључује новац, производе, информације или услуге. Такође неопходно је водити књиговодство да би се забележило шта је се догодило. Често се књиговодство врши помоћу рачунара због своје скалабилности, поузданости и трошкова. Комуникација између страна укључених у пословну трансакцију често се одвија преко рачунарске мреже као што је интернет. Обрада пословне трансакције помоћу рачунара повезаних рачунарском мрежом назива се обрада трансакција (енгл. transaction processing, TP). Постоји много захтева који се постављају пред рачунарски орјентисану обраду трансакција као што су на пример:

- Пословна трансакција захтева извршавање више операција. На пример размотримо куповину неког производа са онлајн каталога. Једна операција је забележити плаћање а друга обавеза је забележити обавезу испоруке производа купцу. Лако је замислити једноставан програм који извршава овакав посао. Ипак када се скалабилност, поузданост и трошкови додају слици ствари врло брзо постану веома компликоване.
- Обим трансакција и величина базе података додају комплексност и отежавају ефикасност. Сви смо имали искуство чекања због тога што продавац чека терминал на каси да одговори или због тога што је доста времена да учитате веб страницу. Ипак компаније желе да услуже своје кориснике брзо и уз најмање могуће трошкове.
- Да би прилагодили систем високим перформансама, трансакције се морају извршавати истовремено. Неконтролисано истовремено извршавање трансакција може генерисати погрешне одговоре. На рок концерту, када се десетине операција такмиче да резервишу иста преостала седишта врло је важно да само једна особа буде додељена једном седишту. Такође ту је проблем праведности. На пример, Amazon.com је уложио значајне напоре како би осигурао да када је прва хиљада икс бокс конзола изашла на продају да сваки од 50,000 купаца који су се борили за икс бокс има поштену шансу да га добије.
- Ако се трансакција покрене она се мора извршити у целости. У продавници један артикал се мора разменити за новац или се неће продати уопште. Када се догоди грешка, а оне се неизбежно дешавају, важно је избећи делимично завршен посао, као на пример прихватити плаћање а не испоручити производ или обрнуто.
- Свака трансакција треба да врати потврду да је успешно извршена или да врати потврду да није успешно извршена. Ове потврде су врло битне. Ако не добије повратну информацију корисник не зна да ли је потребно да поново пошаље захтев да поново покрене извршење трансакције.
- Систем треба да буде постепено скалабилан. Како посао расте мора се повећавати капацитет за покретања трансакција, пожељно је да то буде постепеном куповном а не заменом постојеће машине већом или још горе преправљањем целе апликације како би се носила са повећаним оптерећењем.

- Када веб страница електронске продавнице (е-продавница) престане са радом она је затворена за пословање. Системи који покрећу трансакције су од виталног значаја за активности пословања које подржавају. Они никада не би требали да се престану да раде.
- Када се трансакција једном евидентира она мора бити трајна и ауторитативна. Ово је често правни захтев као на пример код финансијских трансакција. Трансакције никада не смеју бити изгубљене.
- Систем мора бити у стању да функционише у географски распрострањеном окружењу. Често ово подразумева да је сам систем распрострањен, са машинама на више локација. Понекад је то због правних захтева да систем мора да функционише у земљи у којој се посао обавља. Некада је то неопходно да би се испунили технички захтеви као што су ефикасност, постепена скалабилност или отпорност на отказе система (коришћење бекап система).
- Систем би требало да буде у стању да прилагоди онлајн искуство за сваког корисника на основу његових ранијих начина коришћења. За малопродајног купца треба да идентификује релевантне попусте и огласе и понуди производе прилагођене том кориснику.
- Систем мора да буде у могућности да се повећава предвидљиво и економично да би издржао оптерећење милиона потенцијалних интернет купаца. Не постоји начин да се контролише колико корисника ће се пријавити у исто време или којим трансакцијама ће они изабрати да приступе.
- Систем треба да буде једноставан за управљање. У супротном особље неопходно за управљање системом великих размера може постати превише бројно а самим тим и превише скупо. Комплексан менаџмент система такође повећава шансе за грешке а самим тим и застоје.

Укратко, систем за обраду трансакција мора да се ефикасно избори са великом количином података, да избегне грешке приликом истовремених операција, да избегне стварање делимичних резултата, да расте постепено, да избегне отказе система, не сме никада да изгуби резултате, да омогући географску распрострањеност, да буде прилагодљив и да буде једноставан за управљање [2].

2.1.1 Онлајн трансакције

Онлајн трансакција је извршавање програма који извршава једну административну функцију приступајући дељеној бази података, обично у име једног онлајн корисника. Као и многе системске дефиниције и ова је уопштена што значи да не мора да буде тачна од речи до речи. Ипак један детаљ је битан – трансакција је увек извршење програма. Програм садржи кораке који су укључени у пословну трансакцију, на пример евиденција продаје књиге и резервисање примерка из инвентара.

Кажемо да трансакција извршава једну административну функцију иако то није увек случај. На пример то може бити извршавање функције у реалном времену као што је

преусмеравање позива у телефонској централи или контролисање неког алата у систему за контролу процеса у неком производном погону. Али најчешће ту је укључен новац, као што је продаја карата или пребацивање новца са једног рачуна на други.

Већина трансакционих програма приступа дељеним подацима али не баш сви. Неки имају чисто комуникациону функцију као што су прослеђивање поруке од једног система другом. Неки имају системску административну функцију као што је ресет неког уређаја. Апликација у којој ни један део програма не приступа дељеним подацима се не сматра системом за обраду трансакција због тога што таква апликација не захтева ни један од многих специјалних механизама које систем за обраду трансакција нуди.

Ту је обично једна онлајн корисник, као што је корисник од куће за веб прегледачем или кондуктер који наплаћује карте помоћу уређаја за наплату. Ипак неки системи немају ангажоване кориснике, као што су на пример који бележе поруке пристигле са сателита. Неки трансакциони програми раде офлајн или у беч (енгл. batch) режиму рада, што значи да укључују више корака који могу да трају дуже него што је корисник у могућности да чека програм да врати резултат. На пример приликом продаје производа онлајн већина посла се одвија након што је поруџбина унета: радник преузима поруџбину за обраду, узима производе са полице, брише их са стања, штампа етикету за отпрему, пакује је и предаје курирској служби.

2.1.2 Апликације за обраду трансакција

Апликација за обраду трансакција је скуп трансакционих програма дизајнираних да обављају функције неопходне да се аутоматизује задата пословна активност. Прва онлајн апликација за обраду трансакција која је се нашла у широкој употреби је био систем за резервацију авионских карата. SABRE систем је развијен раних 60их година прошлог века као заједнички подухват компаније IBM и American Airlines. SABRE је био један од највећих рачунарских система тог времена, и даље је веома велики систем за обраду трансакција. SABRE је се одвојио од компаније American Airlines и сада је вођен као засебна компанија Sabre Holding Corporation која пружа услуге за више од 200 авионских компанија и више од хиљаду путничких агенција и која поседује Travelocity веб сајт. Може да обрађује велики број летова, омогућава путницима да резервишу места, посебне оброке месецима унапред, нуди бонусе редовним корисницима и организује одржавање летилица и других операционих активности за авио компаније. Њихове максималне перформансе превазилазе 20,000 порука у секунди.

Данас постоји велики број других типова апликација за обраду трансакција и нове се стално појављују. У табели 1. дат је преглед неких од њих. Како се трошкови покретања трансакција и управљања великим базама података смањују све више типова административних функција ће бити корисно аутоматизовати помоћу апликације за обраду трансакција како због смањења трошкова администрације тако и да би се генерисао приход приликом пружања услуге купцу.

У почетку тржиште апликација за обраду трансакција било је резервисано пре свега за велике компаније које су имале потребу за подршком за административне функције за велики број корисника. Такви системи често су укључивали хиљаде терминала, десетине

дискова и доста великих процесора и могли су да обрађују хиљаде трансакција дневно. Велики системи за обраду трансакција постали су још важнији због популарности онлајн сервиса на интернету. Ипак са појавом мањих система дошло је до потребе за малим апликацијама за обраду трансакција, оних где постоји само пар прегледача повезаних на неки мали сервер да би се руковало поруџбинама за мали пословни каталог, да би се регистровали за неки курс у школи или да би се заказала посета зубару. Све ова апликације, велике и мале, ослањају се на исте структуре система и апстракције софтвера.

Табела 1. Апликације за обраду трансакција

Примена	Пример трансакције
Банкарство	Подизање новца са рачуна
Безбедно трговање	Куповина 100 акција на берзи
Осигурање	Уплата премијум осигурања
Контрола инвентара	Евиденција за испуњење налога поруџбине
Производња	Пријавите корак процеса склапања
Малопродајна тачка продаје	Забележи продају
Влада	Регистрација возила
Онлајн куповина	Направи поруџбину користећи онлајн каталог
Транспорт	Прати испоруку
Телекомуникације	Повезати телефонски позив
Војска команда и контрола	Испалити пројектил
Медиј	Одобрити дозволу за скидање видеа

Системи за обраду трансакција се такође нуде и као услуга другим компанијама. На пример amazon.com пружа услуге хостинга другим веб продавницама. Неке авио компаније развијају и пружају услуге резервације другим авио компанијама. Неки произвођачи готових апликација нуде своју апликацију као услугу коју може користити трећа страна путем интернета а која за узврат помаже трећој страни да понуди друге услуге за обраду трансакција својим корисницима. Када се узму у обзир трошкови, експертиза и квалитет менаџмента неопходни да би се направио један високо квалитетан систем за обраду трансакција овакав тренд развоја апликација за обраду трансакција ће највероватније расти.

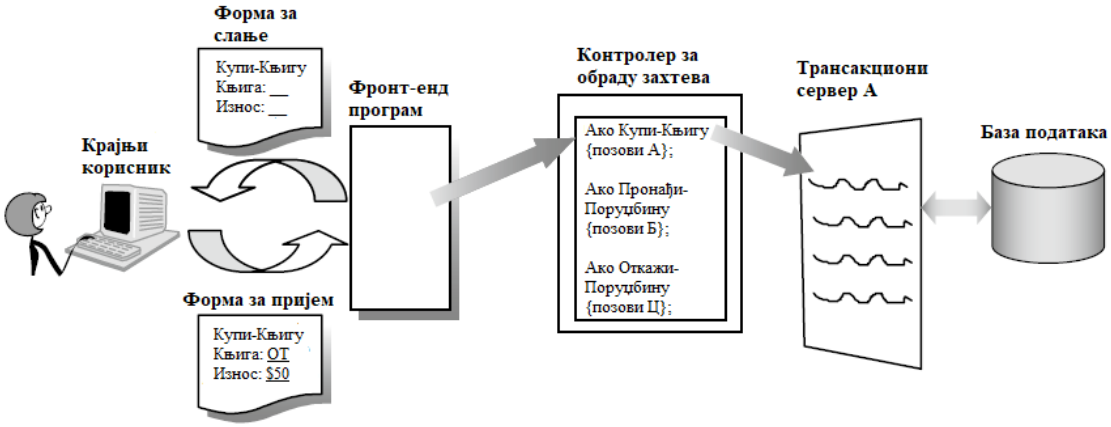
2.1.3 Основне функције програма за обраду трансакције

Програм за обраду трансакције углавном обавља следеће три ствари:

1. Узима унос преко веб прегледача или неке друге врсте уређаја као што су бар код читачи или сензори на роботу.
2. Обавља стварни посао који је захтеван.
3. Генерише одговор и ако је могуће враћа га претраживачу или уређају који је обезбедио унос.

Свако позивање трансакционог програма резултира независном јединицом рада који се извршава тачно једном и производи трајне резултате. О овоме ће бити више речи у даљем тексту рада.

Већина апликација за обраду трансакција укључује и неки део кода који се не извршава као трансакција. Тај код се извршава као обичан програм, није неопходно да се извршава као независна јединица посла која се извршава тачно једном и мора да произведе трајне резултате. Термин апликација за обраду трансакција се користи у ширем смислу. Он укључује трансакциони програм, програм који прикупља улазне информације за трансакције и функције за одржавање као што су брисање застарелих података о инвентару, реконфигурација система за извршавање и ажурирање валидационих табела које служе за проверу грешака.



Слика 1. *Делови апликације за обраду трансакције.*

2.2 Архитектура система за обраду трансакција

Систем за обраду трансакција је рачунарски систем, који укључује и хардвер и софтвер, који служи за хостинг трансакционих програма. Софтверски делови система за обраду трансакција обично су структурирани на посебан начин. Као што се може видети на слици, систем се састоји из неколико битних компоненти и различити делови апликације извршавају се на свакој од ових компоненти.

1. Уређај крајњег корисника: Крајњи корисник је неко ко захтева извршавање трансакција, то може бити корисник банке или веб продавнице. Уређај крајњег корисника може бити физички уређај као што је каса у продавници или може бити веб прегледач који је покренут на персоналном рачунару. Ако је то неки једноставан уређај онда он једноставно само приказује податке који су му прослеђени и шаље податке које је корисник унео са друге стране ако је то паметан уређај онда он извршава апликациони код који се назива фронт-енд програм.
2. Фронт-енд програм: Фронт-енд програм је апликациони код који комуницира са уређајем крајњег корисника. Обично шаље и прима менију и форме да понуди кориснику избор трансакција за покретање и да прикупи унос корисника. Често, уређај је веб прегледач а фронт-енд програм је апликација којом управља веб сервер који комуницира са прегледачем преко ХТТП-а. Фронт-енд програм врши проверу

уноса корисника и онда шаље поруку другом делу система чији је посао да заправо изврши трансакцију.

3. Контролер за обраду захтева: Контролер за обраду захтева је одговоран за примање порука од стране фронт-енд програма и претварање сваке поруке у један или више позива одговарајућим трансакционим програмима. У централизованом систему ово је једноставно ствар позивања локалног програма. У распрострањеном систему за обраду трансакција то захтева слање поруке систему где се програм налаза и може бити извршен. Ако је неопходно извршавање више од једног програма он је задужен за праћење тачке до које је дошао захтев док се креће између програма.
4. Трансакциони сервер: трансакциони сервер је процес који покреће делове трансакционог програма који извршава посао који је корисник захтевао обично читајући и уписујући податке у дељену базу података. Такође могуће је, ако је потребно, да позива друге програме или да враћа одговор који се прослеђује назад до уређаја који је обезбедио унос података за тај захтев.
5. Систем базе података: Систем базе података управља дељеним подацима који су потребни да би апликација извршила свој посао.

На пример, у апликацији за обраду поруџбина путем интернета, корисник доставља поруџбине преко веб прегледача. Веб сервер управља фронт-енд програмом који чита и уписује форме и мениje и евентуално одржава корпу за куповину. Контролер за обраду захтева прослеђује захтев од веб сервера трансакционом серверу који може да обради поруџбину коју је корисник захтевао. Трансакциони сервер приликом обраде поруџбине приступа бази података која прати наруџбине, информације из каталога и инвентар у складишту робе и евентуално контактира друге трансакциони сервер који наплаћује рачун преко кредитне картице.

Трансакциони програми који се покрећу на серверу су ограниченог броја типова и одговарају оперативним пословним процедурама као што су слање неке поруџбине или трансфер средстава. Обично тај број је између неколико десетина и не више од неколико стотина. Када апликација постане већа од овога обично се дели у неколико независних апликација мањег обима. Сваки од ових програма обично извршава мали део посла. Не постоји стандардни концепт по коме се одређује просечна величина трансакционих програма због тога што се они разликују на основу примене. Просечна трансакција може имати између нула и тридесет приступа бази података, између неколико хиљада до неколико милиона инструкција и најмање две поруке али често више од тога у зависности од тога како је подељена. Може бити подељена јер су потребне различите апликације за њихову обраду или због тога што је неопходно више машина да би се носиле са оптерећењем апликације. Очекује се да се програм изврши у року до секунде или две тако да корисник добије брз одговор. Касније ће се говорити о још једном, више техничком, разлогу зашто трансакције треба да буду мањег обима, а то има везе са конфликтом закључавања.

Базе података играју врло важну улогу у подржавању трансакционих програма и та улога је често већа од улоге самог апликативног програма. Иако база података може бити довољно

мала да може да се смести у примарну меморију чешћи је случај да је много већа од тога. Неке базе података захтевају велики број уређаја за трајно складиштење као што су магнетни или чврсти дискови гурајући систем базе података до крајњих граница. Да би могла да испрати те захтеве база података може бити копирана или подељена на више машина.

Још једна од битних категорија софтвера за обраду трансакција је трансакциони слој који је софтверска компонента између апликације за обраду трансакције и компонената нижег нивоа као што су оперативни систем, база података и алати за управљање системом. Ове компоненте обављају разне функције. Оне могу да омогуће апликацији да ефикасно користи процесе оперативног система, конекције на базу података, комуникационе сесије и тако омогући апликацији скалирање. На пример могу да обезбеде функцију коју ће корисничке апликације користити да би проследиле захтев до тачно одређене серверске апликације. Могу да интегришу апстракцију трансакције са апликацијом, оперативним системом и системом базе података, на пример да омогуће извршавање подељене трансакције кроз хетерогена окружења. Могу да интегришу алате за управљање системе да поједноставе управљањем апликација, на пример тако да систем менаџери могу да распореде оптерећење на више сервера у распоређеном систему. Такође они могу понудити програмски интерфејс и/или конфигурациона својства која поједностављују коришћење сродних услуга која потичу из оперативног система и система базе података.

Пре појаве светске мреже (енгл. World Wide Web, WWW) илити веба трансакциони слојеви називани су мониторима за обраду трансакција или мониторима за онлајн обраду трансакција (енгл. on-line transaction processing, OLTP). Током средине деведесетих година прошлог века апликациони сервери су представљени да би помогли програмерима апликација да се носе са новим проблемима који су настали доласком веба, као што су интеграција са веб серверима и веб прегледачима. У почетку апликациони сервери су створили мост између постојећих комерцијалних система контролисаних мониторима за обраду трансакција и интернета. У врло кратком периоду функционалност апликационих сервера и монитора за обраду трансакција је конвергирала. Током истог периода постао је популаран и MOM (енгл. Message-oriented middleware) трансакциони слој. MOM је постао основа за категорију производа названих EAI системи (енгл. Enterprise application integration system). Усвајањем стандардних протокола за комуникацију између апликација путем интернета, названом веб услуге (енгл. Web Services), довело је до још једног производа трансакционог слоја који се назива ESB (енгл. Enterprise Service Bus). На крају трансакциони слојеви су постали такви да помажу корисницима да дефинишу и управљају дугорочним пословним процесима. Иако се трансакциони слој као производ сматра комплетним окружењем за развој и извршавање апликације за обраду трансакција, корисници по некад користе компоненте из више трансакционих слојева да саставе своје окружење за обраду трансакција.

2.3 ACID

Постоје четири кључна својства трансакција која је неопходно разумети на самом почетку:

- Атомност (енгл. Atomicity) - Трансакција се извршава комплетно или се не извршава уопште
- Конзистентност (енгл. Consistency) – Извршење трансакције треба да преведе базу података из једног у друго конзистентно стање
- Изолација (енгл. Isolation) – Трансакција не треба своје промене базе података да учини видљивим другим трансакцијама пре него што се она оконча, односно промене потврде у бази података
- Трајност (енгл. Durability) – Када су у бази података потврђене промене које је извршила нека трансакција те промене се више не могу изгубити [3]

Коришћењем почетних слова енглеских речи ова четири својства трансакција добијамо акроним ACID. Често се каже да систем за обраду трансакција извршава ACID трансакције и у том случају систем за обраду трансакција пролази ACID тест.

2.3.1 Атомност

Прво трансакција треба да буде атомична (све или ништа) што значи да се извршава комплетно или се не извршава уопште. Не сме се оставити могућност да се изврши само један део трансакционог програма.

На пример претпоставимо да имамо трансакциони програм који пребацује 1000 динара са рачуна А на рачун Б. Дакле подиже се 1000 динара са рачуна А и уплаћују се на рачун Б. Када се покрене ова трансакција она мора бити атомска – или се извршавају оба ажурирања или ни једно. Не сме бити могуће да се једно ажурирање изврши а друго не.

Систем за обраду трансакција гарантује атомичност преко механизама базе података и прати извршавање трансакција. Ако трансакциони програм буде стопиран из било ког разлога пре него што изврши свој задатак систем за обраду трансакција ће поништити ефекте свих ажурирања које је трансакција већ извршила. Једино у случају када се сва ажурирања изврше до краја систем за обраду трансакција ће дозволити да ажурирања постану стални део базе података.

Ако дође до отказа система за обраду трансакција онда као део његовог опоравка поништиће се ефекти ажурирања свих трансакција које су се извршавале у моменту отказа система. Ово обезбеђује да се база података врати у познато стање пре отказа система чиме се смањује потреба за мануелном интервенцијом током рестарта система.

Користећи својство атомичности можемо писати трансакционе програме који имитирају једну атомску пословну трансакцију као што је подизање новца са банковног рачуна, резервација лета или продаја акција на берзи. Свака од ових пословних акција захтева ажурирање више података. Имплементацијом пословне акције помоћу трансакција обезбеђујемо да ће се или сва ажурирања извршити или неће ни једно.

2.3.2 Конзистентност

Друго својство трансакција је конзистентност – извршење трансакције мора да преведе базу у друго стабилно стање [4]. То значи када извршимо трансакцију над базом података која је на почетку у конзистентном (стабилном) стању онда када се извршавање трансакције заврши база података се поново враћа у конзистентно стање.

Када се говори о конзистентности мисли се на интерну конзистентност. За базу података то значи да база података у најмању руку задовољава сва ограничења интегритета. Постоји неколико различитих ограничења интегритета које систем базе података обично одржава:

- Сви примарни кључеви су јединствени (нпр. не постоје два уписа производа под истом шифром)
- База података има референцијални интегритет што значи да се подаци односе само на објекте који постоје (нпр. подаци о поруџбини имају референцу на податке о производу и купцу који заиста постоје)
- Одређене вредности се налазе у одређеном опсегу (нпр. број година радника је мањи од 120, ЈМБГ нема вредност NULL) итд.

Обезбеђивање да трансакције одржавају конзистентност базе података је добра програмерска пракса. Ипак за разлику од атомичности, изолације и трајности конзистентност је подељена одговорност између трансакционог програма и система за обраду трансакција који извршава овај програм. Систем за обраду трансакција обезбеђује да су трансакције атомичне, изоловане и трајне без обзира да ли су програмиране да сачувају конзистентност. Због овога иако систем за обраду трансакција обавља свој део у одржавању конзистентности он може гарантовати само атомичност, изолацију и трајност, а на програмеру апликације је одговорност да обезбеди да трансакциони програм сачува конзистентност.

2.3.3 Изолација

Треће својство трансакција назива се изолација. Кажемо да је сет трансакција изолован ако је ефекат система који их извршава исти као да их систем извршава једну по једну. Техничка дефиниција изолације је серијабилност (линеарност). Извршавање је серијабилно (изоловано) ако је ефекат исти као да се трансакције извршавају серијално, једна након друге, у низу, без преклапања у извршавању било које две од њих. Ово има исти ефекат као када се трансакције извршавају једна по једна.

Класичан пример не изолованог извршавања је банкарски систем где две трансакције обе покушавају да подигну последњих 1000 динара са рачуна. Ако обе трансакције провере стање на рачуну пре него што било која од њих ажурира податке онда ће обе трансакције одлучити да је на рачуну довољно новца да се изврши њихов захтев и обе ће подићи последњих 1000 динара. Очито ово је нежељени резултат. Шта више то није серијабилан резултат. Приликом серијабилног извршавања само ће прва трансакција бити у могућности да се изврши и подигне последњих 1000 динара. Друга ће имати увид да је рачун празан.

Приметићемо да се изолација разликује од атомичности. У наведеном примеру обе трансакције се извршавају до краја успешно што значи да су атомске. Ипак оне нису изоловане и због тога производе нежељено понашање.

Ако је извршавање серијабилно онда са тачке гледишта крајњег корисника, који покреће захтев да се трансакција изврши, систем изгледа као издвојени систем који покреће само ту трансакцију. У међувремену између два захтева клијента да се изврше неке трансакције, трансакције других клијената већ могу бити покренуте. Али током периода током којег систем обрађује трансакцију нашег корисника он има илузију да систем не ради ништа друго. Наравно ово је само привид. Систем који би заиста извршавао трансакције серијабилно био би прилично неефикасан. Да би се ресурси што боље искористили трансакције се извршавају истовремено.

Ако свака трансакција очува конзистентност онда свако појединачно извршавање таквих трансакција у низу сачуваће конзистентност. Како свако серијабилно извршавање је еквивалентно редном извршавању онда ће и серијабилно извршавање ових трансакција одржати конзистентност базе података. Комбинација конзистентних трансакција и изолације обезбеђује да ће извршавање сета трансакција очувати конзистентност базе података.

База података обично поставља локот на податке којима приступа свака трансакција. Ефекат постављања локота је да учини да се извршење дешава у низу. Заправо интерно систем покреће трансакције паралелно једну за другом.

Честа заблуда је да серијабилност није важна због тога што ће систем базе података одржати конзистентност примењујући ограничења интегритета. Ипак као што смо видели постоје конзистентна ограничења која систем базе података не може да спроведе. Чак шта више понекад корисник не говори систему базе података да спроведе нека ограничења јер то умањује перформансе система. Последња линија одбране је да трансакциони програм самостално одржава конзистентност и да систем гарантује серијабилност.

2.3.4 Трајност

Четврто својство трансакција је трајност. Трајност значи да када трансакција заврши извршавање сва ажурирања су смештена на трајну меморију. Трајна меморија је меморија која ће преживети нестанак струје или пад оперативног система. Данас се трајна меморија, која се такође назива и постојана или секундарна меморија, обично састоји од магнетних (хард) дискова или SDD уређаја који користе флеш меморију и који су све заступљенији као алтернатива хард дисковима. Чак и у случају да трансакциони програм откаже или да оперативни систем падне, једном када је трансакција извршена њени резултати су трајно смештени на трајну меморију и они се могу пронаћи ту након опоравка од пада система.

Трајност је важна јер свака трансакција која пружа услугу кориснику је и уговор између корисника и предузећа које пружа услугу. На пример, ако пребацујемо новац са једног рачуна на други, једном када добијемо одговор од трансакције који потврђује да је успешно извршена, очекујемо да је резултат сталан. То је законски уговор између корисника и система да је новац пребачен са једног на други рачун. Због тога је важно да трансакција

обезбеди да су ажурирања смештена на неку трајну меморију да би тиме обезбедила да та ажурирања не могу бити изгубљена након што је трансакција извршена. Чак шта више трајност резултата мора бити сачувана на дужи временски период. На пример чак и ако корисник провери рачун који није коришћен неколико година он очекује да ће наћи свој новац на њему следећег пута када му буде приступио.

Својство трајности се обично обезбеђује тако што систем за обраду трансакција додаје копију свих трансакција у лог фајл док је трансакциони програм покренут. Када трансакциони програм изда Commit наредбу, систем прво обезбеди то да су сви подаци који се налазе у лог фајлу смештени на трајну меморију а након тога враћа се трансакционом програму указујући на то да је трансакција заиста у потпуности извршена и да су резултати трајни. Ажурирања могу бити уписана у базу података одмах или у неким случајевима са малом задршком. Како год ако систем откаже након што је трансакција успешно извршена а пре него што се ажурирања изврше над базом података, након што се систем опорави од отказа он мора опоравити и базу података. Да би се ово извршило систем чита лог фајл и проверава свако ажурирање које су извршене трансакције извршиле над базом података. Ако се догоди да неко ажурирање није извршено над базом података оно се поново извршава. Када се опоравак система заврши он наставља са нормалним радом. Стога након што се систем опорави свака нова трансакција ће прочитати стање базе података која укључује све исправке трансакција које су учињене пре отказа.

2.3.5 Генерализација ACID својстава

ACID је сет својстава који гарантује поузданост базе података [5]. ACID својства су првенствено развијена за традиционалне, пословно орјентисане апликације као што је банкарство. Стога, оне не могу у потпуности подржати функционалност и перформансе које захтевају напредне апликације база података какве су CAD и CAM софтвери, аутоматизација канцеларијског пословања, управљање рачунарским мрежама, MDBS итд. На пример трансакције у некој апликацији за рачунарско пројектовање генерално дуго се извршавају и придржавање традиционалних ACID својстава у таквим трансакцијама би захтевало закључавање ресурса на дуг временски период. Ово је довело до генерализације ACID својстава и сада имамо Опоравак (енгл. Recovery), Доследност (енгл. Consistency), Видљивост (енгл. Visibility) и Сталност (енгл. Permanence). Циљ овакве генерализације је да се смање нека од ограничења која су наметнута ACID својствима. На пример видљивост смањује својство изолације омогућујући дељење делимичних резултата и стога унапређује сарадњу између трансакција које се одвијају истовремено. Што су више генерализована својства ACID флексибилнији је одговарајући модел трансакције.

Ограничења наслеђена од изворних ACID својстава и особености напредних апликација база података довела је до генерализације ACID својстава. Својство опоравка се односи на способност да се база података доведе у стање које се сматра исправним у случају отказа. Доследност се односи на тачност стања базе података које производи извршена трансакција. Видљивост се односи на способност једне трансакције да види резултате друге трансакције која се извршава. Сталност се односи на способност трансакције да смести своје резултате у базу података. Флексибилност трансакционог модела зависи од начина на који су ACID својства генерализована. Напредни модели трансакција су детаљно описани у књизи Трансакциони Модели База Података за Напредне Апликације [6]. Саге (енгл. Saga),

Угнеждене трансакције (енгл. Nested Transactions) и Флекс (енгл. Flex) су репрезентативни примери модела који генерализују ACID својства док је ACTA пример формалног фрејмворка за изражавање ових модела.

Сага је ланац трансакција који је атомичан. Својство изолације је смањено на нивоу Саге и видљивост је дозвољена на границама трансакционих компонената. Сага је сама по себи атомска али због умањеног ограничења видљивости она подржава семантичку конзистенцију. То значи да свака трансакција у ланцу има семантички инверзну или компезујућу трансакцију која је повезана са њом. Ако се једна од трансакција у низу Саге не изврши до краја трансакције се враћају уназад обрнутим редоследом од њиховог извршења. Извршене трансакције се враћају уназад извршавајући њима одговарајуће компезујуће трансакције.

Угнеждене трансакције омогућавају истовремено одвијање трансакција унутар трансакције и тако омогућавају хијерархијску контролу за управљање у случају отказа. Оригинални угнеждени трансакциони модели који подржавају само затворене под-трансакције су проширени да укључују и отворене под-трансакције. Затворене под-трансакције могу да не подржавају конзистентност и трајност. Затворене под-трансакције предају резултат свом родитељу. Ови парцијални резултати који се добијају постају видљиви тек након што се врховна (родитељ) трансакција изврши. Ово обезбеђује атомичност и изолацију целе трансакције. Због умањеног ограничења својства видљивости отворене под-трансакције одмах омогућују приступ својим резултатима који тако постају видљиви другим трансакцијама.

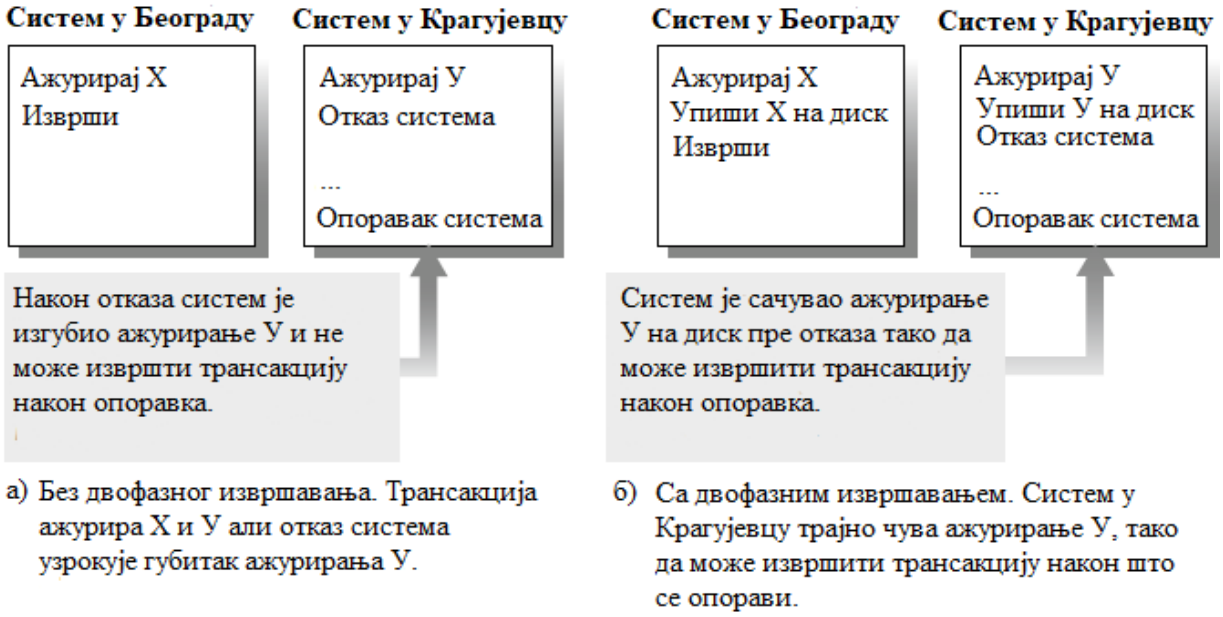
Флекс трансакциони модел је био предвиђен да генерализује ACID својства у MDBS. Он умањује ограничења својства атомичности и изолације угнеждених трансакција да би омогућио кориснику увећану флексибилност приликом одређивања могућности својих трансакција. Флекс трансакција може наставити и извршити се чак и ако се нека од њених под-трансакција не изврши успешно. Такође омогућава дефинисање зависности од под-трансакција на интерне или екстерне зависности. Интерне зависности дефинишу редослед извршавања под-трансакција док екстерне зависности дефинишу зависност извршења под-трансакција у односу на неки догађај који не припада трансакцији као што је на пример време почетка и време краја извршења трансакције. Флекс модел такође омогућује корисницима да контролишу грануларност својства изолације трансакције коришћењем компензујућих под-трансакција.

ACTA фрејмворк који олакшава формални опис својстава проширеног трансакционог модела. Он дефинише конструкције које олакшавају синтезу проширених модела трансакција прилагођавањем/комбиновањем постојећих модела изградњом од самог почетка. Различите нотације су уведене у ACTA фрејмворк да би се омогућила генерализација ACID својстава из различитих перспектива. На пример појам делегирања дозвољава трансакцијама да селективно прекину неке од операција које су се извршавале а да се сама трансакција ипак успешно изврши [7].

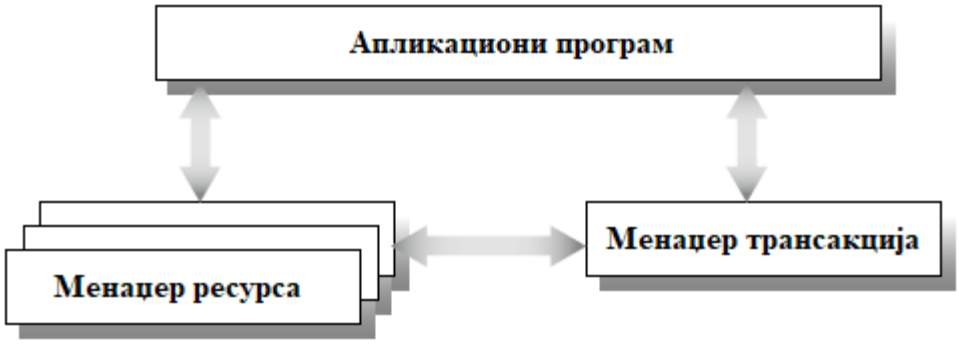
2.4 Двофазно извршавање

Када трансакција ажурира податке на две или више система база података и даље се мора обезбедити својство атомичности, наиме или обе базе података трајно инсталирају ажурирања или ни једна то не чини. Ово је изазов због тога што системи база података могу независно да откажу и да се опорављају. Ово је свакако проблем када се системи базе података налазе на различитим чворовима дистрибуираног система. Али то може чак бити проблем и на једној машини ако системи базе података функционишу као серверски процеси са приватним складиштем пошто процеси могу да раде независно. Решење за овај проблем је протокол који се назива двофазно извршавање [енгл. two-phase commit, 2PC], који се извршава од модула који се назива трансакциони менаџер.

Суштина проблема је у томе што трансакција може извршити ажурирања на једном систему база података, али други систем база података може отказати пре него што се трансакција изврши и на њему. У оваквом случају када се систем који је отказао опоравља он мора бити способан да изврши трансакцију до краја. Да би извршио трансакцију систем који се опоравља мора имати копију трансакционих ажурирања која су извршена на њему. С обзиром да систем може изгубити садржај из примарне меморије у тренутку отказа он мора сместити трајну копију трансакционих ажурирања пре него што откаже да би их имао након што се опорави. Овај начин размишљања доводи до суштине двофазног извршавања: Сваки систем базе података којем приступа трансакција мора трајно чувати свој део ажурирања трансакције пре него што се трансакција изврши било где. На овај начин ако систем С откаже након што је трансакција извршена на другом систему С' али пре него што се та трансакција изврши на систему С онда се та трансакција може извршити на систему С након његовог опоравка (погледати Слику 2.).



Слика 2. *Како двофазно извршавање обезбеђује атомичност*



Слика 3. *X/Open Трансакциони модел (XA)*

Да би боље разумели двофазно извршавање помаже визуализација целокупне архитектуре у којој менаџер трансакцијама (енгл. Transaction Manager) функционише. Стандардни модел, приказан на Слици 3, је представљен од стране IMB-овог CICS-а, а популаризовали су га Ораклеов Tuxedo и X/Open. У овом моделу трансакциони менаџер комуницира са апликацијом, менаџером ресурса и другим трансакционим менаџерима. Концепт ресурса укључује базе података, низове, фајлове, поруке и друге дељене објекте којима се може приступити у оквиру трансакције. Сваки менаџер ресурса нуди операције које мора покренути само ако је трансакција која позвала операције успешно извршена.

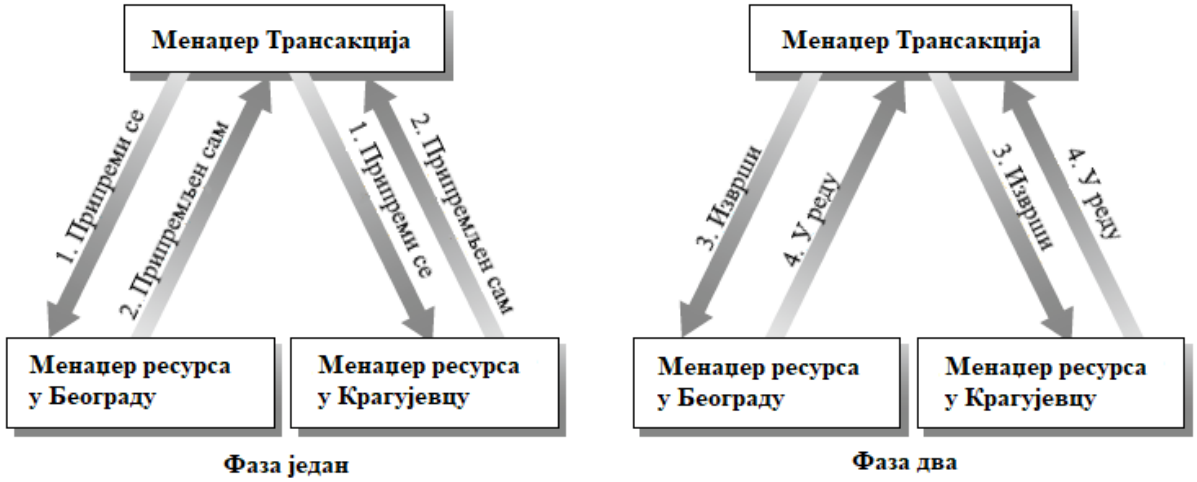
Трансакциони менаџер процесира основе трансакционе операције за апликацију: покрени (енгл. Start), изврши (енгл. Commit) и откажи (енгл. Abort). Апликација позива Start да започне извршавање нове трансакције. Позива Commit да затражи од трансакционог менаџера да изврши трансакцију. Позива Abort да каже трансакционом менаџеру да прекине трансакцију.

Трансакциони менаџер је пре свега књиговођа који води евиденцију о трансакцијама да би се обезбедила атомичност када је више од једног ресурса укључено. Обично постоји један трансакциони менаџер на сваком чвору дистрибуирано рачунарског система. Кад апликација изда операцију Start трансакциони менаџер издаје јединствену идентификацију (енгл. Identification, ID) за трансакцију који се назива идентификатор трансакције. Током извршавања трансакције он води евиденцију свих менаџера ресурса којима је та трансакција приступила. Ово захтева сарадњу да апликацијом, менаџерима ресурса и комуникационим системом. Кад год трансакција приступу новом менаџеру ресурса неко то мора да пријави менаџеру трансакција. Ово је важно због тога што када дође време да се трансакција изврши трансакциони менаџер мора да зна све менаџере ресурса са којима треба да разговара да би могао да изврши протокол двофазног извршавања.

Када трансакциони програм заврши извршавање и изда команду Commit, та команда Commit долази до менаџера трансакција, који обрађује команду извршавајући протокол двофазног извршења. Слично томе ако менаџер трансакција прими команду Abort он говори менаџеру ресурса да поништи сва ажурирања која је трансакција извршила, тј. да прекине трансакцију на сваком менаџеру ресурса. Због тога сваки менаџер ресурса мора разумети концепт трансакције у смислу да поништава или трајно чува ажурирања која је трансакција

извршила у зависности од тога да ли је трансакција успешно извршена (Commit) или је прекинута (Abort).

Када покреће двофазно извршење, менаџер трансакција шаље две рунде порука – по једну за сваку фазу активности извршења. У првој рунди порука он говори свим менаџерима ресурса да припреме извршење уписујући копију резултата трансакције на трајну меморију, али не и да заиста изврше трансакцију. У овом тренутку менаџери ресурса говоре да су спремни за извршење. Када менаџер трансакција добије потврде назад од свих менаџера ресурса он зна да је читава трансакција припремљена. У том тренутку он зна да је су сви менаџери ресурса сачували трајну копију ажурирања трансакције али ни један од њих није извршио трансакцију. Због тога он шаље другу рунду порука где говори менаџерима ресурса да заправо изврше трансакцију. Слика 4. даје један пример извршења двофазног извршења са укључена два менаџера ресурса.



Слика 4. *Протокол двофазног извршавања*

Двофазни протокол извршавања избегава проблем са слике 2(а) због тога што сви менаџери ресурса имају трајну копију ажурирања трансакције пре неко што их било који од њих изврши. Због тога чак и ако дође до отказа система током Commit активности, као што се на примеру на слици дешава са системом у Крагујевцу, он може извршити трансакцију након оправка. Како год да би било омогућено да све ово функционише, протокол мора бити способан да се избори са сваким могућим сценаријом отказа и опоравка система. На пример на слици 2(б) он мора рећи систему у Крагујевцу да изврши трансакцију.

Двофазни протокол извршавања је неопходан када год трансакција приступа два или више менаџера ресурса. Стога једно кључно питање на које дизајнери апликације за обраду трансакција морају дати одговор је да ли је потребно или не њихов трансакциони програм буде распрострањен на више ресурса. Коришћење двофазног извршавања увећава опште трошкове (због система порука које двофазни протокол извршавања користи), али могућност да се систем размести на више локација може обезбедити бољу скалабилност (додавање додатних система да би се повећао капацитет) и доступност (у случају да један систем откаже други остају оперативни).

2.5 Типови система

До сада смо говорили о обради трансакције као типу апликације који покреће кратке трансакционе програме који приступају дељеној бази података. Обрада трансакција је такође и тип система, начин конфигурације софтверских компонената да извршавају одређени посао који је захтеван од апликације за обраду трансакција. Корисно је упоредити овај тип система са другим типовима који постоје да би се уочила разлике и због чега је конструкција трансакционог система другачија од њих. Постоји неколико других типова система на које треба обратити пажњу:

- Пакетно (енгл. Batch) процесирање – систем где подносите захтев а тек касније примите резултат у форми фајла
- Системи у реалном времену (енгл. Real-time systems) – системи где подносите захтев да се изврши мали задатак који мора бити завршен пре истека врло кратког временског рока
- Системи складишта података (енгл. Data warehouse systems) – су системи где програми за извештаје и ад хок упити приступају подацима који су интегрисани из различитих извора

Дизајнирање система да извршава један од ових типова процесирања назива се системско инжењерство. Уместо дизајнирања специфичне компоненте, као што је један оперативни систем или систем базе података, дизајнира се један интегрисани систем комбиновањем различитих врста компонената да изводе одређени тип задатка. Често, системи су дизајнирани да могу да подржавају различите типове, али за сврху упоређивања различитих типова о њима ће се говорити као да се сваки тип система покреће у посебном дизајнерском окружењу. У наставку су дати захтеви за сваки од ових типова система и њихово поређење са системом за обраду трансакција.

2.5.1 Batch processing систем

Batch је сет захтева који су процесирани заједно, често дуго након што је захтев поднесен. Системи за процесирање података шездесетих и раних седамдесетих година су били углавном batch системи за процесирање. Данас они су још увек у употреби. Али уместо да се покрећу на системима додељеним за batch процесирање они се често извршавају на системима који покрећу и системе за обраду трансакција. Системи за обраду трансакција могу извршавати batch послове током периода када систем није оптерећен с обзиром да посао који обављају batch системи има флексибилно време одговора. Да би направили јасну разлику између система за обраду трансакција и batch система упоредићемо систем за обраду трансакција који покреће чисте задатке обраде трансакција наспрам класичног batch система који покреће чисте batch послове иако је комбинација ова два система уобичајена.

Batch систем за обраду извршава сваки batch као секвенцу трансакција, једна по једна трансакција. Како се трансакције извршавају серијално не постоји проблем серијабилности. За разлику од тога на систему за обраду трансакција може се извршавати више трансакција истовремено тако да систем има додатни посао да обезбеди серијабилност.

На пример израчунавање вредности акција портфолија берзе могло би се извршавати као batch апликација која се покреће једном дневно након затварања финансијских тржишта. Рачунање месечног рачуна за телефонске кориснике може бити batch апликација која се извршава на дневном нивоу за различите групе корисника. Такође генерисање докумената за пореске извештаје може бити batch апликација која се извршава једном квартално или једном годишње.

Главна мера перформанси batch процесирања је пропусност, односно количина посла урађена по јединици времена. Време одговора је мање важно. Batch се може извршавати минутима, сатима чак и данима. На супрот тога систем за обраду трансакција има важне захтеве у вези времена одзива због тога што је у већини случајева корисник који очекује приказ резултата трансакције.

Код класичног batch процесирања апликација узима свој унос као фајл оријентисан ка подацима чији подаци представљају низ порука захтева. Резултати апликацију су такође смештени у фајлу. За разлику од тога систем за обраду трансакција обично поседује велику мрежу уређаја за узимање захтева и приказивање резултата.

Batch процесирање може бити оптимизовано сортирајући улазне захтеве у складу са редоследом података у бази података. На пример ако захтеви одговарају додељивању кредита за скорашње летове награђеним корисницима, захтевани подаци о летовима корисника могу бити сортирани према броју рачуна за наградне миље. На овај начин врло је лако и ефикасно да се обраде подаци спајањем процедуре која чита базу података о додели наградних миља према редоследу броја рачуна. Са друге стране захтеви за обраду трансакција долазе у насумичном реду. Због тога што одговор мора да да стигне у што краћем времену систем не може да троши време на сортирање улазне захтеве да би били конзистентни са базом података. Он мора бити способан да приступа подацима насумично у редоследу према којем су подаци захтевани.

Класично batch процесирање узима фајл са порукама захтева и постојећи фајл базе података као улаз и производи нову мастер базу података као резултат покренутих трансакција на захтеве. Ако се догоди batch процесирање не успи да се изврши не постоји никаква учињена штета због тога што су улазни фајл и улазна база података не измењени, у том случају једноставно се одбаци добијени излазни фајл и поново се покрене batch програм. Када говоримо о систему за обраду трансакција он ажурира базу података на мрежи како захтеви пристижу. Због тога откази могу остави базу података у неконзистентном стању због тога што садржи резултате незавршених трансакција. Овај проблем атомичности за трансакције у окружењу обраде трансакција не постоји у batch окружењу.

На крају у batch процесирању оптерећење система је фиксно и предвидљиво па систем може бити дизајниран за то оптерећење. На пример можете програмирати систем да покрене batch у одређено време и издвојити довољне капацитете да то изврше, због тога што тачно знате колико оптерећење ће бити. Код система за обраду трансакција оптерећење генерално варира током дана. Постоје пикови када се одвија много активности и периоди када се одвија много мало активности. Систем мора бити димензионисан тако да може да се носи

са максималним оптерећењем али и дизајниран тако да ефикасно користи вишак капацитета током мирног периода.

2.5.2 Real-Time системи

Системи за обраду трансакција су слични real-time системима, такви системи прикупљају унос са сателита или контролишу опрему у фабрикама. Обрада трансакција је у суштини врста real-time система са захтеваним real-time временом одзива од 1 до 2 секунде. Он одговара стварном процесу који се састоји од крајњих корисника који међусобно делују са приказом на уређајима који комуницирају са апликацијским програмима који приступају заједничкој бази података. Тако да није изненађујуће што постоји много сличности између ове две врсте система.

Обе врсте система имају предвидива оптерећења са повременим пиковима. Системи у реалном времену обично имају нагласак на прикупљању уноса а не обраде, док системи за обраду трансакција углавном врше оба.

Због различитих реалних процеса које контролишу, real-time системи морају да користе више специјализованих уређаја него системи за обраду трансакција, као што су лабораторијска опрема, опрема у фабрикама или сензори и контролни системи у једном аутомобилу или авиону.

Real-time системи генерално не морају да користе специјалне механизме за атомичност и трајност. Они једноставно обрађују улазне информације најбрже што могу. У случају да изгубе неке уносе они једноставно игноришу губитке и настављају да се извршавају. Да би разумели зашто размотримо пример система који прикупља информације од сателита за праћење. Није добро када систем пропусти део података који пристиже али систем свакако не може прекинути са радом да би се вратио да поправи ствари као што би систем за обраду трансакција урадио. У овом случају подаци настављају да пристижу и систем мора да да све од себе да настави да их обрађује. Са друге стране окружење у којем функционише систем за обраду трансакција може престати да прихвата унос на кратак временски период или може задржати унос на кратко. Ако дође до отказа може се зауставити прикупљање уноса покренути се процедура опоравка а након тога наставити са обрадом уноса. Према томе толеранција коју ова два типа система према квару је прилично различита.

Real-time системи нису оптерећени серијабилношћу. У већини real-time апликација, обрада порука из уноса не укључује приступ дељеним подацима. Како обрада два различита уноса нема утицај између себе чак и у случају да се они обрађују истовремено, они ће се понашати као да су се извршили серијски. Нису потребни специјални механизми као што је закључавање. Приликом обрађивања података у реалном времену за дељене податке, појам серијабилности је подједнако релевантан као и да обраду трансакција. Међутим у оваквом случају real-time апликација користи примитиве синхронизације ниског нивоа за међусобно искључивање, уместо да се ослања на механизме синхронизације опште намене који је скривен иза апстракције трансакције.

2.5.3 Складиште података

Систем за обраду трансакција обрађује податке у свом сировом стању како долазе. Систем складишта података интегрише податке из више извора у базу података одговарајућу за упите. На пример, дистрибутивна компаније одлучује сваке године како да додели буџет за маркетинг и оглашавање. Она користи систем за обраду трансакција да обради продајне налоге које укључују врсту и вредност сваког налога. База података о купцима говори локацију сваког купца, годишњи приход и стопу раста. База финансија укључује информације о трошковима и приходима и говори која линија производа је најпрофитабилнија. Компанија повлачи податке из ова три извора у складиште података. Пословни аналитичари могу извући из складишта података информације како да најбоље распореде промотивне ресурсе.

Системи складишта података извршавају две врсте задатака: batch zadatak при чему се врши извлачење података из извора, чишћење података да би се уклониле разлике између њих, претварање података у облик који је пригодан за упите и учитавање података у складиште података; и извршавања упита над складиштем података који могу да варирају од кратких интерактивних захтева до комплексних анализа које генеришу велике извештаје. Оба од ових задатака су прилично различити од обраде трансакција која се састоји од кратких ажурирања и упита. Такође, за разлику од обраде трансакција садржај складишта података може бити и мало застарео са обзиром на то да корисник тражи трендове на које не утичу превише последња ажурирања. Чак шта више понекад је важно радити на статичној копији базе података да би резултати узастопних упита били упоредиви. Покретање упита над складиштем података а не на бази података за обраду трансакција је такође корисно са становишта перформанси с обзиром да упити за складиштење података успоравају ажурирања трансакција. Упоређивање стилова система је дато у табели 2.

2.5.4 Остали типови система

Два друга типа система која су повезана са обрадом трансакција су тајмшеринг (енгл. timesharing) и клијент-сервер.

2.5.4.1 Тајмшеринг

У тајмшеринг систему уређај за приказивање је повезан са процесом оперативног система и у оквиру тог процеса корисник може позвати програме који могу да интерагују са екраном. Пре широке употребе персоналних рачунара када су тајмшеринг системи били популарни системи за обраду трансакција су често мешани са њима због тога што оба система укључују управљање многим уређајима који су повезани са заједничким сервером. Али они су прилично различити у смислу оптерећења, захтева за перформансама и захтевима доступности:

- Тајмшеринг систем има веома непредвидљиво оптерећење због тога што корисници константно праве другачије захтеве на систему. Ради упоређивања, оптерећење система за обраду трансакција је прилично предвидљиво, свакога дана је шаблон оптерећења исти.

- Тајмшеринг системи имају мање строге захтеве доступности и атомичности него системи за обраду трансакција. Извршење ACID концепта се не примењује.
- Тајмшеринг апликације нису критичне у истој мери као апликације за обраду трансакција и због тога имају слабије захтеве за доступношћу.
- Перформансе тајмшеринг система се мере у смислу капацитета система као што су инструкције по секунди или број корисника на мрежи. За разлику од обраде трансакција не постоје општеприхваћени тестови који могу тачно представљати понашање широког спектра тајмшеринг апликација.

Табела 2. Упоређивање типова система

	Обрада трансакција	Batch	Real-Time	Складиште података
Изолација	Серијабилно, мулти-програмирано извршење	Серијско, једном-програмирано извршење	Нема трансакциони концепт	Нема трансакциони концепт
Оптерећење	Велика варијација	Предвидљиво	Предвидљивост зависи од апликације	Предвидљиво оптерећење, велика варијација упита
Мерење перформанси	Време одзива и пропусност	Пропусност	Време одзива, пропусност, прекорачени рокови	Пропусност за оптерећење, време одзива за упите
Унос	Мрежа уређаја за унос захтева	Фајл орјентисан ка подацима	Мрежа уређаја за унос података и операција	Мрежа уређаја за унос и извршење упита
Приступ подацима	Директан приступ	Приступ сортиран да буде конзистентан са распоредом у бази	Слободан	Сортиран за учитавање, слободан за упите
Опоравак	Након отказа, обезбеђује да база изврши успешна ажурирања и одбаци остала	Након отказа, поново покреће batch да створи нови мастер фајл	Одговорност апликације	Одговорност апликације

2.5.4.2 Клијент-сервер

У клијент-сервер систему велики број персоналних рачунара разговара са дељеним сервером на локалној мрежи. Ова врста система је веома слична окружењу за обраду трансакција где се велики број уређаја конектује на дељени сервер који покреће трансакција. У неком смислу систем за обраду трансакција је био оригинални клијент сервер систем са веома једноставним десктоп уређајима, такозваним "глупим" терминалима. Како су десктоп уређаји постајали све моћнији, систем за обраду трансакција и системи персоналних рачунара су конвергирали у један тип рачунарског окружења са различитим типовима сервера као што су фајл сервер, комуникациони сервер и сервери за обраду трансакција.

2.5.5 Зашто дизајнирати систем за обраду трансакција

Сваки тип система који је поменут дизајниран је за одређен шаблон коришћења. Иако је дизајниран за тај шаблон коришћења он заправо може бити коришћен и на други начин. На пример људи су користили тајмшеринг систем да покрену апликацију за обраду трансакција. Ове апликације обично нису скалабилне или не користе ресурсе оперативног система ефикасно али је и то изводљиво. На пример за специјалне сврхе дизајниран је систем за обраду трансакција користећи системе у реалном времену и batch системе који се покрећу у тајмшеринг системима.

Обрада трансакција има довољно специјалних захтева да је вредно дизајнирати систем за ту сврху. Количина новца коју предузећа троше на системе за обраду трансакција оправдавају додане инжењерске и дизајнерске радове које произвођачи ових система улажу да би што боље прилагодили њихов производ обради трансакција, за боље перформансе, већу поузданост и једноставност коришћења.

2.6 Закључавање

Једно важно својство трансакција је да су изоловане. Технички, ово значи да извршавање трансакција има исти ефекат као и покретање трансакција серијски, једну за другом у низу без преклапања у извршавању било које две од њих. Такво извршавање се назива серијабилно, што значи да оно има исти ефекат као серијско извршавање. Серијабилно извршавање даје сваком кориснику једноставну илузију да читав систем извршава само ту његову/њену трансакцију и да не ради ништа друго.

Најпопуларнији механизам који се користи да се постигне серијабилност је закључавање. Концепт је једноставан:

- Свака трансакција резервише приступ подацима које користи. Ова резервација се назива закључавање.
- Постоје локот читања (енгл. read locks) и локот писања (енгл. write locks).
- Пре читања дела података трансакција поставља локот читања. Пре него што уписује податке поставља се локот писања.

- Локот читања је у конфликту са локотом писања, а локот писања са оба и читања и писања.
- Трансакција може обезбедити локот само ако ни једна друга трансакција на исте податке нема постављен локот који је у конфликту са њим. Према томе може обезбедити локот читања на податке x само ако ни једна друга трансакција нема локот писања постављен над x. Може обезбедити локот писања над x само ако ни једна друга трансакција нема локот читања или локот писања над x.

Две акције над истим објектом података биће у конфликту (сукобиће се) ако је најмање једна од њих писање. Логика која стоји иза овога је да редослед извршавања конфликтних операција прави разлику, због тога што мења или вредност прочитану од стране трансакције (јер операција читања чита различите вредности у зависности на то да ли се извршава пре или након операције писања) или коначне вредности податка (јер промена редоследа две операције писања над истим подацима мења коначну вредност податка). Као што видимо редослед операција је важан и због тога је битно да се тај ред контролише.

Логика која стоји иза закључавања је та да се избегну мешања између трансакција коришћењем конфликтних локота да би се синхронизовао ред извршавања конфликтних операција. Ако трансакција држи локот читања онда друга трансакција не може поставити локот писања чиме се избегава истовремено извршавање конфликтне операција писања. Ово слично функционише и за конфликтне локоте писања.

Иако је концепт закључавања једноставан, његови ефекти на перформансе и тачност могу бити комплексни, контрапродуктивни и тешки за предвидети. Изградња робусне апликације за обраду трансакција захтева добро разумевање закључавања.

Закључавање утиче на перформансе. Када трансакција постави локот то одлаже друге трансакције које треба да поставе конфликтне локоте. Када све остало остаје исто, што се више трансакција извршава истовремено веће су шансе да ће се такво одлагање догодити. На учесталост и трајање таквих одлагања може се утицати дизајном трансакције, организацијом базе података и дистрибуираним трансакционим и системом база података. Да би разумели како да минимизујемо овај утицај на перформансе морају се разумети механизми закључавања и како се они користе и како ови механизми и сценарији у којима се примењују утичу на перформансе.

Закључавање такође утиче и та тачност. Иако закључавања обично асоцирају на тачност, не доводе сва закључавања до тачних резултата. На пример резервисање приступа подацима пре него што им заиста приступимо изгледа да успешно елиминише могућност да се трансакције међусобно мешају. Ипак уколико је циљ остваривање серијабилности онда једноставно закључавање података пре него што им приступимо није сасвим довољно. Такође и тајминг операције откључавања је такође важан.

2.6.1 Основна закључавања

Неки од основних типова закључавања која су подржана од стране Мајкрософт SQL Server су:

- Дељено закључавање (енгл. Shared locks (S)) – користе се код извршавања операција само за читање над базом података. Ресурси се закључавају како би била омогућена Select наредба, али не и измене.
- Искључиво закључавање (енгл. Exclusive locks (X)) – користе се код операција као што су Insert, Update, и Delete наредбе, које модификују податке и захтевају искључиво закључавање.
- Намерна закључавања (енгл. Intent locks) – представљају хијерархију закључавања. Типови намерних закључавања су намерно дељено закључавање (IS), намерно искључиво закључавање (IX) и дељено са намерним закључавањем (SIX).
- Апдејт закључавање (енгл. Update (U)) – ово закључавање се обично поставља на страницу пре него што се изврши измена. Када је SQL Server спреман да измени страну ово закључавање се промени у искључиво закључавање странице.
- Шема закључавање (енгл. Schema) – користи се да спречи да се табела или индекс користе у другој сесији како не би били избрисани или њихова шема измењена. Када је ресурс закључан са шема закључавањем објекту се не може приступити.
- Булк апдејт закључавање (енгл. Bulk Update (BU)) – Користи се да спречи остале процесе да приступе табели док се извршава операција опсежног очитавања.

2.6.2 Двофазни протокол закључавања

Двофазни протокол закључавања (енгл. two-phase locking) подразумева да се извршавање сваке трансакције састоји из две фазе. Фазе закључавања објекта и фазе откључавања објекта и те две фазе се извршавају једна за другом (серијски). У фази закључавања нема ни једне радње откључавања објекта док у фази откључавања више нема радњи закључавања (наравно обе фазе укључују и друге операције као што су читање, упис, итд.). Фаза откључавања објекта започиње првом радњом откључавања [9]. Поштовање овог протокола гарантује да се проблеми конкурентности као што је проблем изгубљених ажурирања и остали неће јавити.

2.6.2.1 Стриктни двофазни протокол закључавања

Овај протокол се придржава двофазног протокола закључавања али се у овом протоколу искључиви локоти ослобађају тек након краја трансакције. На овај начин се гарантује да су сви објекти измењени од стране трансакције која још није потврдила свој измене закључани искључивим локотом све док се трансакција не изврши до краја онемогућујући тако друге трансакције да прочитају измењене податке који нису потврђени. Стриктни двофазни протокол се састоји из следећих корака:

1. Трансакција која жели да прочита неку врсту мора прво над њом да постави дељиви локот (S).
2. Трансакција која жели да ажурира неку врсту мора прво да над њом постави искључиви (X) локот. У случају да трансакција већ држи локот (S) над том врстом, трансакција мора да захтева унапређење локота S у X.
3. Ако је захтев за постављањем локот од стране трансакције T_2 одбијен јер је у конфликту са већ постављеним локот од стране трансакције T_1, трансакција T_2 иде у стање чекања. Трансакција T_2 чека док трансакција T_1 не ослободи локот (систем мора да гарантује да трансакција T_2 неће заувек остати у стању чекања).
4. Искључиви локот се задржава до краја трансакције (наредбе Commit или Rollback). Дељиви локот се задржава најдуже до краја трансакције.

Кажемо да је извршавање датог скупа трансакција коректно ако је серијабилно, то јест ако производи исти резултат као серијско извршавање истог скупа трансакција. Ефекат закључавања јесте да исфорсира серијабилност трансакција. Важи следећа теорема: ако све трансакције примењују двофазни протокол закључавања тада су сви могући распореди извршавања трансакција серијабилни.

2.6.3 Мртви чворови

Када се две или више трансакција такмиче за исти локот у конфликтним ситуацијама неки од њих ће постати блокирани и морају да чекају друге трансакције да ослободе своје локоте. Понекад се догађа да се сет трансакција блокира међусобно, свака од њих је блокирана а у исто време блокира друге трансакције. У овом случају ако ни једна трансакција не може да се настави осим ако систем не интервенише каже се да су трансакције у мртвом чвору.

Претпоставимо ситуацију са слике 5. где трансакција T_1 поставља локот читања на x а онда трансакција T_2 поставља локот читања на y. Након тога када T_2 захтева локот писања над x она је блокирана чекајући T_1 да ослободи локот читања који је поставила. Када T_1 захтева локот писања над y она је такође блокирана чекајући T_2 да ослободи свој локот читања. Како свака трансакција чека ону другу ни једна трансакција не може да настави са извршавањем и кажемо да су трансакције у мртвом чвору.

Мртви чвор је начин на који двофазно закључавање детектује несеријабилна извршавања. У моменту када се догоди мртви чвор не постоји могући ред којим би се извршиле преостале операције на начин на који би то довело до серијабилног извршавања. У датом примеру на слици 5. када T_1 и T_2 добију захтеване локоте читања имамо делимично извршење $R_1[x]R_2[y]$. Постоје само два начина да завршимо извршавање, $R_1[x]R_2[y]W_1[y]W_2[x]$ или $R_1[x]R_2[y]W_2[x]W_1[y]$ од којих ни један није серијабилан.

$R_1[x]$

Подаци	Постављен локот	Захтеван локот
x	T_1, читање	
y		

а.

$R_1[x]R_2[y]$

Подаци	Постављен локот	Захтеван локот
x	T_1, читање	
y	T_2, читање	

б.

$R_1[x]R_2[y]W_2[x]$-{блокирана}

Подаци	Постављен локот	Захтеван локот
x	T_1, читање	T_2, писање
y	T_2, читање	

в.

$R_1[x]R_2[y]W_2[x]$-{блокирана}$W_1[y]$-{блокирана}

Подаци	Постављен локот	Захтеван локот
x	T_1, читање	T_2, писање
y	T_2, читање	T_1, писање

г.

***Слика 5**. Извршавање које доводи до мртвог чвора*

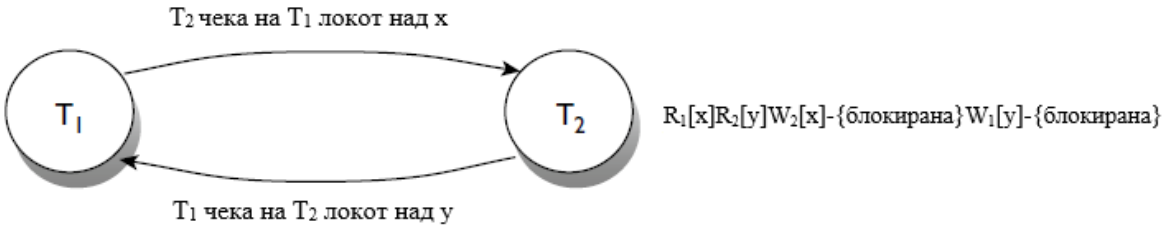
Једном када се догоди мртви чвор, једини начин да трансакције које се нађу у тој ситуацији направе прогрес јесте да се бар једна од њих одрекне свог локота који блокира другу трансакцију. Једном када трансакција уклони локот, правило двофазног закључавања каже да она не не може поставити више ни један локот. Како се не би прекршило правило двофазног закључавања трансакција која је одустала од локота не може поставити нове локоте због чега она сама не може да се изврши и менаџер трансакција може потпуно прекинути њено извршење или је поново покренути од почетка. Једини начин да се разреши мртви чворе је да се прекине извршење једне од укључених трансакција.

2.6.3.1 Детекција мртвог чвора

Постоје две технике које се најчешће користе да се детектује мртви чвор: временски орјентисана детекција и детекција базирана на графу. Временски орјентисана детекција претпоставља да је дошло до мртвог чвора кад код је нека трансакција блокирана дужи временски период. Она користи период чекања који је доста дужи него време извршења већине трансакција (нпр. 15 секунди) и прекида извршење било које трансакције која је блокирана дуже од тог времена. Главна предност овог приступа је да је једноставан и лак за имплементацију а функционише и у распрострањеним окружењима без додатне сложености. Ипак има два недостатка. Прво може прекинути трансакцију која није заиста у мртвом чвору. Ова грешка додаје одлагање трансакцији која је непотребно прекинута јер сада мора да се покрене од почетка. Друго може дозволити да мртав чвор постоји превише дуго. На пример ако се мртви чвор догоди након једне секунде од почетка извршавања трансакције он неће бити детектован док не истекне период чекања.

Алтернативни приступ, који се назива детекција базирана на графу, експлицитно прати ситуације чекања и периодично их проверава да ли су узроковани мртвим чвором. То се ради креирајући граф чекања чији су чворови модел трансакција а гране модели ситуација чекања. Ако трансакција T_1 не може да добије локот због тога што конфликти локот држи трансакција T_2 онда ту је грана $T_1 \rightarrow T_2$ значи T_1 чека на T_2. У општем случају менаџер података креира гране $T_i \rightarrow T_к$ кад год је трансакција T_i блокирана локотом који поседује

трансакција $T_к$. Грана се брише када се T_i одблокира. Долази до мртвог чвора када год граф има циклус то јест када низ грана поново се врати на почетни чвор. Пример мртвог чвора у којем учествују две трансакције $T_1 \rightarrow T_2 \rightarrow T_1$ можете видети на слици 6. У мртвом чвору може учествовати више трансакција па можемо имати ситуацију $T_1 \rightarrow T_7 \rightarrow T_4 \rightarrow T_2 \rightarrow T_1$.



Слика 6. *Граф базиран на чекању*

Свака ново-додата грана у графу може проузроковати циклус. Због тога менаџер података мора проверавати да ли је дошло до циклуса (мртвог чвора) када год дода нову грану. Док је то свакако исправан начин, такође је могуће да се провера мртвих чворова врши ређе на пример на сваких неколико секунди. Мртви чвор неће нестати сам од себе тако да нема штете у само периодичном проверавању. Вршењем провере периодично систем смањује трошкове детекције мртвих чворова. Као и временски орјентисана детекција то омогућава неким мртвим чворовима да прођу недетектовани дуже него што је неопходно али за разлику од временски орјентисане детекције сви детектовани мртви чворови су заиста мртви чворови.

2.6.3.2 Одабир жртве

Након што је детектован мртви чвор коришћењем детекције помоћу графа, мора се прекинути извршење једне од трансакција у циклусу. Та трансакција се назива жртвом. Као све трансакције у мртвом чвору, жртва је блокирана. Трансакција сазнаје да је жртва тако што прима поруку од операције да је блокирана која каже да јој је прекинуто извршавање. Сада је на апликацији која је издала операцију да одлучи како да се настави даље. Обично трансакција се поново покреће од почетка најчешће са кратким одлагањем да би се дало времена другим апликацијама у циклусу да заврше своје извршавање да не би дошло до мртвог чвора поново.

Постоји више критеријума који детектор мртвог чвора може користити. Он може изабрати једну трансакцију у мртвом чвору која:

1. Затвара циклус мртвог чвора - ово је можда и најлакше да се идентификује с обзиром да је то трансакција која је проузроковала мртви чвор.
2. Има најмањи број локота – ово се узима како мера количине посла који трансакција обавља. Бира се трансакција која је урадила најмању количину посла.

3. Генерисала најмању количину записа у лог фајл – како трансакција генерише лог упис за свако ажурирање које изврши ова трансакција је вероватно најпогоднија за прекид извршења.
4. Има најмањи број локота писања – ово је још један начин избора трансакције која је најпогоднија за прекид.

Уместо да сам детектор мртвог чвора одлучује о жртви трансакцији сама апликација може изабрати једну. Оракле базе података подржавају изјаве које генерише откривање мртвог чвора и враћају грешку остављајући тако апликацији да одлучи коју трансакцију ће прекинути.

Неки системи омогућавају да сама трансакција има утицај на избор жртве. У мајкрософт SQL server може поставити приоритет трансакције. Трансакција може рећи "SET DEADLOCK_PRIORITY LOW" или "SET DEADLOCK_PRIORITY NORMAL". Ако једна или више трансакција имају ниво низак приоритета једана од њих ће бити изабрана као жртва. Међу апликацијама које приоритет чини квалификованим да буду изабране за жртву систем бира ону коју је најлакше прекинути.

Још једна ставка на коју треба обратити пажњу приликом избора жртве је избегавање цикличног ресетовање. Циклично ресетовање је када се трансакције непрестано ресетују због мртвог чвора чиме се онемогућује њихово извршавање. Један од начина на који долази до ове ситуације је када се старија трансакција увек бира као жртва. На пример претпоставимо да се покрене трансакција T_1, након тога покрене се трансакција T_2, након чега долази до мртвог чвора између T_1 и T_2. Како је T_1 старија она је жртва и она се прекида и поново покреће. Убрзо након тога T_1 и T_2 се поново налазе у мртвом чвору али овога пута T_2 је старија (због тога што је T_1 ресетована након што је T_2 покренута) па је она жртва. T_2 се прекида и поново покреће и затим поново улази у мртви чвор са T_1 и тако у недоглед.

Један од начина да се избегне циклично рестартовање је да бира најмлађа трансакција за жртву. Ово обезбеђује то да се најстарија трансакција у систему никада неће ресетовати у случају мртвог чвора. Трансакција се може ресетовати у више наврата у случају лоше среће, ако је она увек најмлађа трансакција у циклусу, али ово је врло мало вероватно.

Овај проблем се може избећи ако се трансакцији поставља исто време покретања сваки пут када је ресетована тако да ће у једном тренутку она бити најстарија у систему. Ово захтева да менаџер подацима прихвата време покретања трансакције као улазни параметар да би покренуо операцију што врло мали број менаџера података подржава.

На крају апликација или трансакциони слој обично обезбеђује решење пратећи број пута колико је нека трансакција поново покретана. Апликацији се пријављује грешка у случају да се нека а трансакција покрене превише пута без обзира да ли се ради о мртвом чвору или неком другом разлогу и од тог тренутка је на апликацији да одреди узрок зашто се нека апликација налази у мртвом чвору тако често.

2.7 Нивои изолованости трансакција

Један од главних извора загушења приликом истовременог извршавања трансакција су упити. То су захтеви који само читају податке у сврху подршке доношења одлука и извештавања. Упити се обично извршавају много дуже него трансакције и приступају великој количини података. Ако се они извршавају поштујући правило двофазног закључавања они често постављају велики број локота и задржавају те локоте дуг временски период. Ово често доводи до нетолерантног одлагања извршења трансакција. Да би се избегли ови проблеми много апликација једноставно одустаје од серијабилности за упите користећи мање строга правила закључавања. Ова правила се називају степеном изолације или изолационим нивоом и она су прописана SQL стандардима тако да их нуде готово све SQL базе података.

Једно од правила се назива read commited (понекад се назива и другим степеном изолације (енгл. Degree 2)). Ако се упит извршава са изолационим нивоом read commited онда менаџер података држи локот читања над подацима само док упит чита податке. Чим су подаци прочитани он ослобађа локот.

Бенефит read commit изолационог нивоа огледа се у перформансама. Неке симулације мешовитих радних оптерећења показале су да се пропусност повећава и до три пута у односу на двофазно закључавање. За нека радна оптерећења побољшање перформанси је чак и веће.

Read commited изолациони ниво је слабији од двофазног закључавања који захтева од трансакције да држи локот читања док не добије све захтеване локоте. Read commit изолациони ниво обезбеђује да упит само чита податке које је произвела трансакција која се извршила. Због тога ако једна активна трансакција ажурирања мења податке упит неће бити у могућности да постави локот док та трансакција није извршена или прекинута. Ипак ово не обезбеђује серијабилност. На пример ако упит чита податке x и y, и једна трансакција ажурирања врши ажурирање ових података могућ је следећи сценарио:

- Упит чита податке x и након тога ослобађа локот над x.
- Након тога трансакција врши ажурирање x и y, извршава се и ослобађа своје локоте.
- Онда упит чита податке y.

У овој ситуацији упит је податке x прочитао пре ажурирања а податке y након ажурирања. Овакав резултат је немогућ приликом серијског извршавања.

Под изолационим нивоом read commited трансакција која чита исте податке два пута може прочитати различите вредности за сваку операцију читања. Ово може да се догоди због тога што друга трансакција може ажурирати податке између два читања. Из овог разлога кажемо да изолациони ниво read commited дозвољава непоновљиво читање (енгл. nonrepeatable read). Ово име је мало погрешно с обзиром на то да се трансакцији дозвољава поновно читање, само што може добити различите вредности сваког пута када изврши читање.

Мало јача верзија од изолационог нивоа read commited назива се ниво стабилности курсора (енгл. cursor stability) који је понуђен од појединих SQL система база података. У SQL-у резултат упита је сет редова који се враћа програму како курсор. Програм може прегледати резултат упита понављајући се помоћу курсора један ред по један. Користећи ниво изолације read commited програм поставио локот читања на ред пре читања и уклонио га одмах након што заврши. Користећи ниво стабилности курсора програм задржава локот читања мало дуже, док не затражи прелазак у следећи ред користећи операцију SQL fetch. У овој тачки систем базе података прво ослобађа локот над тренутним редом а онда стиче локот на следећем реду. Стога, ред који курсор тренутно идентификује је стабилан (на њему је постављен локот читања) док га програм гледа – одатле и потиче термин стабилност курсора.

Коришћењем курсора стабилности, програм може ажурирати тренутни ред курсора без опасности да ће се умешати неке друге трансакције и ометати његово извршење. Како држи локот читања над редом када изда ажурирање менаџер података може променити локот читања у локот писања. За разлику од тога ако програм користи read commited ниво изолације он ће склонити локот читања са реда одмах након што се изврши читање података и пре него што се захтева операција писања. Због тога два програма могу извршити ову радњу истовремено. Оба ће прочитати податке, онда ће оба ослободити своје локоте читања а након тога ће оба ажурирати податке што ће довести до тога да један програм препише податке преко података које је унео други. Ниво курсора стабилности онемогућује овакав исход.

Због користи које изолациони ниво read commited има на перформансе многе SQL базе података га постављају као свој основни изолациони ниво па апликација мора користити посебне кључне речи да би добила серијабилно понашање. Иако резултати на овом изолационом нивоу могу бити нетачни за крајњег корисника то није много битно. Ово посебно важи за упите који се често користе да дају кратак преглед базе података. Ако се база ажурира често онда мале разлике узроковане грешкама серијабилности нису много битне због тога што ће резултати упита застарити већ одмах након што буду презентовани кориснику. Чак шта више како се ово користи у сврху помоћи приликом доношења одлука онда није битно што подаци нису апсолутно тачни.

Упите је могуће покренути у још слабијем начину закључавања у којем нема никаквих локота. Овај ниво изолације се назива и read uncommited или прљаво читање (енгл. dirty read) или први степен изолације (енгл. Degree 1). У овом случају упит може извести прљаво читање приликом чега чита несталне податке (енгл. uncommited data) – ово су подаци који могу бити обрисани у случају да се трансакција не изврши до краја. Ово ће умањити одлагање извршења упита мање чак и од нивоа изолације read commited. Цена ово га је још већа недоследност у подацима који се читају.

Важно је то да иако упити користе неки од изолационих нивоа read commited или read uncommited трансакције ажурирања и даље могу користити двофазно закључавање стога и даље бити серијабилне у односу између себе. У ово случају стање базе података је и даље конзистентно у смислу да је резултат серијабилног извршавања трансакција. Само упити могу читати несталне верзије тог стања.

Већина SQL система база података нуди могућност покретања ажурираних трансакција користећи ниво изолације read commited или чак read uncommited извршавајући исказ да се постави изолациони ниво. Покретање трансакције на једном од ова два изолациона нивоа крши правило двофазног закључавања и може довести до несеријабилног извршавања. Иако перформансе могу бити боље резултати могу бити нетачни. На пример ако два трансакције обе читају и пишу вредност x (нпр. желе да увећају вредност x) онда ниво изолације read commited ће омогућити да обе трансакције прочитају вредност x пре него што било која од њих упише x. Ово несеријабилно извршавање скоро сигурно бити нежељено за корисника с обзиром да узрокује да се једно ажурирање губи.

Када користимо терминологију степана изолације серијабилност се често карактерише као трећи степен (енгл. Degree 3). Овај степен изолације се често назива и поновна читања (енгл. repeatable reads) због тога што за разлику од нивоа курсора стабилности читање података читање податак више пута враћа исту вредност јер је локот читања задржан током целог извршења трансакције. Највиши ниво изолације се назива серијабилност (енгл. serializable) и означава баш то: извршавање трансакција мора бити еквивалентно серијалном извршавању. У табели 3. имамо сумиране информације нивоа изолованости.

Већина система база података омогућава администратору базе података да постављи ниво изолације по бази података. Системи база података и слој трансакција такође обично дозвољавају програмеру апликације да сам одреди степен изолације за одређени трансакциони програм без обзира на основни ниво који је постављен од стране базе података. Неки системи база података дозвољавају трансакцијама да покрену операцију промене изолационог нивоа у току самог извршавања.

Табела 3. Степени изолације

Степен изолације	ANSI SQL термин	Понашање
1	Read uncommited	Не поставља локоте читања
2	Read commited	Чита само податке које је произвела трансакција која се извршила
3	Serializable	Серијабилно

Доста система база података нуде степене изолације који су нижи од серијабилног али се не уклапају у потпуности ни у један ANSI SQL стандард. На пример Мајкрософтов SQL Server нуди могућност закључавања названу READPAST. Ако трансакција користи ниво изолације read commited и назначи опцију READPAST у SQL изјави онда ће изјава радије игнорисати све редове који имају локот писања него што ће чекати да се ови локоти ослободе. Логика која стоји иза овога је да с обзиром да апликација користи ниво изолације read commited она не очекује апсолутно тачне резултате по сваку цену. У неким случајевима исплати се избећи одлагање које проузрокује чекање да се локоти писања ослободе једноставним прескакањем закључаних редова.

2.8 Избегавање фантомских читања

У стандардном моделу закључавања који се користи у наредном примеру операције додавања (енгл. insert) и брисања (енгл. delete) су моделиране као операција писања и нећемо их третирати засебно. Ипак у оквиру система менаџер података мора бити посебно опрезан са овим операцијама да би избегао несеријабилне резултате.
Да би смо преставили потенцијални проблем узмимо у разматрање базу података са слике 7. Табела Рачуни има ред за сваки рачун и укључује број рачуна, локацију филијале и стање на том рачуну. Табела Средства има укупан баланс за све рачуне у свакој филијали.

Рачуни

Број рачуна	Локација	Стање
1	А	50
2	Б	50
3	Б	100

Средства

Локација	Укупно
А	50
Б	150

Слика 7. *База података Рачуни како би се илустровала фантомска читања*

Сада претпоставимо да се следећи низови операција извршавају од трансакција T_1 и T_2:

1. T_1: Чита Рачуни 1, 2, 3.
2. T_1: Идентификује редове у табели Рачуни где је Локација = Б (редови 2 и 3) и рачуна суму њихових стања (=150).
3. T_2: Додаје нов ред у табелу Рачуни [4, Б, 100].
4. T_2: Чита Укупно за локацију Б у табели Средства (враћа 150).

5. T_2: Уписује у Средства [Б, 250].
6. T_2: Завршава извршавање (Commit).
7. T_1: Чита табелу Средства за локацију Б (враћа 250).
8. T_1: Завршава извршавање (Commit).

Трансакција T_1 врши ревизију рачуна на локацији Б. Прво чита све рачуне у табели Рачуни (корак 1) и сабира стања на локацији Б (корак 2) и након тога проверава укупно стање (колона Укупно) у табели Средства за локацију Б (корак 7) да утврди да се подударају. Они се не подударају јер T_1 не види нову колону у табели Рачуни коју је додала T_2, али види ажурирану вредност у табели Средства за локацију Б која укључује вредност уноса T_2.

Ово извршавање није серијабилно. Да се T_1 и T_2 извршавају серијално T_1 би или видела T_2 ажурирања и табеле Рачуни и табеле Средства или не би видела ни једно од та два ажурирања. А приликом оваквог извршења као у примеру она види ажурирање T_2 над табелом Средства али не види ажурирања у табели Рачуни.

Проблем је ред у табели Рачуни [4, Б, 100] који је T_2 додала. T_1 не види овај ред када чита табелу Рачуни али види ефекат T_2 на табелу Средства где је додала 100 на укупан баланс. Ред у табели Рачуни [4, Б, 100] се назива фантомско читање због тога што је невидљив приликом дела извршења T_1.

Чудна ствар у вези са овим извршавањем је што се чини да је оно дозвољено од стране двофазног протокола закључавања. У наставку додате су операције закључавања које захтева двофазни протокол закључавања:

1. T_1: Закључава редове 1, 2 и 3 у табели Рачуни. Чита Рачуни 1, 2, 3.
2. T_1: Идентификује редове у табели Рачуни где је Локација = Б (редови 2 и 3) и рачуна суму њихових стања (=150).
3. T_2: Додаје нов ред у табелу Рачуни [4, Б, 100] и закључава га.
4. T_2: Закључава ред Локације Б у табели средства. Чита колону Укупно за ред Б у табели Средства (враћа 150).
5. T_2: Уписује у Средства [Б, 250].
6. T_2: Завршава извршавање (Commit) и откључава ред Б у табели Средства и ред [4, Б, 100] у табели Рачуни.
7. T_1: Закључава ред Локације Б у табели Средства. Чита табелу Средства за локацију Б (враћа 250).
8. T_1: Завршава извршавање (Commit) и откључава ред Б у табели Средства и редове 1, 2, 3 у табели Рачуни.

Нажалост као што видимо протокол двофазног закључавања не гарантује серијабилност када је укључена операција додавања. Постоје нека скривена понашања која захтевају постављање додатног закључавања и то није приказано у извршавању. Све се своди на то како T_1 зна да је у табели Рачуни тачно три реда. Ту мора да постоји нека врста структуре података која ће то да каже: неки маркер за крај фајла, бројач броја редова у фајлу, листа

показивача на редове у фајлу или нешто тако. Пошто се чита та структура података да би се утврдило да треба да се прочитају редови 1, 2 и 3 неопходно је поставити и локот читања на ту структуру података. Пошто T_2 додаје ред у табелу Рачуни она такође мора да погледа ту структуру података и то у моду писања да би могла да је ажурира. T_2 би била спречена да то уради због тога што T_1 већ има постављен локот читања на ту структуру података тако да се приказано извршење не би могло догодити.

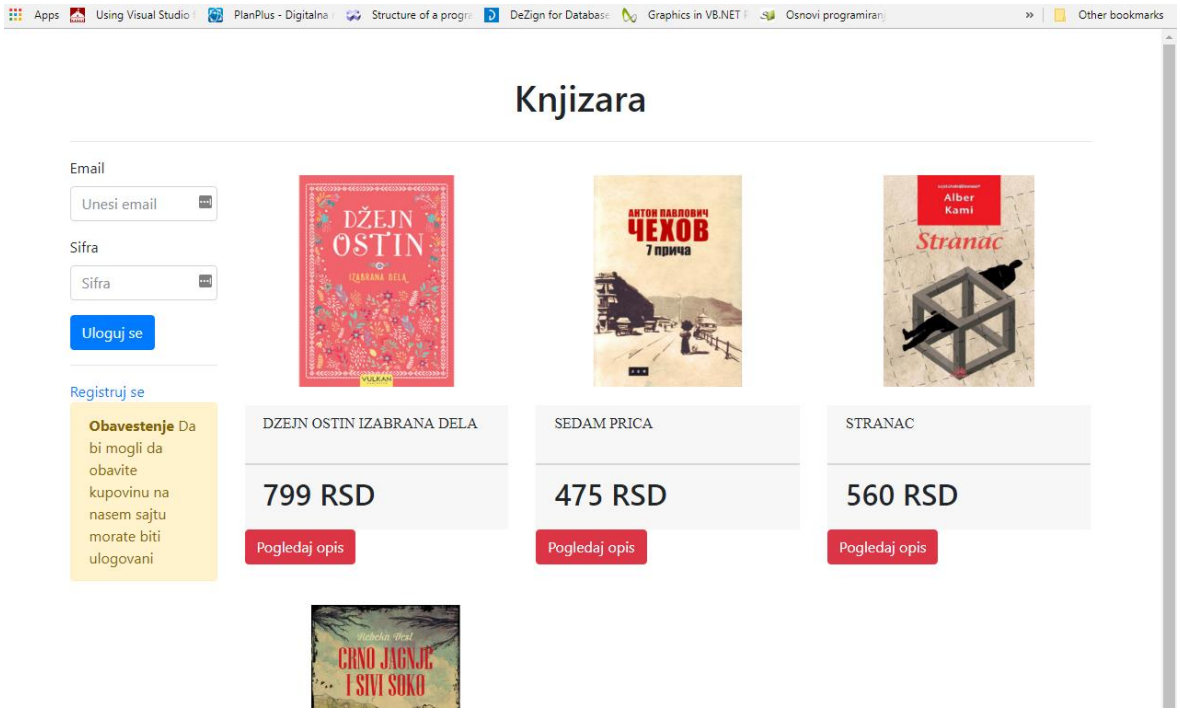
Дакле проблем фантомског читања није проблем под условом да менаџер података поставља локоте на све дељене податке којима приступа укључујући и системске структуре које користи интерно у име трансакционих операција.

3 Пројектни задатак

Овај пројектни задатак има за циљ да покаже практичну употребу трансакција и обраде трансакција у реалном окружењу. Са растућим трендом онлајн плаћања у свету и код нас за потребе задатка симулирано је радно окружење е-продавнице да би се што реалније представио начин функционисања обраде трансакција, начин на који се користе и предности које нам доноси такав систем.

3.1 Опис продавнице

Приликом израде овог задатка највећи фокус је стављен на појашњење система за обраду трансакција и начина његовог функционисања тако да један комплексан систем какав је е-продавница није у потпуности направљен већ су креирани неки његови делови како би се што реалније симулирало једно окружење у којем се одвијају трансакције. На пример нашој продавници недостаје администраторско подручје, иако је остављен простор за његову изградњу то би одузело превише времена а како није неопходно за функционисање саме продавнице и не утиче на закључке и примере које желимо да покажемо одлучили смо да га изоставимо. Такође додавање таквих функција би повећало сам изворни код који је и овако обиман што би отежало сналажење приликом његовог прегледа, а као што је речено не би смо добили ништа у смислу квалитета примера који су сврха самог задатка. Иако наша е-продавница можда нема могућност додавања нових производа и слично са стране крајњег корисника она је потпуно функционална. На слици 8. можемо видети почетну страну е-продавнице за продају књига Књижара. На почетној страни се налазе књиге које су у понуди, форма за пријављивање и линк ка регистрационој форми.



Слика 8. *Почетна страна е-продавнице Књижара*

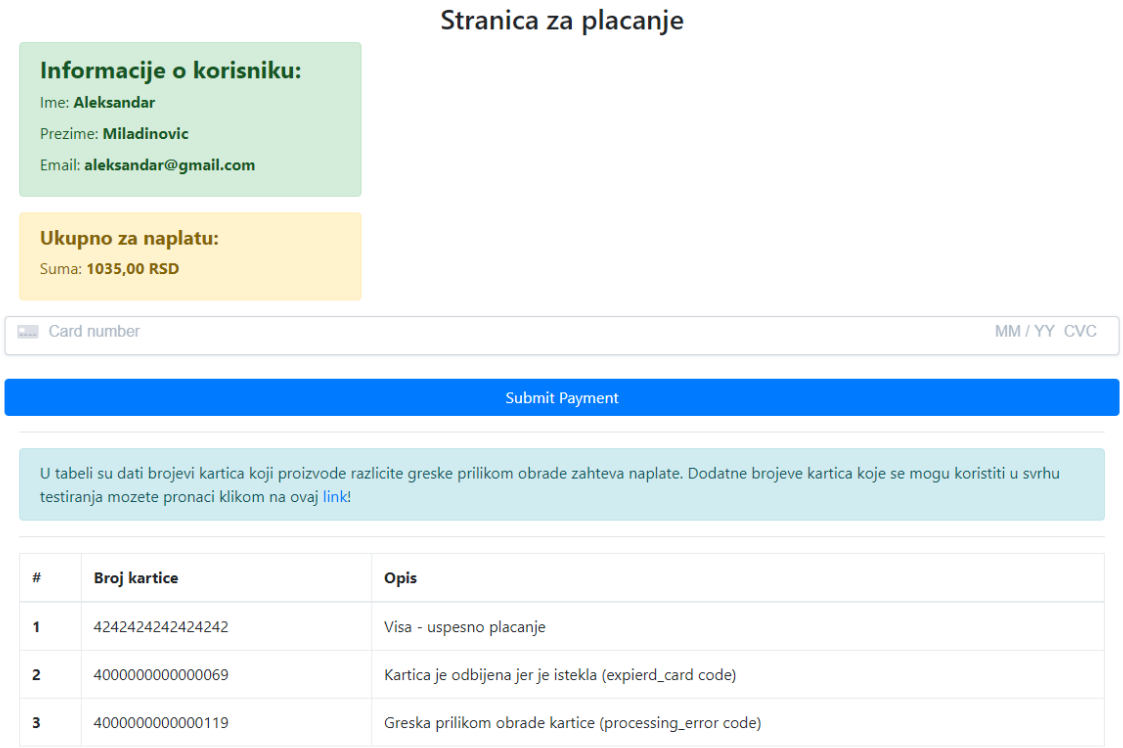
Корисник који жели купити наше производе (књиге) мора бити пријављен. Када се корисник пријави онда се поред дугмета Погледај опис (Почетна страна ако смо на страници о детаљима производа коју можемо видети на слици 9.) појављује друго дугме Додај у корпу.

Слика 9. *Страница са детаљима производа*

Поред ове две странице корисник наше е-продавнице се сусреће са још две да би могао да несметано да функционише. То су страница корпа која представља електронску корпу у коју се смештају производи које корисник жели да купи. Када се корисник нађе на овој страници која се може видети на слици 10 корисник може обрисати производе из корпе које не жели да купи, ажурирати количину производа које жели да купи, може се вратити на почетну страницу и наставити своју куповину или може ако је задовољан својим избором може наставити на страницу за плаћање коју можемо видети на слици 11.

Слика 10. *Електронска корпа е-продавнице Књижара*

Страница за плаћање је врло једноставна, на њој корисник може видети основне податке о себи, укупну суму за наплату, форму која се састоји од поља за унос података са кредитне картице и дугмета да се изврши плаћање (енгл. Submit Payment) ову форму (поље за унос и дугме) обезбеђује Страјп (енгл. Stripe) апликација за обраду онлајн плаћања и она се брине о форми уноса, тачности података кредитне картице итд. Како се овај пројекат користи у показне и сврхе тестирања испод ове форме додата је табела у којој можемо видети бројеве картица које симулирају различите грешке које се могу догодити у реалном окружењу. Да би имали увид у то шта се дешава са трансакцијом која се не изврши до краја.



Слика 11. *Приказ странице за плаћање*

Поред ове четири странице које нас воде до извршења трансакције постоји још известан број страница које служе за преусмеравање и обавештење корисника.

3.2 Израда и покретање е-продавнице

Приликом израде пројектног задатка коришћене су различите технологије које су или неопходне за израду самог задатка или су олакшале његову израду.

Две основне технологије које су коришћење су PHP и MySQL ипак пре свега да би смо уопште покренули било коју апликацију у локалу неопходно је да имамо инсталиран веб сервер у нашем случају Apache. Ове три технологије се могу наћи у бесплатној веб сервер платформи XAMPP која је отвореног кода. XAMPP је скраћеница од Cross-Platform (X), Apache (A), MariaDB (M), PHP (P) и Perl (P). То је једноставна дистрибуција која је креирана од стране Apache и која чини веома једноставно за програмере да креирају локани веб сервер за сврхе тестирања и развоја софтвера. Све што је потребно да би се поставио веб сервер је серверска апликација (Apache), база података (MySQL) и скриптни језик (PHP) што је све укључено у један инсталациони фајл. XAMPP је такође и крос платформа што значи да ради подједнако добро на различитим системима Линукс, Виндоус и Мек. Како већина актуелних решења користе исте компоненте као XAMPP прелазак са локалног тест сервера на прави сервер је прилично лако.

Након инсталације XAMPP платформе и Apache сервера логичан избор програмског језика био је PHP (PHP: Hypertext Preprocessor) који је већ интегрисан у XAMPP окружење. PHP је специјализовани скриптни језик првенствено намењен за израду динамичког веб садржаја и изводи се на страни сервера.

За систем за управљање базом података изабран је MySQL који је такође део XAMPP платформе али и такође испуњава све неопходне услове за израду пројектног задатка као што је подршка трансакција. Систем ради као сервер обезбеђујући вишекориснички интерфејс за приступ бази података.

За креирање директан приступ и вршење операција над базом података у фази развоја коришћен је phpMyAdmin. phpMyAdmin је алатка писана у програмском језику PHP за потребе администрирања MySQL система база података путем веба. Неке од основних операција које може да изводи су стварање и уклањање базе података, извођење операција над табелама и пољима, да извршава SQL упите, руководи кључевима и пољима итд.

Visual Studio Code је едитор који је коришћен приликом писања PHP скрипти, HTML, CSS JS фајлова. Visual Studio Code је едитор изворног кода развијен од стране мајкрософта. Укључује разне корисне ствари као што је подршка за дебаговање, означавање синтаксе, интелигентно допуњавање кода, форматирање кода итд. Врло корисна ствар која у многоме олакшава рад.

Веза између базе података и PHP скрипти на серверу се остварује помоћу PDO екстензије за PHP. PHP Data Object (PDO) је интерфејс за приступ бази података у PHP. PDO обезбеђује абстракциони слој приступа подацима што значи да без обзира која база података се

користи на страни сервера ми користимо исте функције да извршимо упите и преузмемо податке. Што значи да у случају да желимо да променим базу података у будућности тј. да пређемо са MySQL на SQL Server, једина ствар коју мењамо на сервер страни су основна подешавања у PDO конекцији као што су назив базе, корисничко име, шифра итд. Све остало остаје непромењено.

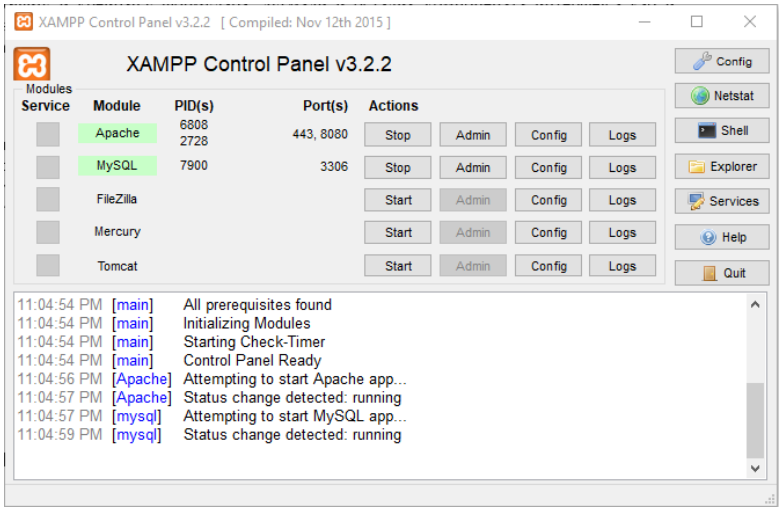
За обраду кредитне картице коришћења је услуга треће стране. Коришћен је софтвер компаније Страјп (енгл. Stripe) које обезбеђује испуњавање различитих захтеве који су неопходни да би се обављало онлајн плаћање као што су детекција преваре инфраструктура и техничка инфраструктура. Страјп је компанија која је прва развила јава скрипт решење за имплементацију обраде кредитних картица које се врло лако интегрише у постојеће окружење.

Поред набројаних технологија које су коришћене уз HTML и CSS можемо поменути да је коришћен још и бутстреп (енгл. Bootstrap) који представља бесплатни фронт-енд фрејмворк отвореног кода за креирање веб сајтова и веб апликација. Базиран је на HTML и CSS за типографију и креирању формулара, дугмади и осталих компонената интерфејса као и опционим јава скрипт додацима. Бутстреп у многоме олакшава програмирање изгледа саме апликације.

3.2.1 Покретање е-продавнице Књижара на рачунару

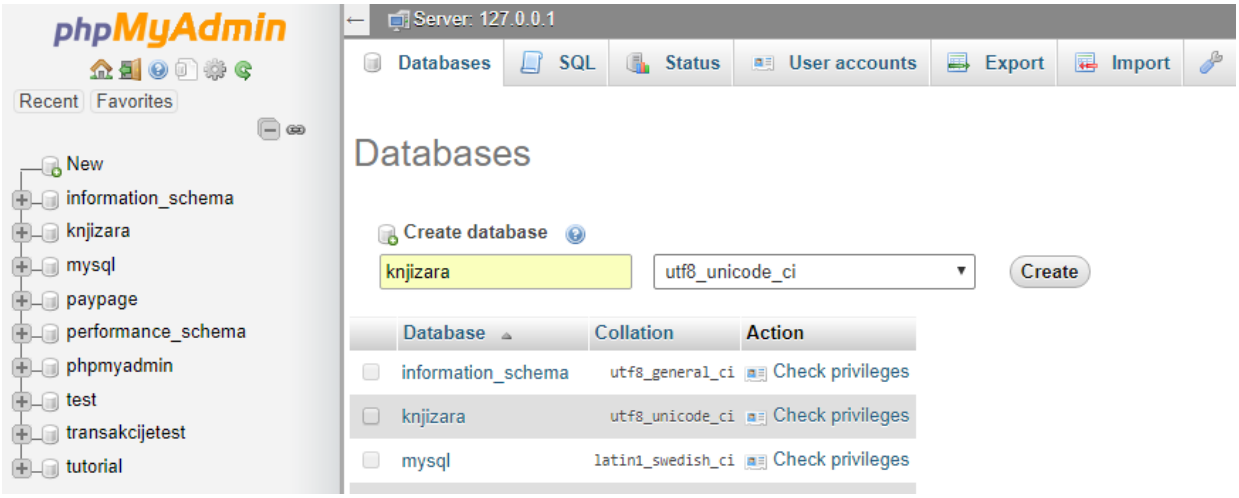
Да би сте могли да покренете е-продавницу књижара неопходно је да на свом рачунару имате инсталиран веб сервер (Apache) и MySQL за шта је најлакше да инсталирате XAMPP платформу. Постоје и друга решења као што је WAMPP али како је приликом израде задатка кориштена XAMPP платформа и да је то платформа коју можете покренути и на Линукс оперативном систему она је препоручена.

Након инсталације XAMPP платформе покрените је и у прозору покрените Apache веб сервер и MySQL. На слици 12 можете видети отворен XAMPP прозор.



Слика 12. *Изгледа XAMPP прозора са покренутим Apache веб сервером и MySQL-ом*

Да би подигли базу података knjizara на сервер неопходно је да ја фајл knjizara.sql увеземо преко phpMyAdmin администратора. Да би смо приступили phpMyAdministrator у претраживачу укуцамо localhost/phpMyAdmin, у случају да је предефинисани порт 80 заузет сервер ће користити порт 8080 и у том случају у прегледачу куцамо адресу localhost:8080/phpMyAdmin. Са леве стране у менију изаберемо опцију New и у главном делу екрана отвара се прозор у којем креирамо базу података (погледати слику 13). За име базе података уносимо knjizara и бирамо из падајућег менија Collation utf8_unicode_ci фонт и након тога кликнемо на дугме Create.



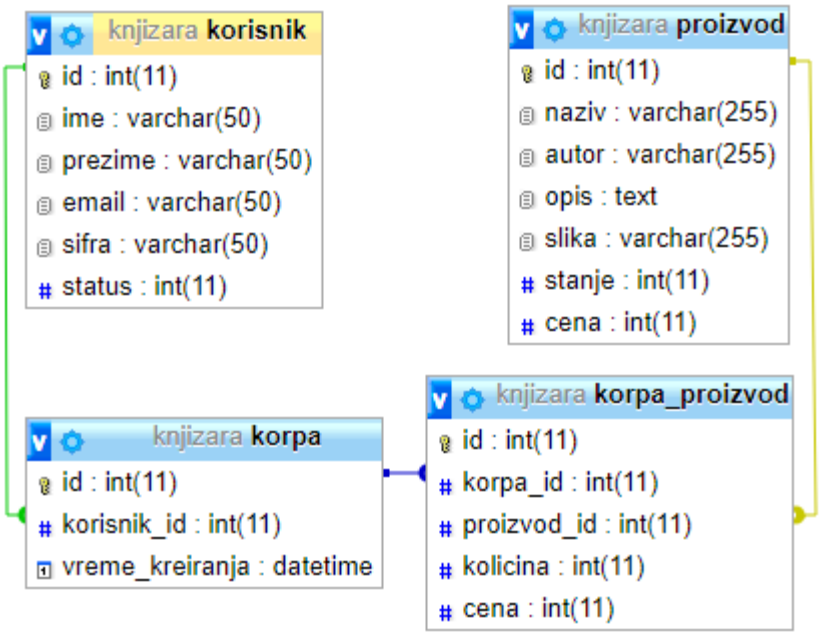
Слика 13. *Креирање базе података knjizara*

Након притиска на дугме Create у левом менију се појављује база knjizara у коју треба да увеземо податке из фајла knjizara.sql. Селектујемо базу knjizara и у десном прозору кликнемо на дугме Import. У новоотвореном менију кликом на дугме Choose file изаберемо фајл knjizara.sql и кликом на дугме Go у дну странице увозимо податке из фајла у базу података knjizara.

Избором на знак плус поред базе података knjizara можемо видети креиране табеле са подацима које су увежени из фајла knjizara.sql. Ту су четири табеле:

- korisnik која садржи све податке о регистрованим корисницима сајта
- korpa у коју се смештају подаци о е-корпама које су креиране
- korpa_proizvod табела у коју су смештени подаци о производима који су смештени у нашу е-корпу, количини производа и цени производа у моменту куповине.
- proizvod табела у којој се налазе подаци о производима који се могу купити на сајту

На слици 14 можемо видети ER дијаграм који детаљно приказује табеле и везе између табела у бази података knjizara.



Слика 14. *ER дијаграм базе података knjizara*

Након тога остало је још да фолдер knjizara копирамо у фолдер htdocs који у случају да ништа нисте мењали приликом инсталације XAMPP платформе у виндоус окружењу налази на локацији C:\xampp\htdocs. Након тога веб апликацију knjizara можемо покренути у претраживачу на локацији localhost/knjizara/ или localhost:8080/knjizara/ у случају да порт 80 није био слободан приликом инсталације XAMPP платформе. Фолдер књижара са свим .php фајловима и фајл knjizara.sql можете наћи на диску достављеном уз рад у фајлу Projektni Zadatak Aleksandar Miladinovic 2006-2015.rar.

3.3 Трансакција

Након што покрене нашу апликацију корисник ће се регистровати, пријавити, након неког времена одабрати производе које жели да купи и у једном моменту ће се наћи на страници за плаћање коју смо већ могли да видимо на слици 11. Након тога унеће податке о кредитној картици и кликнути на дугме Submit Payment и у том тренутку он ће покренути код који је најбитнији део пројектног задатка и који се бави трансакцијама и помоћу којег можемо стећи увид како и шта трансакције раде и зашто су важне.

Кликом на дугме Submit Payment корисник извршава форму payment-form преко које шаље податке корисника, суме за наплату и кредитне картице корисника странци charge.php која прихвата ове податке, обрађује их затим креира објекат $transakcija класе Transakcija и у асоцијативни низ $statusTransakcije смешта све податке које ће добити када изврши функцију објекта $transakcija→azurirajProizvode() којој прослеђује сређене податке. У низ $statusTransakcije се смештају све релевантне информације о току извођења функције azurirajProizvode() и грешкама које су се евентуално догодиле приликом њеног извођења. Ова функција почиње PDO командом за започињањем трансакције beginTransaction() и

завршава се или командом за успешно завршавање трансакције commitTransaction() или у случају неуспешног извршења командом rollBack().

Након што започне трансакцију функција azurirajProizvode() креира асоцијативни низ у који смешта податке о статусу трансакције следећом линијом кода:

```
//status transakcije
$statusTransakcije = array("status" => true, "poruka" => '',
            "proizvod_id" => 'xx', "customer_id" => 'xx', "kod_greske" => '10');
```

На крају извршења функције овај низ $statusTransakcije ће се као параметар вратити страници charge.php и на основу њега ће ова страница ће на основу њега обавестити корисника апликације о статусу трансакције и у случају неке грешке обавестиће корисника до каквог је проблема дошло.

Након тога функција проверава да ли су сви производи који су додати у корпу доступни на стању. То се врши тако што се за сваки производ у корпи селектује релевантно стање тог производа из табеле proizvod у којој постоји колона која садржи количину производа на стању. Затим упоређује тражену количину производа и количину производа на стању, уколико је тражена количина производа већа од количине на стању у низ $statusTransakcije се уписују подаци о промени статуса трансакције, производу којег нема на стању итд. Након тога уколико је неког од производа нема довољно на стању прекида се извршавање трансакције у низ се уписује порука да нема довољно производа на стању и као резултат функције се враћа низ $statusTransakcije. У самом коду то изгледа овако:

```
// proveri da li svih proizvoda ima na stanju
foreach($proizvodiIzKorpe as $pk){

    $id = $pk->proizvod_id;

    $sql = 'SELECT stanje FROM proizvod WHERE id=:id';
    $stmt = $this->pdo->prepare($sql);
    $stmt->execute(array(":id" => $id));
    $proizvod_stanje = (int) $stmt->fetchColumn();
    $stmt->closeCursor();

    if($pk->kolicina > $proizvod_stanje){
        $statusTransakcije["status"] = false;
        $statusTransakcije["kod_greske"] = 1000;
        $statusTransakcije['proizvod_id'] = $id;
    }
}

if ($statusTransakcije["status"] == false) {
    $this->pdo->rollBack();
    $statusTransakcije["poruka"] = 'Nemamo proizvod na stanju';
```

```
        return $statusTransakcije;
    }
```

Уколико свих производа има на стању функција наставља са извршењем и приступа ажурирању свих производа у табели производи умањујући их за потребну количину.

```
    // azuriraj stanje proizvoda
    foreach($proizvodiIzKorpe as $pk){

        $id = $pk->proizvod_id;
        $kolicina = $pk->kolicina;

        $sql_update_from = 'UPDATE proizvod
                            SET stanje = stanje - :kolicina
                            WHERE id = :id';
        $stmt = $this->pdo->prepare($sql_update_from);
        $stmt->execute(array(":kolicina" => $kolicina, ":id" => $id));
        $stmt->closeCursor();
    }
```

Када се сви производи успешно ажурирају приступа се наплати жељеног износа са кредитне картице. Да би се износ наплатио користимо код који је генерисан од софтвера компаније Stripe и чије услуге користимо за процесирање кредитне картице.

```
    // fajl obezbedjen od Stripe aplikacije
    require_once('vendor/autoload.php');

    \Stripe\Stripe::setApiKey('sk_test_XJjry1FP1zcOlSauqlHAat0M');

    //kreiraj korisnicke informacije Stripe
    $customer = \Stripe\Customer::create(array(
    "email" => $email,
    "source" => $token
    ));

    //podatci o naplati Stripe
    $charge = \Stripe\Charge::create(array(
    "amount" => $ukupnaSuma,
    "currency" => "usd",
    "description" => "Kupovina knjige",
    "customer" => $customer->id
    ));
```

Када се изврши наплата износа са кредитне картице уколико не дође до никакве грешке у низ $statusTransakcije уносимо информацију о идентификацији корисника апликације.

Завршавамо трансакцију наредбом commit() и као резултат функције враћамо низ $statusTransakcije.

```
        $statusTransakcije["customer_id"] = $charge->customer;
        // commit transakciju
        $this->pdo->commit();

        return $statusTransakcije;
```

Уколико пак дође до неке грешке приликом самог процесирања картице или неочекиваног прекида кода у catch делу try/catch блока можемо "ухватити" грешке ако су се догодиле. У случају да је грешке дошла приликом обраде кредитне картице користимо код који обезбеђује Stripe софтвер а уколико дође до насилног прекида кода користимо стандарде PDO изузетке.

```
    } catch (Stripe\Error\Card $e) {
        $this->pdo->rollBack();
        die($e->getMessage());
    } catch (PDOException $e) {
        $this->pdo->rollBack();
        die($e->getMessage());
    }
```

Када узмемо у обзир два горе наведена примера када се функција azurirajProizvod() успешно изврши до самог краја без неког непредвиђеног прекида или када се сама функција избори са случајем када немамо довољно производа на стању ми не можемо приметити разлику у резултатима у односу на то да ли користимо трансакције или не. У првом случају функција се успешно извршава до краја и свакако нема никакво непредвиђено понашање које може узроковати неконзистентне резултате. У другом случају функција се сама прекида пре него што уопште почне да врши неке промене над самим подацима у табели. Ипак у наставку су дата два примера која недвосмислено показују важност и корист од коришћења трансакција.

3.3.1 Проблем насилног прекида извршења програма

Да би смо што боље објаснили последице до којих насилан прекид извршења програма без коришћења трансакција може довести пратићемо промене над табелом proizvod у колони stanje у два случаја тј. када дође до насилног прекида извршења програма када не користимо трансакције и у случају када их користимо.

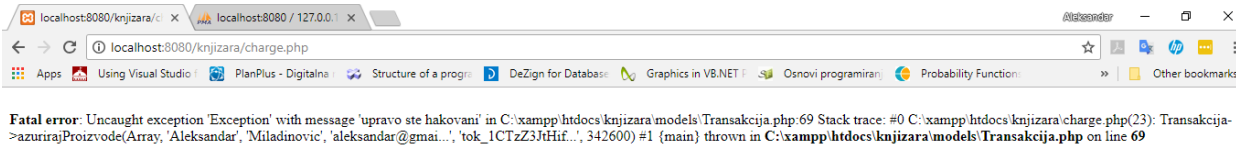
Да би смо могли да прикажемо ове случаје пре свега морамо да доведемо до насилног прекида програма. У стварном свету до овакве ситуације може довести нестанак струје, пад сервера, неки напад на нашу веб апликацију који омета њено правилно коришћење. Ми ћемо до насилног прекида програма довести тако што ћемо усред ажурирања података над табелом proizvod избацити изузетак (енгл. throw exception) након два успешно ажурирана производа. То ћемо постићи изменом кода у самој петљи за ажурирање података.

Поставићемо бројач итерација у петљи и након што се дође до одређеног броја итерације избацићемо изузетак који ће насилно прекинути извршење програма.

```
$brojac = 0;

// azuriraj stanje proizvoda
foreach($proizvodiIzKorpe as $pk){

    if($brojac == 2)
        throw new Exception("upravo ste hakovani");

    $id = $pk->proizvod_id;
    $kolicina = $pk->kolicina;

    $sql_update_from = 'UPDATE proizvod
                        SET stanje = stanje - :kolicina
                        WHERE id = :id';
    $stmt = $this->pdo->prepare($sql_update_from);
    $stmt->execute(array(":kolicina" => $kolicina, ":id" => $id));
    $stmt->closeCursor();

    $brojac++;
}
```

Такође да не би користили трансакције неопходно је да искоментаришемо или обришемо све линије кода које су везане за трансакције $this→pdo→beginTransaction(), $this→pdo→rollBack() и $this→pdo→commit().

Након ових промена приступимо нашој веб апликацији додамо сва четири доступна производа у корпу на страници плаћање унесемо податке кредитне картице и кликнемо на дугме Submit Payment. Након тога функција azurirajProizvod() почиње да се извршава али се у једном тренутку насилно прекида избацивањем изузетка и у том тренутку видимо страницу са слике 15. на којој видимо поруку да смо хаковани.



Слика 15. *Изглед странице након насилног прекида програма*

На слици 15 можемо видети стање у којем се налази табела proizvodi пре почетка извршења функције azurirajProizvode() и након тога што се трансакција насилно прекине.

id	naziv	autor	opis	slika	stanje	cena
1	DZEJN OSTIN IZABRANA DELA	Dzejn Ostin	Jedinstveno jednotomno izdanje izabranih dela Džej...	dzejn ostin.jpg	2	799
2	SEDAM PRICA	Anton Pavlovic Cehov	Anton Pavlovič Čehoj najčuveniji je pisac priča u ...	cehov.jpg	6	475
3	STRANAC	Alber Kami	Prvi roman Albera Kamija koji je izdat 1942. smatr...	kami.jpg	4	560
4	CRNO JAGNJE I SIVI SOKO Putovanje kroz Jugoslaviju	Rebeka Vest	"Toliko je prilika da ljudi žive kako treba, da sr...	vest.jpg	3	1592

а)

id	naziv	autor	opis	slika	stanje	cena
1	DZEJN OSTIN IZABRANA DELA	Dzejn Ostin	Jedinstveno jednotomno izdanje izabranih dela Džej...	dzejn ostin.jpg	1	799
2	SEDAM PRICA	Anton Pavlovic Cehov	Anton Pavlovič Čehoj najčuveniji je pisac priča u ...	cehov.jpg	5	475
3	STRANAC	Alber Kami	Prvi roman Albera Kamija koji je izdat 1942. smatr...	kami.jpg	4	560
4	CRNO JAGNJE I SIVI SOKO Putovanje kroz Jugoslaviju	Rebeka Vest	"Toliko je prilika da ljudi žive kako treba, da sr...	vest.jpg	3	1592

б)

Слика 16. *Стање у којем се налази табела proizvod а) пре и б) после извршења насилно прекинуте функције azurirajProizvod() у случају када не користимо трансакције*

Као што видимо у случају када не користимо трансакције табела proizvodi је само делимично ажурирана тј. ажурирана су само два производа из корпе након чега је извршење функције насилно прекинуто. Ова ситуација је довела нашу табелу у неконзистентно стање, стање појединих производа у табели је ажурирано али ти производи нису наплаћени, чак шта више ни сви производи који су у требали бити ажурирани нису.

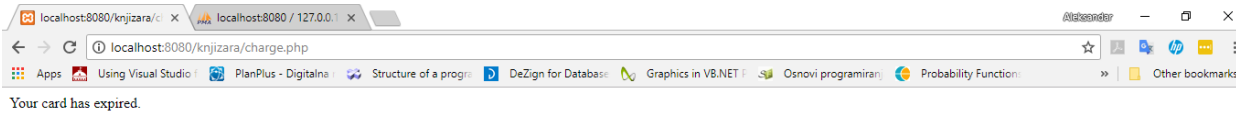
Када вратимо линије кода које се односе на трансакције и прођемо исту процедуру додавања производа у корпу (сва четири као у примеру без трансакције) приступимо страници за плаћање и након уноса података кредитне картице кликнемо на дугме Submit Payment и покренемо извршење функције azurirajProizvod() која се насилно прекида за разлику од ситуације коју смо имали у примеру изнад када нисмо користили трансакције видећемо да се табела proizvod не мења тј. да вредности у колони stanje нису ажуриране ни за један производ јер трансакција није успешно извршена.

3.3.2 Проблем процесирања кредитне картице

Још један пример корисности трансакција је моменат када корисник уноси податке о кредитној картици која је истекла, на којој нема довољно средстава или се једноставно догоди грешка приликом процесирања картице која нема везе са нашом веб апликацијом. Срећом Stripe софтвер нас је обезбедио изузецима који нас штите од оваквих ситуација и враћају нам поруку о насталим проблемима. Све што треба да урадимо је да процесирање картице обухватимо трансакцијом и тако заштитимо нашу базу података од нежељених ажурирања. На следећим примерима симулираћемо коришћење истекле кредитне картице са и без употребе трансакције.

Да би смо симулирали коришћење истекле кредитне картице све што треба да урадимо је да се пријавимо на нашу веб апликацију изаберемо производ који желимо да купимо (у овом примеру Седам Прича) и додамо га у корпу. Одатле приступимо страници за плаћање али овога пута уместо броја рачуна који се користи за плаћање из табеле испод бирамо број

картице којој је истекао рок (4000000000000069) унесемо остале податке картице и кликнемо на дугме Submit Payment након чега нам апликација хвата изузетак који је Stripe софтвер обезбедио и видимо страницу са слике 17.



Слика 17. *Страница са поруком о истеклој картици коју је обезбедио Stripe софтвер*

Уколико се не користе трансакције имаћемо ситуацију где је наша база података ажурирана али новац са кредитне картице није пребачен на наш рачун јер картица није валидна. У тренутку када добијамо поруку да је картица истекла (или да је се појавио било који други проблем са картицом или процесирањем картице) и да ми не можемо да наплатимо наш новац ми никако не желимо да ажурирања над базом података остану трајна. У случају без трансакција где купујемо производ Седам Прича са истеклом кредитном картицом видимо да је табела proizvod ажурирана (види слику 18) што никако не смемо да дозволимо.

id	naziv	autor	opis	slika	stanje	cena
1	DZEJN OSTIN IZABRANA DELA	Dzejn Ostin	Jedinstveno jednotomno izdanje izabranih dela Džej...	dzejn ostin.jpg	9	799
2	SEDAM PRICA	Anton Pavlovic Cehov	Anton Pavlovič Čehoj najčuveniji je pisac priča u ...	cehov.jpg	5	475
3	STRANAC	Alber Kami	Prvi roman Albera Kamija koji je izdat 1942. smatr...	kami.jpg	4	560
4	CRNO JAGNJE I SIVI SOKO Putovanje kroz Jugoslaviju	Rebeka Vest	"Toliko je prilika da ljudi žive kako treba, da sr...	vest.jpg	3	1592

а)

id	naziv	autor	opis	slika	stanje	cena
1	DZEJN OSTIN IZABRANA DELA	Dzejn Ostin	Jedinstveno jednotomno izdanje izabranih dela Džej...	dzejn ostin.jpg	9	799
2	SEDAM PRICA	Anton Pavlovic Cehov	Anton Pavlovič Čehoj najčuveniji je pisac priča u ...	cehov.jpg	4	475
3	STRANAC	Alber Kami	Prvi roman Albera Kamija koji je izdat 1942. smatr...	kami.jpg	4	560
4	CRNO JAGNJE I SIVI SOKO Putovanje kroz Jugoslaviju	Rebeka Vest	"Toliko je prilika da ljudi žive kako treba, da sr...	vest.jpg	3	1592

б)

Слика 18. *стање табеле proizvod а) пре и б) након извршеног неуспешног процесирања кредитне картице без коришћења трансакције*

У случају када користимо трансакције табела производ у оваквим случајевима остаје не промењена након краја трансакције и база података остаје конзистентна.

4 ЗАКЉУЧАК

Пословна предузећа се морају непрестано прилагођавати технологијама које константно напредују и мењају се. Ако су принципи обраде трансакција остали прилично непромењени током последњих тридесетак година технологије које имплементирају ове принципе су напредовале. Скорашње промене које почињу да имају утицај на производе трансакционог слоја укључују рачунарство у облаку (енгл. Cloud computing), високо скалабилни рачунарски дизајн, SSD меморије (енгл. Solid state memory), ESP технологије (енгл. Event stream processing) итд. Ове промене су превише нове да би са сигурношћу могли да предвидимо какав ће утицај имати на следећу генерацију трансакционог слоја али ипак могу да идентификују трендове које треба пратити.

Рачунарство у облаку је рачунарска услуга која се нуди преко интернета. Пружалац услуге може понудити комплетне апликације као што су електронска пошта, претрага, планирање ресурса у корпорацијама – ERP (енгл. Enterprise Resource Planning) или менаџмент односа са клијентима - CMR (енгл. Customer Relationship Management). Или могу нудити опште инфраструктурне услуге као што су процесирање података и складишни капацитети на захтев корисника. Неки системи рачунарства у облаку пружају комбинацију ових могућности. Обично услуга се може брзо прилагодити када се радни терет повећава а купац плаћа само за капацитете које заправо користи.

Највеће веб странице су иновативне у коришћењу прилагођених дизајна за скалабилне системе. Ово прилагођавање је неопходно да би могли да се носе са непредвидљивошћу радних оптерећења и потребом да се оптимизује брзо време одговора због што бољег корисничког искуства. Таква прилагођавања подсећају ране почетке технологија за обраду трансакција када су компаније развијале трансакционе слојеве за своје интерне апликације. Међу техникама које се сада користе су додатно кеширање на свим нивоима вишеслојне архитектуре обраде трансакција ради побољшања перформанси и осигурања што бољег времена одговора, повећане потребе да употребом трансакција упита и пословних процеса итд.

Еволуција трајне SSD меморије као алтернативе магнетним дисковима је још један тренд који утиче на дизајн система за обраду трансакција. Флеш меморија се сада користи за израду SSD уређаја. Њихов капацитет је у сталном порасту док њени трошкови опадају. Њихово време одзива је знатно краће него код магнетних дискова што је чини веома згодном за апликације за обраду трансакција. SSD меморије ће такође имати значајан утицај на решења за кеширање на средњим нивоима и скалабилност.

И на крају све већи значај обраде података и токова догађаја (енгл. Event Stream Processing ESP) је још један тренд који ће вероватно утицати на слој трансакционих производа. ESP технологије омогућују да се подаци обрађују у реалном времену, тако рећи у лету пре него што се трајно сместе у меморију. Ово је слично обради порука али са већим перформансама и пропусном моћи.

5 ЛИТЕРАТУРА

[1] Transaction Processing: Past, Present, and Future, IBM corp., 2012.
[2] Philip A. Bernstein, Eric Newcomer: Principles of Transaction Processing 2nd Edition, Morgan Kaufmann, 2009.
[3] Upravljanje izvršenjem transakcija i oporavak baze podataka, http://www.vps.ns.ac.rs/Materijal/mat677.pdf [10.05.2018]
[4] Prof. Milan Eric: Konkurentni pristup baza podataka i sigurnost baza podataka - Skripta, 2010.
[5] J. Gray, A. Reuter: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
[6] A. K. Elmagarmid: Database Transaction Models for Advanced Applications. Morgan Kaufmann, 1992.
[7] Brahim Medjahed, Mourad Ouyyani, Ahmed Elmagarid: Generalization of ACID Properties, 2009.
[8] Transakcije, https://imi.pmf.kg.ac.rs/indexold.php?option=com_docman&task=doc_view&gid=170, [10.05.2018]