Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Мастер академске студије

Модул: информатика у инжењерству

Предмет: Пројектовање информационих система и база података

Број индекса: 305/2012

Јован В. Поповски

Мобилне базе података

мастер рад

Комисија за преглед и одбрану:

1. **Др. Милан Ерић**
2. _____ Датум одбране:_____
3. _____ Оцена:_____

У оквиру овог мастер рада кандидат треба да анализира структуру и рад мобилних база података, проучи структуру система који су засновани на њима, сагледа тренутне трендове и развој технологија и да практично примени техничка решења и прикаже рад једног система база података.

Препоручена литература:

[1] Vijay Kumar / Miobile Database Systems, Computer Science an Informatics University of Missouri.

[2] Предавања и скрипте из предмета пројектовање информационих система и база података

Крагујевац, датум

25.06.2014

ментор:

Др. Ерић Милан

_____

Садржај

Резиме

Трендови у развоју софтверских решења све више наглашавају примену и имплементацију мобилних решења као основних елемената вишекорисничког система. Мобилне базе података су веома заступљене и произвођачи различитих решења препознају важност мобилних технологија. У овом раду је систематски обрађена тематика самих мобилних система,где је посебан акценат бачен на стандарде у оквиру база података, начине извршавања трансакција, ограничења која поставља мобилност и рад у различитим географским срединама. Практични део се своди на Java апликацију која опонаша један реалан систем који користи мобилни стандард за базе података sqlite .

Abstract

Trends in the development of software solutions are increasingly emphasizing the implementation and deployment of mobile solutions as well as the basic elements of the multiuser system. Mobile databases are very numerous and the manufacturers of various solutions recognize the importance of mobile technology . This paper systematically addressed topics themselves mobile systems where the accent was thrown to the standards within the database, transaction execution modes , restrictions imposed by mobility and work in different geographic areas. The practical part is reduced to the Java application that emulates a real system that uses the mobile standard for database sqlite .

Увод

Информације које користимо на мобилним телефонима и уређајима типа PDA, MP3 плејера постају све чешће свакодневна активност. Навигациони системи су сада све чешће стандардна опрема у аутомобилима као што су некада били музички системи. Сви ови уређаји су прилично корисни јер они могу да преузму одређене информације из базе података преко бежичних мрежа. Дакако постоје одређена ограничења. Ток информација у овим системима је само од сервера ка кориснику. Ово ограничење не омогућава корисницима да управљају базом података која може бити смештена било где на планети. Као последица тога , корисници морају да се задовоље оним што им сервер пошаље, што некада и не мора бити толико тачно и ажурно.

Истраживачи који се баве овом тематиком , и комерцијалне организације имају заједничку визију креирања информацијоног система за упрањање мобилним платформама који је у могућности да обезбеди управљање трансакцијама и базама података било где и када. Последња истаживања показују да смо све ближи остваривању тог циља. Обично се базе података обрађују на неком статичном рачунару, серверу или кријенту.

Просторне координате ових процесорских јединица су фиксне. Под овим моделом управљања информацијама, и процесорске јединице и њихови корисници су непокретни док се податци процесуирају. Сам начин на који се управља информацијама зна понекада да буде неучинковит и да има неприхватљиво малу продуктивност. Зато што је немогуће остварити корак са све већим потребама тржишта. Новија истраживања у друштвеним слојевима , и потреба да се има боља сарадња на међународном нивоу, повећање просторне мобилности , и страх од изолације су генерисали нови тип информација и захтева. Један од важних аспеката ових захтева је да корисник може бити ослобођен привремених и просторних ограничења у обради одабране информације која може бити постигнута географском мобилношћу за време обраде података. Инхерентна непокретност процесорских јединица и њихових система је озбиљна препрека при постизању задатог циља. Ограничен тип мобилности је могуће остварити у конвенционалним системима

(ТВ,музички системи и слично). На пример , даљински управљач може бити повезан на систем преко кабла да би се управљало системом из било ког дела собе или куће. Прва примена даљинског управљања је била у немачкој морнарици где је била употребљавана да уништава бродове , подморнице и активира оружја у првом и другом светском рату, а касније су у САД инжењери експериментисали са ,на пример, аутоматским вратима која се отварају што и заправно означава почетак бежичне ере у том погледу.

Комуникациони системи су такође интересантни . Први мобилни радио који је само радио у једносмерном правцу је развијен за потребе Детроитске полиције, да би 1964. године ***Bell System*** представио побољшану мобилну радио мрежу (IMTS) која се састојала од емитера са транзистором велике јачине. Уређај је радио двосмерно и није било потребе за притискање било каквих тастера при говору. Касније је дозвољавало аутоматски избор канала , и ограничени опсег од 25 до 30 kHz.

До сада сви системи су били засновани на аналогној комуникацији , која је имала бројна ограничена. Да би се то отклонило од 1987 до 1995, нови бежични протоколи као што је TDMA (Time Division Multiple Access), CDMA (Code Division Multiple Access) су представљени. Данашњи мобилни системи су углавном засновани на дигиталној технологији ,али се понегде још користи и аналогна. Да би све ово било јасније табела 1 приказује све битне догађаје у мобилној комуникацији по хронолошком реду.

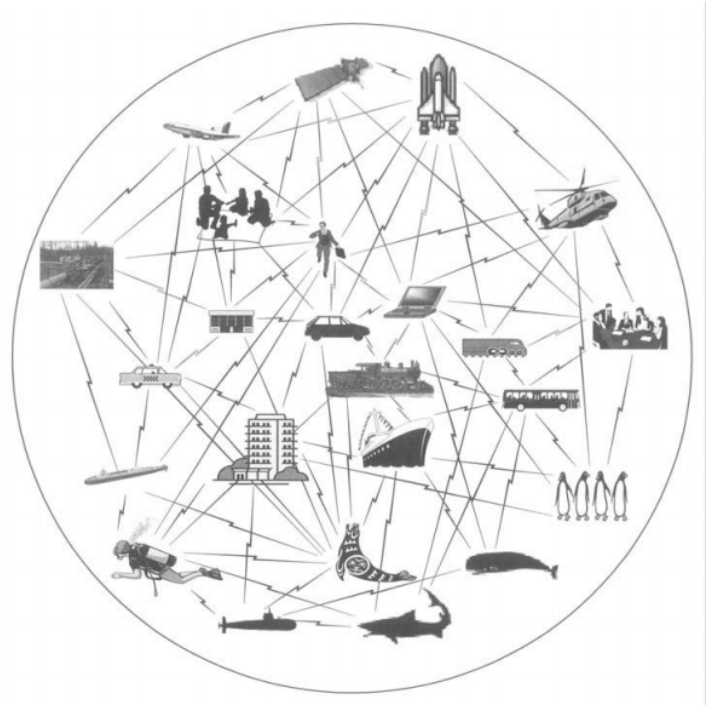
Мобилни телефони у комуникациони уређаји су успели да направе делимично спојен информациони свет који је ослобођен од просторних и временских ограничења. Даље „свуда и у свако време“ постаје све шири појам који више чујемо.

Датум	догађај
1867	Максвел помиње опстојање електромагнетних таласа.
1887	Херз доказује постојање електромагнетних таласа.
1890	Бренли развија технике детекције радио таласа.
1896	Маркони представља бежични телеграф.
1897	Маркони је патентирао бежични телеграф.
1898	Маркони добија награду за допринос комуникацији.
1898	Прва бежична линија од Француске до Енглеске је успостављена
1901	Маркони успешно преноси сигнал од Корнвола до Њуфаундленда
1909	Маркони добија Нобелову награду.
1928	Детроитска полиција инсталира мобилне пријемнике у патролна возила.
1930	Мобилни предајници су уграђивани у аутомобиле.
1935	Армстронг показује шему Фреквенционе модулације (FM)
1940	Све више се прелази на FM
1946	Мобилни системи се пребацују на PSTN(Public Switched Telephone Network)
1949	FCC признаје мобилни радио као нову класу услуга
1950	Број корисника мобилних радио уређаја је порастао за 500.000
1960	Број корисника мобилних радио уређаја је порастао за 1.4 милиона.
1960	IMTS мрежа (Improved mobile service) је представљена.
1976	Bell Mobile користи 12 канала да омогући рад 543 лица у ЊуЈорку.
1979	NTT-Japan се појављује са новим системом комуникације.
1983	AMPS(Advanced Mobile Phone System) је имплементиран у САД
1989	GSM се појављује у Европи као нови дигитални стандард за телекомуникације.
1991	US Digital Cellular phone систем је приказан
1993	IS-95 CDMA (code division multiple access) систем је представљен у САД.
1994	GSM је представљен у САД
1995	FCC је обезбедила 1.8 GHz мрежу за системе личне комуникације PCS.
1997	Број мобилних телефона у САД је порастао за 50 милиона.
2000	Трећа генерација система ?

Табела 1. Прекретнице у мобилној комуникацији по хронолошком реду

ИТ свет и базе података

Слика 1 приказује потпуно повезан информациони простор који људи користе данас. Сваки објекат на овом свету са својом функционалношћу је повезан са другим преко бежичног линка. На пример банка или особа је повезана на конференцију. Даље у сваком моменту особа банка или било који човек на планети може имати информацију о сваком објекту који је повезан на мрежу. Таква бежична комуникација је постала кључна за високо покретно и динамичко друштво. На пример чланови породице који живе односно раде иностранству и желе да буду у контакту или један председник компаније који жели да има информацију која приказује све активности унутар компаније и помаже му при управљању истом.

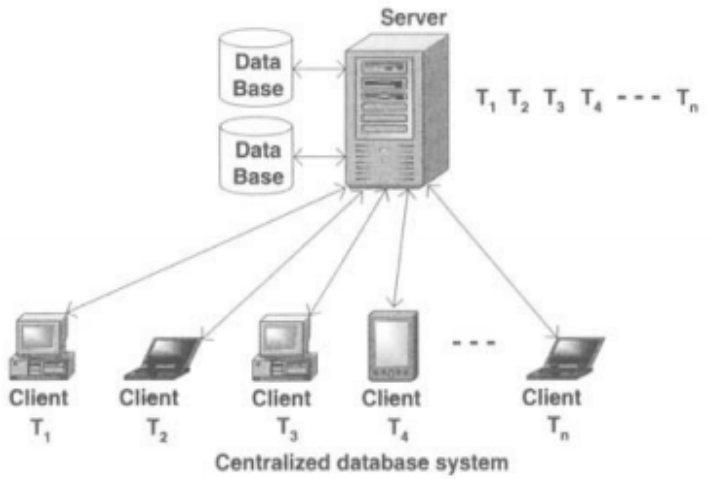


Слика 1.Информациони свет

Систем користи трансакциони механизам да би усагласио све промене у бази података у реалном времену. Софверски модули који се баве трансакцијама се називају DBMS (Database managenent system), а такође се користе као интерфејс између корисника који приступају бази и базе података. Корисник који врши интеракцију са DBMS кроз упит као што је SQL (structured querry language), преузима корисничке податке и процесуира их DBMS у. Управљачки систем базе података односно DBMS врши трансакцију или сет трансакција које покрећу неопходне операције које су под контролом механизама који обрађује захтеве корисника.

Постоје две врсте архитектуре система са када је цела база података имплементирана:

- Централизована
- Дистрибуирана



Слика 2.Архитектура централизованих система база података

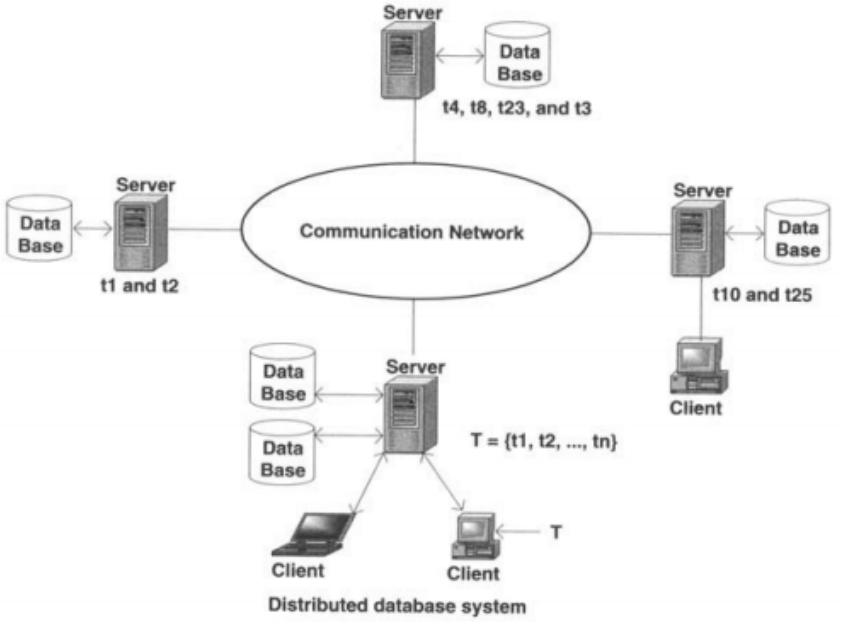
Централизован управљачки систем базе података : У овом систему постоји један сервер односно чвориште и већи број интелигентних терминала клијената повезаних на сервер преко бежичне мреже . Клијент прихвата кориснике преко упитнох језика као што је на пример SQL. Он може да обради упит (односно трансакцију) ако поседује

ресурсе у другом случају шаље их серверу на процесирање . Сервер шаље резултате клијенту и клијент форматира и саље их кориснику. Сервер шаље резултате клијенту и клијент форматира и шаље их кориснику. Слика показује референтну архитектуру оваквог централизованог система.

Дистрибуирани управљачки систем базе података: У дистрибуираном систему велики број централизованих система база података су спојени заједно са комуникационом мрежом .Свака конфигурација индентификује специјални тип дистрибуираног система базе података. На слици 3. је илустрована референтна архитектура дистрибураног система база података.

Постоје два типа често коришћених категорија (а) федерализовани системи база података или (б) вишебазни системи. Ова категоризација се заснива на аутономији чворова, обради података , и начину дистрибуције. У федерализованим системима архитектура је подскуп сервера који су делимично засновани на аутономији при управљању активностима. Они су чланови федеративне и сарађују са осталим серверима под сетом одређених протокола . На пример , у великој компанији свако одељење има свој сервер и може учествовати да одржава коегзистентан преглед активности у својем одељењу. Овај начин рада може бити хомогени где сервери раде на истом управљачком систему базе података или хетерогеним где сваки сервер ради на другом систему. На пример један сервер може користитi Oracle , а други може бити DB 2 ili Sybase.

У вишебазном систему, сви сервери морају имати потпуну аутономију. На сваком је серверу да сарађује или не сарађује са другим сервером унутар архитектуре. Овај начин архитектуре се примењује приликом имплементирања односно сваки провајдер делује индивидуално са или без заједничке кооперације.



Слика 3 Архитектура дистрибуираног система база података

Партиционисање и дистрибуција базе података

У централизованим системима база података постоји само један серверр, стога потање дистрибуције се не поставља. Дакако ово је веома важно питање у дистрибуираним системима и мора се управљати пажљиво зато што знатно утиче на особине система и доступност.

У дистрибуираном систему база података се обезбеђује на 3 начина: (а) партиционисана , (б) делимично реплицирана, и (в) потпуно реплицирана да побољша доступност , квалитет услуга и особине. Избор дистрибуције података зависи на великом броју од параметара система и апликација. Једна од најважнијих ствари коју треба узети у обзир је локална доступност података која је потреба приликом упита да би се минимализовали трошкови комуникације са сервером.

Партиционисање база података: Под овом шемом цела база података је тачно подељена на више партиција. Обично је исти број партиција колико има и процесних .чворова (сервера). Ове партиције су премештене на сервере под неким критеријумима . Најважнији критеријуми су следећи:

- Партиција треба да подржи највечи степен локалности. Ово значи да већина упита се обрађује на тој партицији односно локалном серверу
- Треба да помогне минизацији трошкова комуникације
- Треба да помогне минизацији трошкова одржавања
- Треба да помогне приликом локазилацији серије трансакција
- Треба да минимизује трошкове евентуалног опоравка

Партициона шема је мање поуздана и пати од неких централних недостатака. Ако сервер падне онда цела партиција постаје недоступна док се не поврати сервер. Као додатак овоме због нулте податковне редундансе, немогуће је обезбедити потпуну и адекватну поузданост. Неке од ових ставки су решене делмично реплицирањем базе података

Активност	Пуно реплицирање	Парцијално реплицирање	Партиција
Локализација података	Веома високо	Средње	Основно
Серијализација	Тешко	Напредно	Лако
Глобална конзистентност	Тешко	Напредно	Лако
Поузданост	Веома високо	Напредно	Основно
Трошкови опоравка	Веома високо	Напредно	Основно

Табела 2. Типови реплицирања

Парцијална репликација : Под овом шемом база података је подељена односно партиционисана и ти делови су реплицирани на више од једног сервер. Парцијална репликација има све особине партиционе шеме и даље побољшава локалност и поузданост саме базе података. Ако сервер закаже или откаже реплицирана копија партиције је још увек доступна за обраду упита. Опоравак је доста лакши јел реплицирана партиција може бити копирана тотално на сервер након што се он поврати.

Релативно је доста штедљивије да се одржава глобална конзистенција зато што било која промена на партицији мора бити инсталирана на свакој репликацији која је смештена на разним серверама. Да би даље повећали учинковитост базе она је потпуно реплицирана.

Потпуна репликација: Под овом шемом цела база података је реплицирана на све сервере .Ово има максималну локалност и такође минималитује податковну комуникацију и трошкове приликом обраде упита.

Тотално реплицирана шема омогућава највећу поузданост и доступност, али са ценом одржавања глобалне коегзистенције. Шема се не користи у пракси због високих трошкова смештања. Табела сумаризује сва три приступа дистрибуције података.

Обрада базе података

Механизам који обезбеђује промене се назива трансакција. Трансакција заправо узима базу из својег иницијалног коегзистентног стања у следеће стање и том приликом креира сет операција на физичким податцима. Ова ограничена трансакција је цео сет коегзистентних ограничења, базе и апликативног програма. Стога, трансакциону структуру можемо посматрати као:

Трансакција = < Сет ограничења, База података , Апликативни програм >

Цео скуп тих ограничења може бити имплементиран на два начина: (а) испрограмиран у апликативном коду и (б) укључен унутар саме трансакционе структуре. Први метод је далеко учинковит, али непоуздан и није увек могућ. Програмирање свих тих ограничења је склоно грешкама у коду. Програмер мора да поседује одлично познавање системске архитектуре да би избегао логичке грешке. Тешко је за било ког програмера да зна шта да тестира као ограничење. Из ових разлога други приступ је заступљенији где су ограничења укључена у трансакциону структуру. Са овим повезивањем трансакција добија јединствену позицију унутар целе области базе података и представља себе као механизам за обраду података.

Трансакциона структура

Структура трансакције је најбоље описана раније. Појмови (Атомност, Конзистенција, Изолација и Трајност су заступљени у општепознатом акроними *ACID*).

Систем користи трансакциони механизам да би усагласио све промене у бази података у реалном времену. Софверски модули који се баве трансакцијама се називају *DBMS(Database managenent system),* који такође користи као интерфејс између корисника који приступају бази и базе података. Корисник који врши интеракцију са DBMS кроз упит као што је *SQL (stryctyred qyerry langyage*), који преузима корисничке податке и процесуира их DBMS у. Управљачки систем базе података односно DBMS врши трансакцију или сет трансакција које покрећу неопходне операције које су под контролом механизама који обрађује захтеве корисника

Незаштићене акције : Делимични резултат слободне акције може бити виђен на некој другој слободној акцији, а коначни исход не мора бити коегзистентан. Измена или сортирање операција над фајлом, писање на диск или меморију рачунара, су примери незаштићених операција. Дисциплина база података захтева да акције које се спроводе над податцима не смеју бити незаштићене

Заштићене или ограничене акције: Парцијални резултат ограничене акције није видљив другој неограниченој односно незаштиченој акцији. Доступност крајњег резултата показује да је акција комплетирана успешно. Како обрада базе података мора имати ову особину, која је укључена у трансакцију као атомичност.Атомичност захтева да свака ограничена акција постоји само једном у два могућа исхода (а) потпуно комплетирано или (б) незапочето. Незапочето стање се постиже повратном акцијом. Ова ограничена акција је атомска, или је успешно комплетирана где је коначни резултат доступан другој заштиченој или незаштићеној акцији или ако дође до неуспеха у сред операције онда је парцијално извршење враћено у почетни положај ,а апарцијални део се уклања из базе података, Из овог разлога , посебно за обраду базе података својство конзистенције је

уведено тако што смо сполији особине конзистетције дефиниција ограничене базе података је промењена.

Стварна односно примењена акција : Комплетирање оваквих акција је немогуће повратити може се поистоветити са ситуацијом када испалите метак то је радња која се не може вратити уназад. Коначни резултат је достигнут који може бити исправан али тако и погрешан. Из овог разлога не може бити атомичан. Стварна акција је корисна у управљању базама података ако може бити издата гаранција да покретање акције ће генерисати коегзистенцију која ће задржати резултат. Али такву гаранцију никада не можемо добити. Очигледно је да таква акција може бити незаштићена и може бити прихватљива ако то не утиче на стварни живот.

Равне трансакције

Дискусију о трансакционом механизму почињемо са равног трансакцијом која је увод у касније сложеније методе. Детаљна дискусија о овој тематици је неопходна да би се разумело управљање мобилним базама података.

Равна трансакција је најпростији тип механизма који подржава операцију ан једном нивоу која за време њеног извршења се не покрећу друге трансакције. Трансакција зависи од друге трансакције од које очитава вредности које су модификоване латтер. Неки примери равне трансакције су дебит/кредитна транскакција, плаћање рачуна, и тако даље. Ово се све своди на једнослојнун операцију.

Равне трансакције су ограничене са додатним захтевивма који морају да задовоље потребе коензистентних ограничења. Појачање у атомичности и коензистенцији захтева две додатне особине које су укључене у трансакциону структуру. Главне особине трансакције могу бити дефинисане редом:

Атомичност :

Ово је заштићена акција која подржава само два стања (а) комплетирано и (б) не започето. Комплетирано стање идентификује успешно комплетирану операцију а не започето стање идентификује односно показује да операција није ни започета. Не започето стање се постиже успомоћ повратне операције која је атомична. Атомичност акције

враћања дефинише другу особину која се зове Идемпотентност. Идемпотентност гарантује да ће једна од више операција враћања исте акције креирати најбољи коначни резултат. На пример, додељивање ***x:=5*** је идемпонтентно јел има више успешних решења које могу резултирати успехом операције. Даље ако је атомична операција поновљена више пута њен коначан резултат ће бити исти.

Атомичност трансакције гарантује да средњи резултат не зависи од корисника који је покренуо трансакцију. Ово је такође тачно за поруке које проистичу из трансакције. За време извшења ако корисник прими поруку од транскације и ако је та трансакција више пута враћена, онда порука не може бити враћена да би се установило да порука није послата кориснику. У ствари порука од трансакције је парцијални резултат који под атомичности не сме бити видљив кориснику. Ово показује да за време егзекуције трансакције се не генеришу било какве поруке или ако се то врши то се не види даље од трансакционог простора.

Конзистенција: Ова особина се брине да вредност податка репрезентују чињенице. Односно ако корисник ставља 100$ на свој рачун са постојећим стањем од 500$, онда база података мора показати коначно стање као 600$. У погледу конзистенције ограничења кредитне операције морају задовољити следећи критеријум:

Претходно стање = Тренутно стање – Задњи износ кредита

Следеће стање базе података је коегзистентно само ако је задње стање конзистентно. Трансакција узима за готово да увек управља конзистентим базама података и такође гарантује да ће резултовати конзистентним стањем.

Изолација: Ова особина се брине о томе да задржи коегзистенцију, за две суочене односно операције (упис и испис или испис и испис) које се врше над истом ставком буду онемогућене. Као што је поменуто раније у тексту, апликациони програм може проверавати коегзистеницију у коду , али како је тај прилаз тежи и склон грешкама програмера и буг овима у софтверу. Изолациона особина је спојена у трансакцији тако да помера снагу конзистенције ка нивоу моделирања . Приморана је да кроз посебни софтверскимодул, који је обично конкутентни контролни механизам или серијски механизам.

Трајност: Ова особина гарантује да ће крајњи конзистентни резултат трансакције постојати у бази и неће бити уништен било којим падом система или било каквом грешком истог. Успешно извршење транскације показује да су задовољени критеријуми атомичност, конзистенција и изолација, и резултат мора се наћи унутар базе података. Када су резултати смештени и корисник је обавештен, онда је речено да се трансакција врши. Особина трајности гарантује да се ефекат трансакције не може вратити односно отказати.

Трајност као особина не може се уско довести у везу са претходне три особине. Она се брине да шта је год инсталирано база постоји у садашњем времену под било каквим кваром.

Серијализација трансакција

У централизованим системима трансакција се искључиvo obavlja на jednom месту. Мноштvo клијената започиnje трансакције (T_1,T_2,...,T_n), и све оне се шаљу серверу на извршавање. Сервер преусмерава резултат клијентима на крају извршења сваке трансакције као на слици 2. Начин на који се обрађују истовремено више конкуренских трансакција помаже да се цена имплементације ACID ограничења и повратка базе потакада задржи на ниском нивоу. Ово је прихватљиво само у условима где је мали опсег оптерећења и где трансакције су углавном кратког века. У већим системима где је велики број трансакција коришћење дистрибутивног система је постало потреба.

У дистрибуираном систему у зависности од типа дистрибуције података, трансакција T је подељена на већи број подтрансакција *($T = m_1,m_2,...,m_n$)*. Ове подтрансакције могу бити дистрибуиране на одговарајуће процесне чворове или сајтове (сервере), од којих су неки процесуирани у паралелном режиму (слика 3). Дистрибуирана трансакција је далеко учинковитија и минимизује вишак ресурсам, ипак управљање

оптерећењем посла је веома комплекнса и сложена операција и захтева специјални прилаз. Ово је поготово битно у вишебазном систему.

Постизање конзистенције је могуће само на начин преко серијског извршавања. Серијско извршавање не дозвољава дељење података тако што следећа трансакција почне тек када се последња заврши. Једна од предности оваквог типа је да се конзистенција апсолутно очувава али на штету учинковитости ресурса, при чему систем ради доста спорије и трошкови су превисоки.

Једини начин да се трансакције раде симултано је нажалост серијски а то значи да конзистентност није тотално загарантована, што ej кључно за системе БП. Ако је овај тип извршења постигнут у конкурентном извршењу кроз контролисано дељење података, онда се проблем ефикасности не јавља. Води се рачуна да особина која је постизање конзистенције при серијском извршавању конкурентних трансакција, које заједнички искључују или изолирају две трансакцијепреко коришћења заједничке ставке. Следећих пар одељака се баве овом тематиком детаљно.

Критеријум исправљања серијализације

Серијско извршавање трансакција не може постићи жељену учинковитост, тако да се извршава конкурентно. Неконтролисана конкурентна извршења прете интегритету БП због интерференције са другим трансакцијама које су у току. Када се трансакције обављају инстовремено. Њихове основне операције (упис и испис) су интерлеавед. Интерлеавинг основних операција може довести до тога да доспевају у додир једна са другом тако да се стварају не коегзистентни резултати : што је (а) излаз трансакције који није коегзистентан са вредностима објекта у бази или (б) ефекат трансакције може уништити коегзистенцију БП.

Не коегзистентност је резултат интерференције која може бити илустрована кроз извршавање трансакције у банци користећи проверу налога (идентивиковано као *x*) и налог за чување (означено са *y*) који корисник користи. Прекорачење овог налога није

дозвољен тако да њихове вредности не могу бити негативне. Тако да постоје два главна типа проблема (а) изгубљено ажурирање (б) не коегзистентни повраћај.

Изгубљено ажурирање

На првом примеру размотримо две истовремене трансакције T_1 и T_2 које пребацују новац на један налог *x*. Претпоставимо да се на том рачуну односно налогу налази 100$ ид а T_1 пребацује 50$ и T_2 пребацује 100$.

T_1	**T_2**	**T_3**	**T_4**
begin		begin	
	begin		begin
a=read(x)		a=read(y)	
	b= read (x)		a= read(y)
write(x, a+50)		If (a>=50) then	
	write (x, b+100)	write (y, a - 50)	
commit			If (b>=90) then
	commit		write (y, b - 90)
		commit	
			commit
(a) Конкурентни депозити		**(b) Конкурентна повлачења новца**	

Табела 3

Испреплетано извршавање T_1 и T_2 је показано на слици 5 односно табели, где су a и b локалне променљиве. Извршавање резултује следећом секвенцом операција над БП. Операција читања је идентификована као $r_i [x,v]$, који очитава почетну вредност x као $в$, а исписује операцију идентификовану као (w_i $[x,v]$) која уписује вредност v у x. Pojam c_uденотира извршење трансакције T_i. Корисно је да представимо насумично извршавање са роком који памти релативно извшење наредби или читања, писања и извршавања или прекидања операција. Омогућава лак начин да се види постојање подударних тј истовременим операција на распореду који је потребан у серијализацији. Распоред извршења на слици може бити написан као :

$$r_1[x,100]\ \ r_2[x,100]\ w_1[x,150]\ w_2[x,200]c_1c_2$$

Према овом распореду T_1 і T_2 sy извршене успешно и T_2 је уписала 200$ вредности коначно на x. Ако су T_1 и T_2 вршене серијски као T_1 -- > T_2 или T_2 -- > T_1 , онда коначна вредност x износи 250$. Јасно је да то није случај са 50$ депозита са T_1 који фали. Разлог томе се крије што је T_1 преписан од стране T_2 , како је T_2 учитала иницијалну вредност x пре него што је T_1 извршио упис. Ово се не би десило да се ради о серијском извршењу типа T_1 -- > T_2 или T_2 -- > T_1. Ово је случај уплетања односно интерференције од стране T_2 на извршавање T_1.

Горепоменуто испреплетано извршавање које је илустровало проблем ажурирања није утицало на интегритет ограничења БП за коју се односи. На пример размотримо испреплетане трансакције T_3 і T_4 које повлаче 50$ и 90$ sa y респективно, са претпоставком да је стање на рачуну y 90$. Распоред извршавања је дат на следећи начин:

$$r_3[y,90]\ \ r_4[y,90]\ w_3[y,40]\ w_4[y,0]c_3c_4$$

Испреплетани распоред који имајуТ$_3$ і Т$_4$ заједноали и серијско извршавањеТ$_3$ -- > Т$_4$ је само једно од два повлачења која су уследила од како друго повлачење имплицира за y ≥ 0 и не би било дозвољено ако би угрозило интегритет односно целовитост ограничења.

Некоегзистентан проблем повлачења:

Други тип испреплетања је познат као некоегзистентно повлачење, које може уследити као *неуредно читање* или *непоновљиви трагови.* Неуредно читање се јавља када трансакција T_i очитава ставку коју уписује трансакција T_j ($i \neq j$) . Ако се T_j изврши, онда је верзија T_i некоегзистентна. Другим речима T_j је податке са T_i испунила нкомплетним податцима. У непоновљивом читању ако T_i очитава потадке поново , онда ће добити друге вредности него што би добила раније.

Следећи пример илуструје како некоегзистентност делује на повлачење средстава. Размотримо испреплетано извршење трансакција T_5 и T_6, које су показане на слици на којој су a,b,c,d локалне променљиве. Трансакција T_5 показује суму стања x i y где трансакција T_6 пребацује 100$ са y на x. Претпоставимо да је почетно-иницијално стање на рачуну x 200$ и да је y 100$, извршење T_5 i T_6 је представљено следећим редоследом.

$$r_6[x,200]\ \ r_6[y,100]\ w_6[x,300] r_5[x,300]\ r_5[y,100] w_6[y,0] c_6 c_5$$

T_5	T_6	T_7	T_8
begin		begin	
	begin		begin
	c = read (x)	a=read(x)	
	d = read (y)		b= read(x)
a = read (x)	write (x,c + 100)		c= read(y)
b = read (y)			If (b> c + 1) then
	write (y, d - 100)		write (x, b - 1)
	commit	write (x, a - 1)	commit
oytpyt (a+b)		commit	
commit			commit
(a) Prikaz kombinovanog stanja		**(b) Конкурентна повлачења новца**	

Табела 4

При овој операцији и x и y су ажурирани коректно (односно x = 300$ i y = 0$). Иако је сума на оба непромењена и једнака 300$ T_5 има излаз од 400$. На неиспреплетаном извршавању T_5 и T_6, на којем су T_5 и T_6 извршене редоследом респективно, T_5 ће имати излазну суму од 300$.разлог овог мимоилажења је да је T_5 послатрало парцијално ефекте T_6, како је T_5 послатрало некоегзистентно стање x и y. Приликом овог извршавања, податци са T_5 су постали...

Иако је у горе приказаном извршавању трансакције ниједно ограничење није нарушено, слично као и код изгубљеног ажурирања, некоегзистентна повлачења могу узроковати нарушавањем интегритета саме базе података, као што пример на слици показује, претпоставимо да је стање на рачунимах = 40$ i y = 38$. Као додатак

Претпоставимо да имамо захтев да стање на текућем рачуну је веће од баланса на штедном налогу осим ако оба имају стање 0.(x > y ako je x ≠ 0 i y ≠ 0). T_7обезбеђује трансакцију која подешава стање на штедним рачунима да буду долар мања од текућих. T_8 повлачи долар са текућег рачуна.

$$r_7[x,40]r_8[x,40]w_8[y,38]c_8w_7[x,39]c_7$$

Коначан резултат операције је да је x=39$ i y=39$, која нарушава интегритет ограничења које гласи да је x > y ako i x i y имају вредности веће од0.

Следечи примери показују присуство интерференције и илуструју да тачност индивидуалних трансакција није довољна да обезбеди тачност извршавања.

Интуитивно, тачно извршавање сета трансакција које морају бити ослобођене од мешања. На свим примерима изнад , били смо у могућности да закључимо да било конкурентно извршавање сета трансакција је прихватљиво упоређивањем резултата на не испреплетаним или серијским извршавањима сета трансакција.

Серијско извршавање је ослобођено интерференција, тако да је коегзистентност загрантована. Даље ако је резултат конкурентног извршавања сета трансакција исти ккао и онај који је креиран серијским извршавањем истог сета трансакција онда је конкурентно извршавање такође коегзистентно. Таква конкурентна извршавања су еквивалентна

серијским извшавањима и називају се серијабилним и задовољавају потребе изолације као једне од главних особина БП.

Даље , серијска извршавања су ослобођена интерференција, како следећи распоред извршавања T_7 и T_8 илуструје

,

T_7	T_8	T_7	T_8
begin		begin	
a=read(x)			begin
write (x, a - 1)		a=read(x)	
commit	begin	write (x, a - 1)	b= read(x)
a = read (x)	c = read (x)	commit	
b = read (y)	d = read (y)		c= read(y)
	If (b> c + 1) then		If (b> c + 1) then
	write (x, b - 1)	write (x, a - 1)	write (x, b - 1)
	commit		commit

Табела 5

Према овом распореду T_8 посматра конзистентно стање рачуна x и y које је настало након T_7, премда T_8 не нарушава интегритет ограничења заобилазећи повлачење средстава са x.

$$r_7[x,40]r_8[x,40]w_7[y,39]c_7r_8[y,39]c_8$$

Неколико верзија серијализације се могу пронаћи. У највећем броју протокола серијализација је заснована на бележењу конфликтних операција и зато се назива конфликтна серијализација. Две операције су у конфликту када њихови резултати зависе једна од друге. Конфликтна серијализација омогућава да сви парови конфликтних операција се појаве на истом реду еквивалентног извршавања [18,20,22,39]. Било да је извршавање у конфликту серијализације се може лако установити, како ће бити даље у тексту објашњено.

Различите верзије серијализације укључују и преглед и стање. Предлед серијализације захтева да свака трансакција чита исте вредности и да је коначна вредности иста, на два еквивалентна извршавања. Стање серијализације захтева, да под свим могућим интерпретацијама трансакција, два еквивалента извршавања морају бити иденти;на када се прегледају из једног стања у други. Интерпретација уписа је релација вредности написане унутар базе пре него што се изврши упис трансакцијом. Упис трансакцијом може бити потенционална функција неког подскупа претхнодно очитаних вредности од трансакције. Ове различите вредности дозвољавају већи серијабилност, али нажалост не користе се толико у пракси.

У следећем одељку је конфкликтна серијабилност дефинисана пратећи критеријум тачности.

Теорија серијализације

Кључна ствар око теорије серијализације је појам историје, која представља конкурентно извршење сета трансакција T [8,5,31,29,28,32,33,34]. Како је раније поменуто , трансакције су неопходне да би се обавиле операције над објектима на тачан начин. То је , критеријум тачности који је дефинисан ограничењима по редоследу конкурентних операција и трансакција унутар историјата.

Објекти Операције и Конфликти

База података је ентитет који садржи све дељене објекте унутар система. Има приступ трансакцијама и манипулисе објектима унутар базе тако што позива све операције које су специфицне индивидуалним објектима.Стање објекта је представљено његовим садржајем. Сваки објекат има тип, који дефинише скуп операција које му омогућавају да креира, мења и утврди стање објекта тог типа. Претпоставља се да су операције атомичне и да операција увек има као резултат излаз. Операције које су дефинисане као објекти се сматрају функцијама са једног стања објекта на други. Резултат операције на једном објекту зависи од стања самог објекта. За задато стање ***s***објекта користимо повратак(енг.***remyrn(s,p)***) да би обрисао излаз који је настао операцијом p и користио стање (енг.***smame (s,p))*** да би обрисао стање које је настало након извршења p.

Дефиниција Две операције p и q су сучељене у стању ***s,***

$$(state(state(s,p),q)) \neq state(state(s,q),p)) \vee$$
$$(retyrn(s,q)) \ retyrn(state(s,p),q)) \vee$$
$$(retyrn(s,p)) \ retyrn(state(s,q),p)) \vee$$

Две операције су у конфликту ако стање објекта или неке излазне вредности није независноу односу на ред извршавања. Дакле да би ово било могуће, обе операције треба да врше обраду објекта стим да једна може модификује објекат у реалном времену (то се назива операцијом уписа). Са друге стране , две операције читања на истом објекту могу бити извршене било којим редоследом.

За сваки објекат, релације између сучеоних-конфликтних операција се могу описати табелом . Потврдан унос за (p,q) указује на то да су операције p и q компатибилнетј да немају сучељавање. Табела показује матрицу на којој објекат подржава операције читања и уписа.

Операције	***Читање(x)***	***Испис(x)***
Читање (x)	*Да*	*Не*
Читање (x)	*Не*	*Не*

Табела 6. Матрица компатибилности операција читања и уписа.

Сада је неопходно да се формално окарактеришу трансакције , историје, и тачно извршавање за цео скуп трансакција у погледу позивања сучеоних функција. Сучеоне трансакције могу ометати једна другу само ако обе позивају сучеоне операције на истом објекту у истом времену.

Историјат Трансакција

За време извршавања трансакција, оне укључују операције унутар објеката унутар базе података и трансакционog ypravljanja. Позивање операције на једном објекту или трансакционом управљању… се назива догадјај. Цео скуп операција и …позваних од стране трансакцијеTi могу да се прикажу релацијом "→", која демистификује тренутни распоред којим се догађају дешавају- Разлог томе је да је трансакција моделирана односно сачињена као део из разлога што може да садржи две или више несучеоних операција паралелно. На пример, две операције читања на два различита објекта се могу извршавати упоредо.

.

Користиморі [ob] да би објаснили један догађај у сагласности са позивањем операције п на објекту об од стране трансакцијеTi. Углавном , користимо *Єj* да би објаснили позивање догађаја *Є* од стране трансакције Ti. Претпоставка *Є→Є'* је истинит ако догађај *Є* надмашује догађај *Є'* у трансакцији Ti. У другом случају је нетачно.

Теорема серијабилности

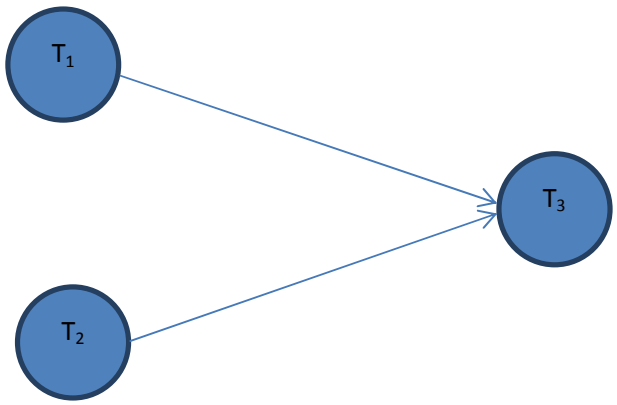
H_c је серијабилно ако $\forall m \in T_{comm}(\tau C^* \tau)$.

Једноставаан начин да проверимо да ли је изазван редослед зависност ацикличан је конструсањем графа зависности, и тражити циклусе. Серијализациони граф у коме чворови представљају трансакције у H, и где Ti Tj, значи да Ti има зависност у редоследу у односу на Tj . На пример слика 4.6 показује фраг серијализације следеће историје.

$$H = r_1[x]w_1[x]r_3[x]w_2[y]c_1r_3[y]c_2w_3[y]c_3$$

Серијализациони граф који је ацикличан може исто бити коришћен да се пронађе еквивалентна серијска историја.Ово може бити постигнуто креирањем тополошког сортитара серијализационог графа. Где се год надовезују две трансакције на пример Ti Tj, онда све операције на Ti морају да се појаве пре оних на Tj еквивалентним редоследом. Како је историја делимично уређена, неколико серијски извршења могу бити регулисана тополошким сортирањем. На пример тополошко сортирање на графу на слици креира две еквивалентне историје.

$$H_8 : T_1\ T_2\ T_3 \text{или } T_2\ T_1\ T_3$$



Слика 5. серијализациони граф

Још једна особина конфликтне серијабилности је да је пре извршења затворена. Префикс затворена приликом извршења значи да било да чува историју H такође чува и извршену пројекцију под H'. Другим речима , непотпуно извршавање трансакција узроковано планером, при чему је трансакцијама дозвољено да се извршавају ако је њихова историја серијабилна. Даље значи да конфликтна серијабилност може бити употребљена као критеријум за исправљање грешака. Након исправке, база података рефлектује извршење само оних трансакција које су биле извршене пре грешке. Треба напоменути и да теорија серијализације представљена у овом поглављу не претпоставља да читање и упис су једине операције које база података извршава. Све дефиниције унутар теорије зависе од нотације конфликата и за било који пар p и q дефинисаних на објекту унутар базе. Даље теорија није ограничена на традиционалне базе података, али је такође применљива на објектно оријентисане.

Атомичности и трајност

Да би се осигурала трајност, неопходно је да све трансакције читају вредности написане неком трансакцијом која касније прекида следећу. Овакав начин рада се назива и каскадни. Збор трајности као сообине трансакција, једном када се трансакција изврши, она не може бити накнадно прекинута, нити се њени ефекти не мењају због каскадног прекида. Извршавање је надокнадиво, јел када се трансакција изврши, гарантовано је да неће бити укључена у каскадни прекид и самим тим никада неће поништити своје ефекте.

Дефиниција *Трансакција Tj чита из Ti у историји H ако је*

$$\exists ob(w_i[ob] \rightarrow r_j[ob]) \wedge \neg(\alpha_i \rightarrow r_j[ob]) \wedge \exists T_k((w_i[ob] \rightarrow w_k[ob]) \Rightarrow (\alpha_k \rightarrow r_j[ob]))$$

Другим речима трансакција Tjчита вредности ob написане од Tiако је Ti последња трансакција које је извршила упис у ob и није прекинута пре него што је Tj прочитала ob.

Дефиниција Историја је повратна ако $\forall T_i, T_j \in T_i \neq T_j$ чита из $(T_j, T_i) \Rightarrow (c_i \rightarrow c_j)$

То значи да историја која се може повратити. За сваку трансакцију T_jкоју изврши T_j, прати извршавање сваке трансакцијеT_i из које T_j очитава вредност.

Имајући у виду историју H чија је пројекција серијабилна, не значи и да је Нповратна односно надокнадива. На примеру, следећа историја H'трансакција T_1 и T_2 је серијска са T_1преT_2 , али не и повратна.

$$H' = w_1[x]r_2[x]w_1[y]w_2[x]c_2c_1$$

Како је присуствои конкурентностиа и грешака присутно, тачно извршавање мора бити и серијабилно и повратно. Трансакције и извршења морају имати додатне особине као додатак серијабилности и повратности. Ово упрошћава управљање трансакцијама и минимизује количину рачунања која би можда била неопходна због грешака које би еветуално уследиле.

Извршавање је каскадно ако омогућава да свака трансакција чита само вредност коју направи извршена трансакција,.

Историја је каскадна ако

$$\forall T_i, T_j \in T_i \neq T_j (wi[ob] \rightarrow rj[ob])(ci \rightarrow rj[ob]) \vee (\alpha \rightarrow rj[ob])$$

Строгост (енг. Strictness), је стање јаче од каскадности, које осигурава да свака трансакција чита и пише само оне податке које изврши трансакција

Историја има ову особинуодносно је стриктна ако и само ако

$$\forall T_i, T_j \in T_i \neq T_j \forall q \in \{r,w\} (wi[ob] \rightarrow qj[ob])(ci \rightarrow qj[ob]) \vee (\alpha \rightarrow qj[ob])$$

Разлика између дефиниција је у *rj[ob]* где постаје *qj [ob]* , имплицира да ово је више уопштено да се примени на обе операције уписа и исписа. Даље стриктност подразумева и каскадност. Стриктно извршавање поред заобилажења каскадних прекиа, дозвољава повратност.

Серијализација не одговара увек извршавању трансакција по реду како је дефинисано редоследом. Са две задате неиспреплетане трансакције T_1 и T_2 где се T_1 извршава пре него што се T_2 изврши, могуће је да T_2 претекне T_1 по редоследу серијализације. Чување редоследа је серијско извршавање где све неиспреплетане

трансакције имају исти редослед серијализације и извршавања. Ова особина је корисна за неке апликације где постоји важна привремена веза између неиспреплетаних трансакција . Узмимо у обзир апликацију која се бави аукцијском продајом где је редослед понудаи продаје утиче на цену робе. Приликом извршавања редоследом, извештај трансакције је пренет након извршења аукционе трансакције T_bкојом је заграрантовано да као повратну информацију садржи цену на коју директно утиче T_b. Дакако ова особина није довољно јака да осигура да редослед серијализације и извршавања буду подешени у исто време. Особина која гарантује да извршавање и серијализација буду истовремено се назива ригорозност (од енг.*Rigoroysness*).

Историја је ригорозна ако и само ако $\forall T_i, T_j \in T, T_i \neq T_j$ p,q$\forall$obгде су сучеоне променљиве $(p,q) \wedge (p_i[ob] \rightarrow q_j[ob]) \Rightarrow (c_i \rightarrow q_j[ob]) \vee (\alpha_i \rightarrow q_j[ob])$

Историја је ригорозна ако ни један податак не може бити прочитан или уписан док се трансакција која укључује конфликтну операцију са тим податком не изврши или поништи. Разлика између претходних дефиниција је да w_i[ob] постаје p_i[ob] што значи да cep_i[ob] примењује на читање и упис података. Ово даље имплицира на стриктност уопштено. Као додатак, онемогућавање преписивања података тренутне трансакције која се још извршава, ригорозност историје омогућава да су читања података поновљива (односно читање трансакције података где он враћа исту вредност као и претходно читање истог податка том трансакцијом). Стриктност не омогућава поновљиво читање. Поновно позивање свих поновљивих читања података је неопходна ставка за серијализацију. У ствари разматрањем и контролисањем реда свих конфликтних односно сучеоних операција, ригорозност посаје ограничена форма конфликтне серијалности.

Ригорозност олакшава управљање трансакцијама, конкретно у дистрибуираним базама података како дозвољава детерминисање серијалности контролисањем редоследа извршавања операција.

Степен изолације

Од свих особинаунутар ACID стандарда изолација највише са коегзистенцијом утиче на перформансе система база података.Последично, таква стриктна веза је неопходна због праћења нових трендова који резултују новим захтевима корисника и променом системске архитектуре. Захтеви за коегзистенцијом су се појавили непотребно стриктно за многе типове модификације података. Размотримо, на пример ситуацију са статистичком базом подата где рачунање просечне вредности не захтева стриктну употребу конзистенције. Ако је једна од продајних вредности на пример не коегзистентна, неће креирати никакав проблем унутар рачунања прихватљиве продајне цене. Управљање дужим трансакцијама постаје проблематично унутар равног (енг. flat) модела, и не уклапа се добро унутар мобилних база података.

Број ревизија и пормена које су имплементиране у класичну дефиницију трансакције да би решили ефикасност (време одговора и пролаз) и проблеме са конзистенцијом. Појам конзистенције је ревизиран и категорисан као **Стање 0, Стање 1, Стање 2, Стање 3** користећи појам нечитљивих података, који су заправо неизвршени податци или активне трансакције. Даље су детаљно описана ова стања са дефиницијама.

Стање 0

T_i посматра стање 0 изолацij ако не преписе непотпуне односно нечитљиве податкеT_j, где важи да i ≠ j. ДаљеT_i извршава своје уписе пре крајева. Ово значи даT_j може читати или писати упис који направиT_i и ако сеT_i поврати онда иT_j мора бити враћено. Са тачке гледишта корисника степен 0 изолације није повратан и није ослобођен каскадности.

Пример: Претпоставимо да T_i ,T_jимају повратне операције

T_i write(x)

T_j write(x)

Историја : $w_i(x)w_j(x)$

Ако се иT_i ,T_jизврше конкурентно, онда је коначна вредност x непредвидљива и у неким ситуацијама може доћи до каскадности. Ако причамо о могућности повратка у

стању 0 то није могуће и трансакције је немогуће повратити , а што се тиче серијализације она треба бити краткорочно примењена нах.

Стање 1

- T_i посматра степен 1 изолације акоT_i нема препис преко непотпуних података T_J и
- T_i не извршава упис пре краја.

Ово значи да T_j не може да препише упис који извршиT_i и нема повратка ако је T_i поништен. Премда T_j, је дозвољено да чита податке са T_i пре него што се T_i комплетира последично из каскаде ако је то могуће. Ако причамо о могућности повратка поништити извршење без подешавања закључавања без брисања ажуирања које је T_j направио. Ово је главни разлог зашто сви комерцијални системи база података аутоматски омогуће изолацију 1 степена.

T_i *write(x)*

T_j *write(x)*

Историја: $w_i(x)w_j(x)$

АкоT_iпоништи акцију, може бити враћена вредност x, која захтева да стање 1 има дугорочно заклување вредности x за серијализацију.

Стање 2.

T_iпосматра степен изолације ако:

- T_i не преписује непотпуне податке T_j
- T_i не врши било какав упис пре завршетка тренутне операције и
- T_iне чита непотпуне податке T_j

ДаљеT_iизолује неизвршени део података од T_j. Притом при степену 1 изолацијеT_i може читати непотпуне податкеT_j, који не могу да се десе у 2 степену. Не постоји

каскадност у степену 2 трансакције, Али непоновљиво читање може да се појави као на примеру испод.

T_i:read(x) T_j: write(x)

Историја: $r_i(x) r_j(x) w_j(x)$ commit$_j$ $r_i(x)$

Приликом овог извршавања T_i чита две различите вредности x. Прво чита иницијалну вредност x и онда чита вредности мофиковану од T_j без нарушавања било ког услова изолације. Приликом степена 2 даље T_i има краткорочно закључавање на x приликом серијализације.

Стање3. T_i посматра степен 3 изолације ако

- T_i не преписује непотпуне податкеT_j
- T_i не извршава неке уписе пре него што се заврши
- T_iне чита непотпуне податкеT_j . и
- T_jне чита непотпуне податкеT_iпре него што сеT_iизврши.

Степен изолације дозвољава смањивање флексибилности приступу података при чему 3 степен има најмање флексибилности али нуди потпуну изолацију по ACID особинама. Такав степен изолације података утиче на коегзистенцију носећи са тиме и ниже перформансе. У ствари многи комерцијални системи база података не подржавају 3 степен изолације.

У многиим трансакцијама које су широко заступљење нижи ниво изолације је прихватљив. Ово се поготово односи на мобилне базе података где је време најбитније јел директно утиче на перформансе и трошкове кроз коришћење података преко интернет протокола.

Мобилне базе података

Ово поглавље се бави управљањем трансакцијама унутар мобилног система база података. Представља архитектуру мобилног система , који је заснован на комбинацији рачунарског и GSM система да би утемељио управљање трансакцијама. Речено је раније да стандардни ACID модел не испуњава све задатке. Неки од битнијих су преузимање које је само по себи непредвидиво, штедљиви мод, дисконектован уређај или принудно гашење са мреже, недостатак ресурса као меморије или бежичних канала, присуствolокације и слично. Стога је да би успешно управљали базама , мора бити присутан јачи односно побољшан ACID модел. Број таквих модела извршавања је пожељан. Број таквих модела је направљен али пар њих тренутно је у употреби.

Мобилни системи база података односно (обезбеђују потпуну комуникациону способност базе података и мобилног уређаја. Дозвољава корисницима мобилних уређаја да покрећу трансакције било откуда и било када и гарантује њихову коегзистенцију приликом извршавања истих. У случају било каквог квара или отказа , мобилне базе гарантују повратак. На слици 6 су представљене основне особине.

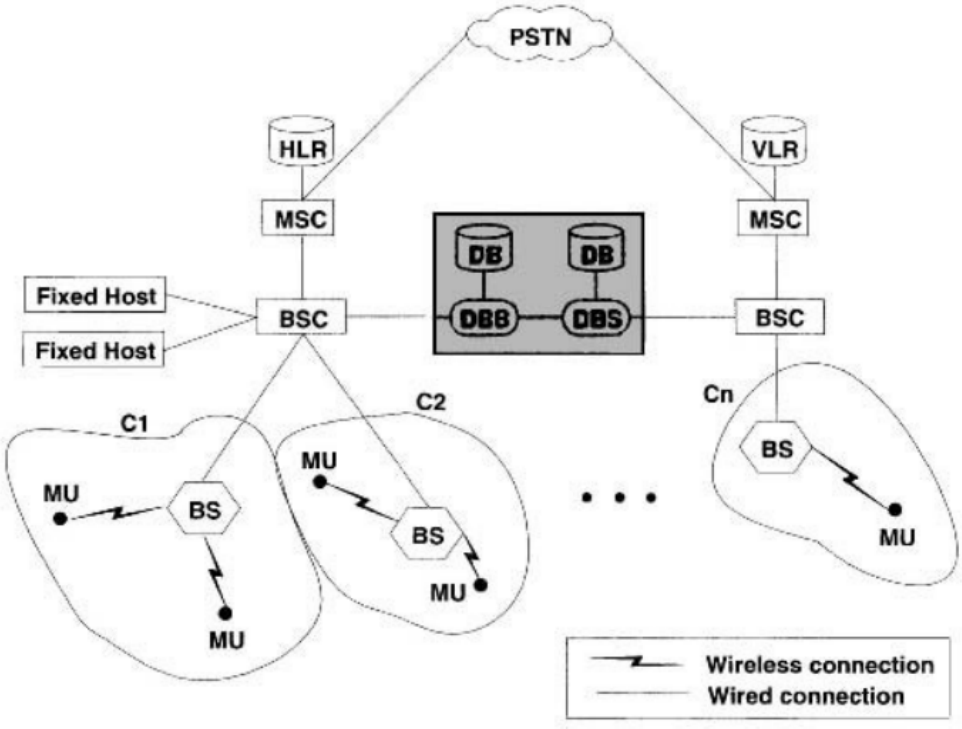
- Географска мобилност: Клијенти су у могучности да се крећу по географском простору без утицаја на процесне могућности и континуирану везу.
- Успостављање и прекид везе: Клијент је у могућности да успостави и прекине везу са сервером било када.
- Опсег обраде података: Клијенти имају до неког нивоа приступ податцима док сервер има потпуни приступ бази података
- Бежична комуникација: Како је мобилност података омогућена кроз мобилну архитектуру, обрада података клијената не утиче на мобилну архитектуру
- Скалабилност: било где у било које време клијент може бити додат или може бити избрисан са мреже.

Мобилне базе података су заправо дистрибуирани вишебазни клијент/сервер системи засновани на PC или GSM. Постоје неке разлике између PC и GSM архитектуре,

дакако оне не утичу на мобилне базе података. Функционалност која је омогућена сетом сервера који су укључени без утицаја на рад мобилне мреже.

Компоненте GSM и PC система су наменски направљење и одговорне за повезивање клијената са системом. Референтна архитектуре мобилних база података је даље описана у оквиру рачунара као што су PC, радне станице, PDA уређаји, паметни телефони.

У мобилним базама података цео скуп рачунара су повезани преко фиксних домаћина (енг. Fixed Hosts) и базних станица . Фиксни домаћини нису везани са трансиверима тако да не комуницирају са мобилним јединицама. Једна или више базних станица је повезана са контролером базних станица BSC , који координира операцијом базних станица користећи уграђен софтвер којим командује мобилни прекидачки центар(MSC Mobile switch center).



Слика 7. Архитектура мобилног система база података

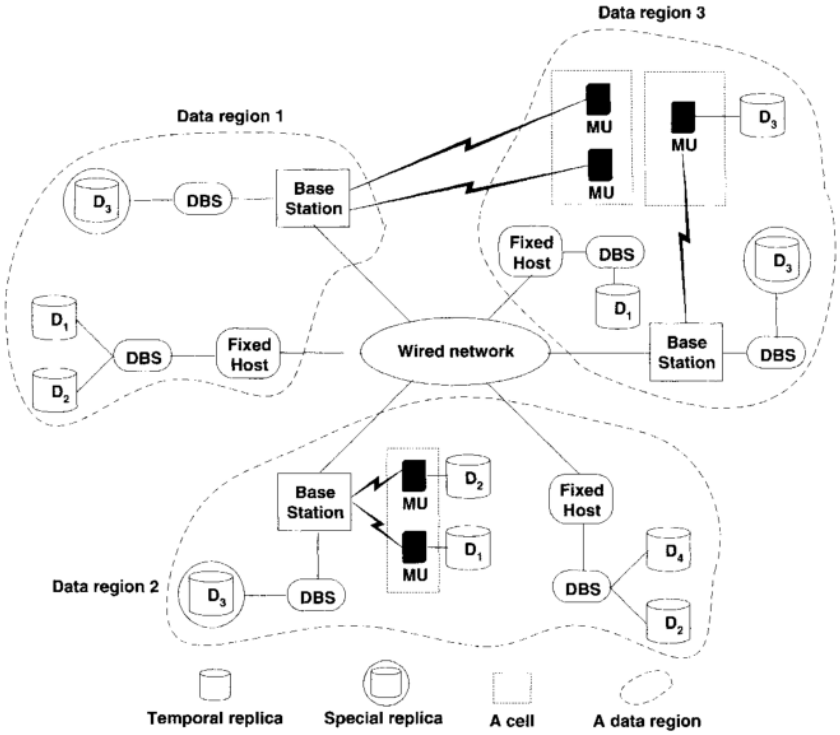
Да би успешно координисао са серверима, неке додатне могућности су урађене у базне станице. Неограничена мобилност у GSM и PC ја је подржана бежичном везом између базних станица и мобилних јединица као што су телефони, паметни телефони, PDA уређаји, лаптоп и сл. Ове мобилне справице се у литератури називају мобилне јединице или мобилни хост ови. Базне станице су опремљене трансиверима и комуницирају са мобилним јединицама преко бежичних канала. Свака базна станица послужује једну ћелију чија вечичина зависи од јачине базне станице. У примени се не користе базне станице високе моћи због много фактора, радије се примењује ниско напонска базна станица за управљане кретањем мобилних јединица.

Да би се употпунила функционалност базе података, неопходно је да се прикључе сервери на персоналне рачунаре или мобилну мрежу. Они могу бити инсталирани

(а) на базне станице или (б) фиксне домаћине-хостове (FH). Даље постоји више проблема приликом ове инталације. Базне станице или FH су прекидачи, и они имају специфичну намену,, која не укључује функције базе података. Да би додали функције базе података цела архитектура базне станице се мора променити, и то је неприхватљиво са аспекта мобилне комуникације. Као додатак овоме та инсталација није ни модуларна ни скалабилна, што би као резултат водило томе да свака промена компоненти база података би ометала комуникацију преко GSM мреже. Због овивх разлога базе података су повезане са мобилним системом преко жичане линије као засебни чворови како је приказано на слици 8. Свакој бази података је могуће приступити преко било које базне станице или фиксног домаћина, или нова база може бити повезана тако да стара буде откачена са мреже без утицаја на мобилну комуникацију. Цео сет мобилних прекидача и PSTN а је повезан на мобилни сервер база података који има додир са спољашњим светом.

Сервер база података комуницира са мобилном јединицом преко базне станице. Корисник остаје конектован у просеку 2 до 4 сата током дана и у сво остало време мора чувати батерију. Да би сачувала снагу, мобилна јединица може бити пребачена да остане у угашеном режиму или будном моду (енг idle) где не комуницира али прати базне станице или активном моду где обрађује податке и слично. Јединица се може кретати са једне ћелије ка другој у било ком режиму, на пример возач може угасити свој мобилни да сачува батерију док вози и може проћи многе покривености базних станица. Тотална поктивеност мобилног система база података је дата изразом $\{C_1 + C_2 + ... + C_n\}$.

Мобилни систем база података има неколико базних сервера који могу бити дистрибуирани преко парцијалне и тоталне репликације. Мобилни систем база података може бити управљан вишебазним системом. Просторно реплицирање мора пратити ограничења локалне зависности и локалних података.



Слика 8. Различити типови реплицирања

Слика 8 илуструје како се база података дистрибурира преко система база података и мобилних јединица. Локацијски независни податци D_1,D_2, и D_4 могу бити реплицирани на све регионе и могу имати исте вредности. Ово је један од разлога што привремене репликације могу бити процесиране користећи читај-једном упиши- све верзије дистрибуираног дво фазног механизма који је непримењив за просторну репликацију. Локацијски зависни податци примера ради D_3 могу исто бити реплицирани на све регионе али сваки регион ће имати другачију вредност. Три основна типа репликације постоје на овој слици (а) нема репликације (б) традиционални дистрибуиран систем (в) локацијски

зависна репликација. Објекат D_4 нема репликацију,а објекти D_1 и D_2 су традиционалне реплике копиране на регионе 1 и 2. Дистрибуиране реплике су копија једне друге које могу иматиразличите вредност привремено али тамо постоји само једна тачна вредност. Локацијски зависни податци D_3 имају више копија и више тачних вредности. Где се тачна вредност одређује локацијом. Реплицирана је на сва три региона. Сваки регион има засебну вредност за D_3. Вредност D_3 која постоји унутар базне станице у региону 3 је реплицирана на мобилну јединицу, али има исте вредности као сервер базе података. Ово је пример привремене репликације која је локално зависна. На пример информација о телефонским бројевима је просторно реплицирана где је податкована грануларност повезана са локалним мрежама и тарифама. Хотелске информације могу имати грануларност повезану са поштанским бројевима. Сваки податак уникатно дефинише и одређује тип репликације и грануларност ћелије.

	Репликација	**Репликација**	**Репликација**
Копије	Једна	Мулти	Мулти
Тачне вредности	Једна	Једна по времену	Једна по локацији и времену
Архитектура	Централизована		Мобилна
Мобилност	не	не	Да

Табела 7. Сумирање репликације података

Табела 7 сумира разлике између три типа репликације података. Приметити да су реплике које су кеширане у мобилној јединици привремене тј да могу имати привремене реплике просторних (објекат O_3 на слици 8). Тако да са задатим простором могу постојати привремене копије података о локацији. На пример мобилна јединица може имати привремену копију информације о хотелу у Канзасу. Ово може омогућити путнику да комотно изврши и прегледа локацију иако је мобилни уређај дисконектован са мреже. То значи да мобилна јединица се креће са једне регије на другу, различите вредности просторних координата могу бити потребне за процесуиране упите.

Извршавање трансакција у мобилним системима

Дистрибуирани систем база података упошљава максималну паралелизацију при процесуирању оперативног обима посла да би побољшала системске перформансе. Имплементира паралелно процесирање фрагментисањем трансакција у већи број подтрансакција које су извршене у више чворова. Укупно извршавање захтева софтверски модул који се назива координатор, који управља сетом активности које воде до гашења трансакције. Постоје три начина да се имплементира координатор (а) централизован (б) делимично реплициран (в) тотално реплициран. У централизованом приступу један од чворова је координатор, у парцијално-делимичном систему подсет чворова служи као координатор, и у тотално реплицираном сви чворови координирају. У мобилним системима база података због лимитирања ресурса, само централизовани приступ делује приступачним.

Индентификација координиатора унутар мобилног система база података

У мобилном систему база података идентификација координатних чворова је процес где координатор мора имати (а) директну и сталну комуникацију са осталим чворовима (б) теоретичку неограничену и сталну снагу и велики простор (в) велику поузданост и доступност. За разлику од конвенционалних система база података, постоји неколико типова процесних чворова унутар мобилног система база података, али само неки могу да имају улогу координатора која је објашњена испод.

- Мобилна јединица не може омогућити сталну повезаност због непредвидивих прекида, има ограничену меморију и електричну снагу, и лични је извор корисника који може бити угашен стога она не може служити као координатор.

- Дистрибуирани систем база је континуално доступан и има довољно простора. Он ипак нема директну комуникацију са осталим чворовима и није опремљен трансиверима за бежичну комуникацију. Као резултат тога не може бити координатор.
- Фиксни хост је опремљен са свиме осим трансивера. Као резултат тога он не може комуницирати директно с мобилним уређајима, стим да и он не може бити координатор.
- Мобилни сервис центар задовољава све критеријуме да буде координатор, сто да мора да користи сервере да би дошао до базне станице. Није довољно ефикасан начин али може да послужи као координатор.
- Базне станице поседују се услове и критеријуме, стим да ако им додамо и ову улогу неће доћи до преоптерећења. Као резултат тога базне станице се најћешће употребљавају као координатори односно додељује им се та улога.

Процесирање трансакција унутар мобилног система

Трансакције унутар мобилног система база података могу бити инициране од стане базе, мобилне јединице или обе. Могу бити процесиране у потпуности на мобилној јединици одакле су и покренуте или на серверу базе података или комбинацијом. Ако се процесуирају у потпуности на једном чвору, када координатор игра мању улогу у извршавању трансакције. Када је више чворова укључено у извршавање може доћи до следећих ситуација.

- Мобилна јединица: три могућа сценарија су у оптицају када трансакција потиче са мобилне јединице: (а)да се трансакција изврши на мобилној јединици преко сервера базе података и да крајњи резултат буде послат на мобилну јединицу, (б)трансакција се процесира само на бази и да резултат буде послат на мобилну јединицу и(в) да се само обради на мобилној јединици. Важно је напоменути да

ниједна друга мобилна јединица не би требало да буде укључена у извршавање трансакције зато што мобилна јединица је лично средство којим клијент располаже и не жели да дели ресурсе и податке. Коначан резултат се у било која три случаја враћа на мобилну јединицу која је и покренула трансакцију.

- Потиче са мобилне јединисе: Три могућа ссенарија су могућа када трансаксија потиче са Мобилне јединисе: (а) Трансаксија се извршава на МЈ и сет сервера БП и коначан резултат иде на МЈ, и трансаксија је просесуирана само на МЈ. Важно је напоменути да ни једна друга МЈ не сме бити умешана у извршавање трансаксије зато што је (а) МЈ је лични ресурс клијента који може бити и искључен или дисконектован са мреже од стране свог корисника у било које време и да (б) корисник мобилне јединисе можда не жели да дели своје ресурсе. Коначан резултат иде на МЈ која инисира трансаксију.
- Потиче са сервера: Трансакција потиче са сервера и извршава се на бројним серверима БП. Коначан резултат иде на онај сервер који иницира трансакцију.

Када мобилна јединица извршава трансакцију она проверава да види да ли трансакција може бити потпуно извршена локално на мобилној јединици. У овом случају мобилна јединица изврпава и шаље трансакцију помоћу кешираних података. На крају извршавања, мобилна јединица шање трансакције систему база података за инсталацију на базу и шаље резултате кориснику. Ако не може у потпуности да се изврши на мобилној јединици, онда постоје 2 опције.

- **Пребачај података са базе на мобилну јединицу**. МЈ индентификује сет података који је неопходан да би трансакција била извршена. Комуницира са БС, која досеже до сервера БП, и пребацује податке на кеш меморију МЈ. МЈ извршава и спроводи трансакцију користећи податке кеширане у меморији. На крају извршења, МЈ шаље трансакционо ажурирање серверу БП да би се инсталирали на базу и послали као резултат кориснику. Ако је немогуће извршити их на мобилној јединици, Постоје две опције.
- **Пребацити податке са сервера БП на МЈ**: МЈ идентификује сет података који су неопходни да би процесирали трансакцију. Комуницира са базом података, која приступа серверу БП и помера податке у кеш меморију МЈ. МЈ извршава и

спроводи трансакцију и саље конацни резултат кориснику. Такође шаље ажурирање преко своје БП на сервер БП да би их инсталирао у БП. Ова шема не захтева сервисе координатора али генерише доста непотребне комуникације.

- **Дистрибуирана трансакција на сет чворова**. Мобилна јединица дели трансакцију на бројне подтрансакције. Чува подтрансакције које може да изврши, и успомоћ координатора, дистрибуира остатак подскупова сервера база података за извршавање. Тренутна БС МЈ постаје координатор трансакције и управља целим извршавањем водећи до тога да се или изврши или поништи. Ова шема генерише компаративно мање комуникационо загушење и трошкове база података.

Кретање МЈ чини посао координатора тешким који захтева извршавање и правилно кретање МЈ. Са мобилношћу три сценарија су могућа:

МЈ се не помера: Трансакција пристиже и комплетира се на МЈ. МЈ се не помера за време извршавања трансакције. Ово је у случају да не постоји било какво кретање за време извршавања трансакције.

Мобилна јединица се помера. Трансакција пристиже и комплетно извршава на мобилној јединици. Захтевани податци се померају овде са других чворова. За време трансакције извршавање на мобилној јединици се помера на различите ћелије. Овај тип извршавања се назива локално атомски који је често заступљен у мобилном рачунарству.

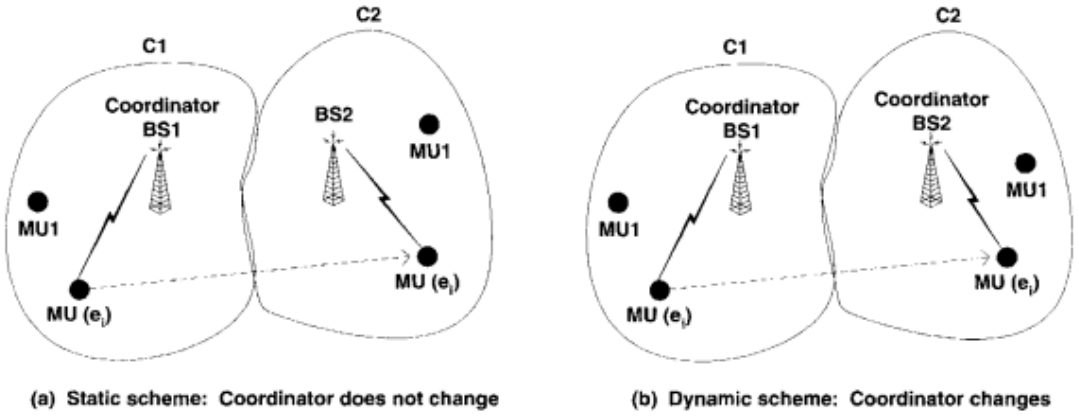
Дистрибуирано процесирање када се мобилна јединица помера: Трансакција је започета са мобилне јединице и није потпуна. Под трансакције су смештене уздуж мобилне јединице и сета БП. Координатор је индентивикован као онај који извршава радњу. Мобилна јединица се премешта на друге ћелије док се трансакција не изврши.

Сврха координатора је та да мобилна јединица мора бити доступна у континуитету за време извршавања транскције. Ова веза може бити прекинута када мобилна јединица пређе границу тј. домет једне ћелије и уђе у домет друге, Ова веза се може одржати на 2 начина:

- **Статички метод**: Када трансакција потиче са мобилне јединице, онда је базна станица ове мобилне јединице координатор трансакција И задржава ту улогу док се трансакција не заврши. Мобилна јединица може наставити да

иде са једне на другу ћелију док процесира подтрансакције али координатр се не мења. Ова шена се налази на слици 7.3. Мобилна јединица се помера са ћелије Ц1 на Ц2 са њеном подтрансакцијом и успомоћ БС2.

Одговорност је Мобилне јединице да информише нову базну станицу И идентитету свог координатора. Ово може бити постигнуто када се мобилна јединица региструје са новом базном станицом која комуницира са координатором чим сазна свој идентитет. Проблем са овим једноставним методом је да коотдинатор И мобилна јединица морају да прођу кроз више базних станица да би се ажурирали. Ово чини да извршење трансакције буде прилично сложено.



Слика 9..Промена координатора због мобилности

- Динамички метод. У овом методу координатор се помера са мобилном јединицом. На пример како је показано на слици 7.3 б, када се мобилна јединица помера на ћелију Ц2, њена базна станица БС 2 постаје координатор трансакције која се извршава на мобилној јединици.. Како се трансакција процесуира на више БП , оне морају знати када је нови координатор додељен на постојећу трансакцију. Ово је постигнуто помоћу новог координатора који информише све Бп о овој трансакцији да идентификује

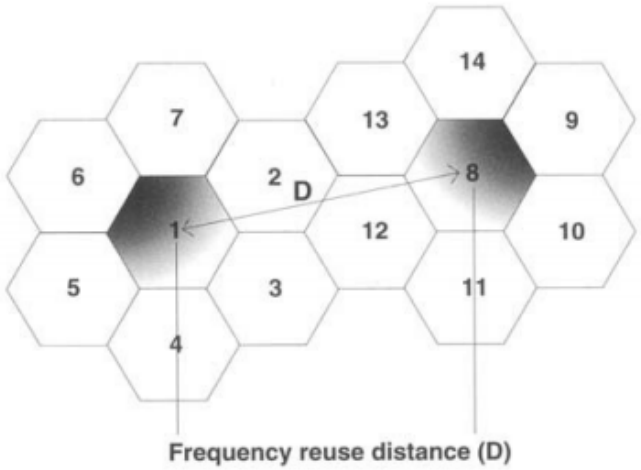
новог координатор. На приер на слици 9 БС 2 је нови координатр који информише све укључене БП којима је БС 2 координатор трансакције.

Премда у динамичком методу мобилна јединица је директно повезана са координатором цело време, долази до потешкоћа за високо мобилну јединицу. Узмимо у обзир ситуацију где мобилна јединица се брзо помера од Ц1 до Ц2. У овој ситуацији могуће је да процес мењања координатора и предаје може бити несинхронизован. Као резултат тога БС2 може остати координантор док МЈ је назад у Ц1 . У овом високо мобилном случају промена координатора није пожељна,

Две шеме могу бити примењене да управљају координатором за високо мобилни сценарио.

Резидентно ограничење: На овој шеми максимално трајање мобилне јединице је колико МЈ може остати на ћелији пре него што координатор промени како је унапред дефинисано. На овој шеми координатор мигрира на тренутну БС само ако је МЈ на ћелији више него што је дефинисано. Под овом шемом координатор мигрира на тренутну базну станицу ако и само ако је резиденција МЈ унутар ћелије дуже од унапред дефинисаног лимита. Даље на фигури 9 координатор се креће од БС1 ка БС2 ако је резинденција МЈ у Ц2 дужа од предефинисане вредности.. вредност резидентног лимита мже бити примењена на све МЈ или може бити уникатна на специфичну МЈ. Бројни параметри као померање, загушење саобраћаја, време, и тако даље могу бити употребљени да дефинишу резидентни лимит. Физичко кретање и пригушење саобраћаја могу варирати значајно дакако они могу вероватно представити или оцртавати неку шему. На пример у јутарњем шпицу МЈ може бити веома покретљива и саобраћај може бити

Несуседна миграција: Под овом шемом координатор се креће само ако МЈ се пребаци на несуседну ћелију. Несуседне ћелије су оне које директно немају прелаз односно контакт преко којих МЈ пролази. На пример на слици 9 ћелије 2,3,4,5,6 и 7 су несуседне у односу на 1 ву и остале као 9, 10 ,11, 12, 13 , 14 су несуседне на ћелију 1. Идеја је да се МЈ помери уназад до ћелије 1 са сусдедне тако да не би било потребе да мењамо координатора. Ова шема је у употреби.



Слика 10. Приказ распореда ћелија

Модел извршавања заснован на ACID окружењу

Концепт ACID трансакција је представљен бо очувања коегзистентности. По дефиницији "трансакција је колекција или сет операција у физичком и апстрактном стању". Конвенционални трансакциони модел осигурава да се ACID особине очувају приликом обраде базе података. Представљање мобилности значајно мења целу архитектуру БП и парадигму управљања, и постаје јасно да стриктно придржавање ACID осоинама није бас до крајње мере неопходно да би се одржала конзистенција. Даље мобилност се мења и у много случајева мора да буде флексибилна зато што је коегзистенција уско везана за локације у географском смислу, како следи.

Географски домен G је укупно подручје покривено свим мобилним јединицама мобилне мреже. Даље $G = (C_1 + C_1 +_{...} + C_n)$ где C_i представља подручје у коме се налази ћелија.

Локација је прецизна тачка унутар Географског домена. Представља најмању тачку која може да се идентификује у мрежи . Свака локација је идентификована специфично. Такође $G = \cup L$, $\forall L$ и $C_i = \{L_1, L_2, \ldots L_m\}$,

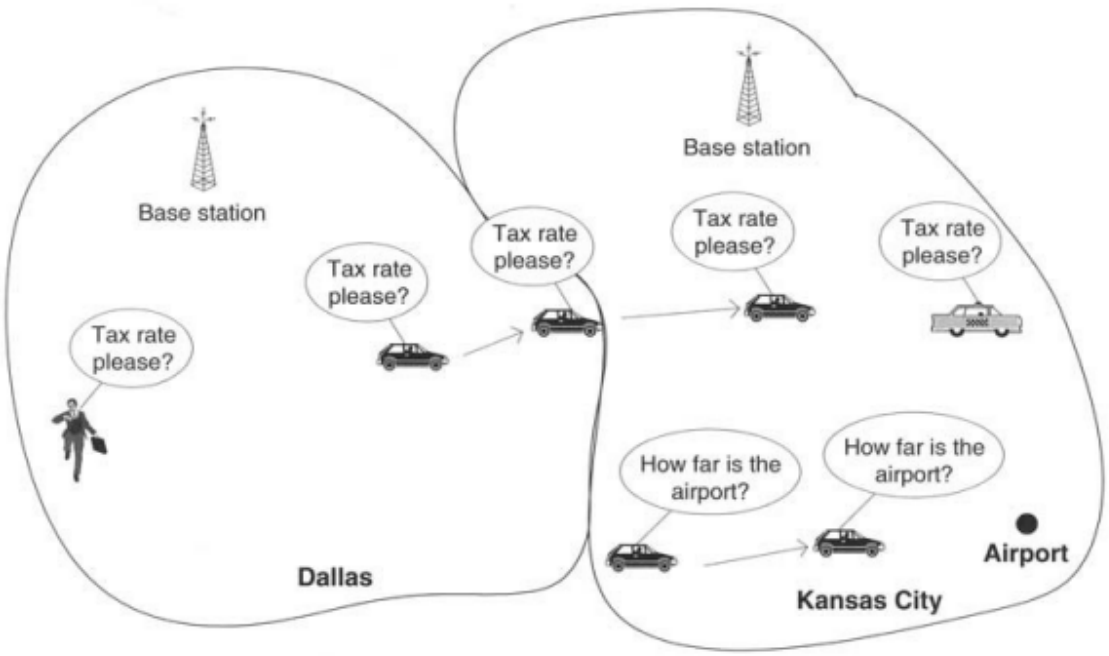
У стварности локација мобилне јединице (MJ) се идентификује као референца у односу набазну станицу. Ако је Географска мрежа негде на планети, онда ће неко помислити да је локација однос географске дужине и ширине . Дакако грануларност локације може бити далеко већа. На пример, локација може бити адреса, држава или земља. У постојећим системима, учесталост приступа податковним ставкама (енг. дата итемс), а не њихова веза са географским локацијама се користи у дистрибуцији (подела и делимична репликација). У Мобилним БП ова веза чини важну улогу у процесуирању како и дистрибуцији истих. Слика 10 илуструје неке важне тачке. Претпоставимо да особа путује аутом од Даласа ка Канзасу и упита колика је цена карте. Одговор на овај упит ће дати цену карте за Далас али ако је Канзас онда за њега. Сада претпоставимо да ће упит да се понови два пута за време путовања-једном у Даласу па затим у Канзасу. Добићемо две тачне вредности. Даље путник може да пита која је температура у Даласу? Одговор неће зависити од карактеристика места где је упит постављен.

Сада размотримо следећи упит од истог агента. Нађи имена и адресе најефтинијег хотела. Одговор на овај упит ће зависити од места одакле потиче зато што је у сваком граду друга цена. У свакој организацији, овакви типови података постоје. На пример осигуравајућа компанија има број пословница по разним градовима и државама. Ове просте реалне ситуације сугестирају да тачан одговор зависи од географске локације упита и субјекта коме се треба дати одговор.

Упит који зависи од локације (енг. Location Dependent Querry LDQ) је упит који процесира податке који зависе од места или (енг Local Dependent Data LDD). Да би процесирао LDQ, мобилна БП мора одлучити да ли је упит везан или не за локацију. Локација упита може бити одређена и учињена доступном процесору да обради тачне резултате. Процес који се назива локално везивање се користи да повеже упите и одреди тачне резултате. Тачне дефиниције података који зависе од локације и просторне репликације могу бити прегледане на основу мапирања одређеног објекта на локацију. Дакле ако имамо задати објект, мапирање идентификује тачне регионе за сваки објекат. За специфичан објекат, подскуп главног скупа R идентификује податковне регионе за тај објекат где P представља сет свих могућих региона.

Дефиниција 1. Са задатим скупом објеката D и сетом регија података P. Мапирање података (енг. Data Location Mapping) је релација DLM:D → P(R), где DLM (D)={R1,R2,…Rn},Ri ∈ R Ri=G i ∀I, j $R_i \cap R_j = Ø$.

Даље сет података за сваки објекат је партиционисање географске мреже. У случају да не



Слика 10.Упит који је везак за локацију

постоји просторна репликација објекта који већ постоји, број региона података је један. Са задатом специфичном локацијом L и G сваки објекат се асоцира са уникатним регионом. Са задатим сетом података D, сет локације је L и сет региона података је R. па је онда мапирање података представљено као DRM:D x L → R где је DRM (<D,L>) ∈ DLM (D).

Oracle DATABASE lite 10g

Oracle Database Lite 10g је интегрисано и потпуно решење за брзи развој апликација за мобилне и преносиве платформе.Велике корпорације га користе због велике поузданости, редуковања трошкова и обезбеђујући брже и боље коришћење услуга ка крајњим корисницима.

Три главне карактеристике Oracle Database Litea су:

1) Клијент стек, укључујући Berkeley DB, који је доступан на различитим платформама укључујући Android, iPhone (Apple iOS), Blackberry, Symbian OS, Windows 32 bit, Windows Mobile, Linux и друге мобилне платформе.

2) Мобилни сервер за синхронизацију, и скалабилну расподелу и управљање апликацијама, корисницима и уређајима

3) Развојне алате који омогућавају брз развој апликација.

Oracle Database Lite клијент укључује разне компоненте да би омогућиле потребе корисника и лаку администрацију. Мобилни клијент подржава SQLite и Oracle Berkeley DB.

SQLite је најпопуларнији уграђен (енг. Embedded) систем база података и широко је заступљен на разним платформама укључујући Android и Blackberry уређаје. Мало простора заузима, трансакциону базу која се сама администрира, не захтева спољашње DBA акције.

BDB је веома широко заступљен уграђени систем који обезбеђује фирмама разне погодности као што је велики опсег упита, скалабилност, кеширање података унутар меморије са малим захтевима за меморију. BDB нуди SQL API који је комплетно компатибилан са SQLite овом библиотеком, док при томе задржава лакоћу коришћења као у SQLite API.

Мобилни клијент подржава обе базе података са синхронизацијом. Зависно од платформе, синхронизациони агент може синхронизовати податке у позадини аутоматски, или GUI апликација може омогућити мануално односно ручно позивање синхронизације.

На неким платформама, агент на мобилном уређају дозвољава командовање да би се мобилно управљано или слале команде и упути ка систему и мобилним податцима коначно на неким платформама, ажурирање омогућава управљање целим животним циклусом.

Решење за фирме је Мобилни сервер, само језгром Oracle Database Lite производа, који може бити инсталиран на неки сервер који користи стандардан фабрицки систем Windows, Linyx, Solaris, HP UX или IBM AIX.

Мобилни сервер омогућава поуздану двосмерну синхронизацију и моћни интерфејс за администрацију.

Велики и скалабилан систем који поседује Oracle Database Lite омогућава асинхроно и синхроно инкременирање између мобилних корисника и Oracle ове базе података.

Oracle Database Lite врши повратак података након пада мреже такође. Ако се корисник сретне са падом мреже или сличном дисфункицијом, Oracle Database Lite клијент ће наставити функцију од последње контролне тачке и рестартовати пренос односно трансмисију.

Мобилни сервер анализира информацију садржану у мобилној апликацији да би аутоматски креирао апликативни сервер са логичком синхронизацијом. Oracle Database Lite омогућава флексибилну архитектуру која дозвољава прилагођавање синхронизационог процеса на више нивоа Позив уназад омогућава испреплетаност разлличитих апликационо специфичних задатака за време различитих фаза синхронизације. Девелопери ће моћи да одаберу само фазу ресурса тако што ће имплементирати јава класе које подржавају тај у дата модел. Алтернативно, развојни тим ће обезбедити потпуну контролу над синхронизацијом тако што ће независно да управљају податцима који садрже ажурарање од стране клијента које ће тек бити преузето.

Do конфликта може доћи када се исти податци мењају и од стране сервера и клијента. Oracle Database Lite аутоматски детектује такве конфликте и регулише их на основу подесиве резолуције управљања и правила које се задају од администратора.

Сложени менаџмент и администрација које Oracle Database Lite омогућава је сигурни ценстрални репозотиријум, са уникатним интерфејсомо за дистрибуцију и управљање софтвера на мобилним системима. Мобилни управљач, административни web програм за мобилни сервер, омогућава 100% тно сервер сиде управљлање над свим апликацијама уређајима, корисницима и мобилним серверима. Администратор може креирати кориснике и групе, додавати привилегије, слати команде уређајима и добити дијагностику о сваком интерфејсу појединачно.

Интеграција са Oracle OID и LDAP директоријумима даље појашњава управљање корисницима. Као додатак, администратор може управљати процесом синхронизације подешавајући њену фреквенцију, управљање грешкама, или анализирање и попдешавање перформанси користећи исти интерфејс. Даље скриптинг језик може бити коришћен за управљање административним функцијама и да смањи непотрбну администрацију.

Убрзано развијање и подршка су такође карактеристике Oracle Database Lite користећи Mobile Development Kit (MDA)који је цео скуп алата, API ја, инструкција и примера које могу убрзати развој мобилних апликација. Mobile Database Workbench (MDW)је визуелни алат за креирање реплицираних база података. Помоћници унутар MDW а убрзавају креирање реплицираних база омогућавајући девелоперима да брзо дефинишу и прилагоде идеје о моделима који се могу применити на реплицираним базама података. Помоћник (енг wizard) омогучава спајање свих компоненти (JSP/servlets, execytables, DLL, i sl) у један .jar фајл да би се лакше отпремио на мобилни сервер одакле може бити имплементиран односно смештен на мобилни или неко преносно окружење.

Девелопери могу да користе Oracle Jdevelopersa ADF Mobile да би визуелно развијали апликације које омогућавају приступ критичним податцима. Oracle Database Lite укључује подршку за имплементирање и управљање апликација креираних унутар Jdeveloper и ADF mobile.

Oracle Database Lite такође подржава слични приступ податцима и стандарде као ODBC, JDBC, и ADO.NET. упутства и примери који су садржани у документацији и MDKуказује како да се поступа са специфичним особинама и развијају апликације на конкретној платформи.

Велике перформансе и скалабилност је нешто што пружа Oracle Database Lite, омогућавајући корисницима да приступе информацији брзо и ефикасно. Мултипроцесор и динамички кеш омогућавају врхунске перформансе за веће базе података и веће бројеве конектованих корисника. Oracle Database Lite омогућава да се фино подесе особине синхронизације података.

Database Lite подржава такође више конфигурација типа један или више корисникакоји примењују конфигурисање гарантујући да ће Oracle Database Lite апликације бити компатибилне и да ће испунити захтеве са сваком променом окружења. Mobile Server интеграција са WebLogic Serverom 11g и Oracle Application Server om(OAS) омогућавају скалирање Mobile Server примењујући предности које доносе њихове особине а то су мало оптерећење и сл.

Са стране мобилних уређаја, Berkeley DB мало потраживање простора, ексремна скалабиност, и добар систем закључавања су је начинили погодном за сваку апликацију. Подржава висок ниво конкурентности, укључујући подршку конкурентних Vacuum и Back up команди. Сигурност унутар Oracle Database Lite се омогућава као и преко инфрастуктуре имплементирањем нових команти да подржавају пословне процесе. На пример , хоћете да издате команду да синхронизујете целу базу података, извршите дијагностику, или промените подешавање апликације. У случају губитка података, крађе и слићно , мошете обрисати све апликације и базе подата, обрисати клијент и променити шифру на мобилном уређају.

SSL енкрипција штити интегритет података док је у транспорту између уређаја и фирмине базе података.

Општи закључак је да Oracle Database Lite омогућава боље пословање, смањује трошкове, и задовољава потребе корисника. Oracle Database Lite има ефекта кроз Sales force аутоматизацију, управљање корисницима (CRM), и поље примена у разним институцијама као што су финансије, здравсто, транспорт, логистика, и слично.

IBM DB2

DB2 Everyplace je решење које се користи на мобилним платформама и уграђеним embedded системима. Са DB2 Everyplace, апликације које користе предузећа и податци су доступни на мобилним уређајима.

DB2 Everyplace је заснована на трослојној архитектури која садржи базу података фирме, сервер, и апликацију на мобилном уређају. Синхронизација, укључујући и ажурирања се обавља успомоћ сервер.DB2 Everyplace је присутан на разним уређајима од Palms, Pocket PC ја и мобилних уређаја. Sa DB2 Everyplace, можете приступити податцима са било ког места.

DB2 Everyplace има две главне компоненте а то су: мобилна база података која се налази на уређају и сервер који врши синхронизацију који се налази у главној станици са којим се мобилни уређај синхронизује. Сервер је одговоран за слање забелешки крајњој бази података.Трећа компонента односно клијент за синхронизацију, је у додиру са крајњим корисником. Ослања се на уређај и има двосмерну улогу.

Развојни тимови имају висе опција приликом развоја апликација наDB2 Everuplace. DB2 Everyplace Mobile Aplication Builder , који је део DB2 Everyplace Software Developer Kit (SDK) , је визуелни алат за брзи развој решења и апликација за мобилне уређаје. SKD такође има и додатак за AppForce MobileVB који дозвољава Microsoft Visual Basic развојним тимовима да развијуDB2 Everyplace апликације. Java и Java Server Page tools могу бити креиране помоћу WebSphere Studio Device Developera.

Архитектура као што је раније у тексту поменуто се састоји од три нивоа . База података унутар предузећа је било која која подржава IBM DataPropagator . Server хостира DB2 Everyplace сервер за синхронизацију, DB2 Universal базу података, и језгро односно енгине. Мобилни уређаји могу бити било којиPocket PC, Palm OS, или Windows 32 битни систем, QNX Neytrino, Symbian,или Linyx.

Мобилна база података је релационог типа која услужује простор на сервер који користе мобилне апликације. Ово омогућава корисницима да приступају прегледају и

ажурирају податке са њихових мобилних уређаја. Што се тиче простора захтеви су веома мали око 180 KB , дозвољавајући коришћење на мобилном уређају.

Сервер за синхронизацију је клијент-сервер заснован који управља двосмерном синхронизацијом између извора и циљане базе.Ова синхронизација дозвољава и мобилном кориснику да ажурира податке, и мобилном уређају да прихвати податке са базе података. Серверза синхронизацију комуницира са клијентом на мобилном уређају. Сервер пружа GUI да би се побољшало управљање мобилним уређајима, групама, апликацијама.

Синхронизација

За време синхронизације, промене које се врше над податцима на мобилном уређају или главној бази података су у комуникацији. Синхронизација мора бити инициаа са мобилног уређаја.Када се захтев направи на мобилном уређају, послат је у центар за синхронизацију на радној станици преко мобилног клијента. Апликација на мобилним уређајима има претплату односно субскрипцију ка контролној бази података на сервер. Ова претплата дозвољава промене да буду реплициране на главну контролну базу података. Промене се смештају унутар табеле. Податци се онда копирају на идентицну табелу користећи JDBC позиве. Дупликати се проверавају и бришу. Након тога се табела умногжава ка изворној користећи DB2 DataPropagator.

Синхронизација од изворне базе података ка мобилној се врши обрнутим редоследом. Промене на изворној бази су сачуване помоћу DB2 DataPropagatora, који их уписује на табелу. Ове промене се примењују на мање базе података, и дупликати се проверавају и регулису. Након тога сервер за синхронизацију одговара поруком на мобилном уређају, и промене сеизвршавају на мобилној бази података.

Примене су разне на пример у колаборацији IBM Zurich лабораторије и авио компаније AirSwiss, је дозвољено корисницима да чекирају летове користећи њихове мобилне уређаје. Након уласка на аеродроме, мобилни телефон је детектован. Информације о лету намењене кориснику се шаљу на мобилни телефон. Корисник потврђује лет, и пролаз се издаје. Ово омогућава корисницима да брже и једноставније обаве цео процес.

NetSetGo је развио апликацију за свој дистрибутивни ланац. Њихова апликација дозвољава продавцима да смештају и бележе наруџбине са интернет или њихових мобилних уређаја. Ово омогућава продавцима да обезбеде њиховим клијентима боље цене и бржу услугу. NetSetGo каже да наруджбине бивају испуњене у пар минута, пре него што је некада требало класичном методом.

Asatte Systems је развио мобилни патент који се користи у болницама и клиникама. Доктори и медицинска лица користе PalmOS уређаје који имају инсталиран DB2 Everyplace, и могу да ажуирају историје болести пацијената. Као додатак, доктори могу да праве графиконе и табеле да би упоређивали историју или ток болести. На почетку дана медицинско особље односно мед техничари преузимају фајл о пацијенту на Palm. Након тога преузимају графиконе и талене осталих пацијената који се лече од сличних или истих болести и упоређују. Доктори уносе податке директно у Palm користећи поља за унос података која су специфична за њихову област рада(говоримо о докторима специјалистима). Уређаји су синхронизовани на крају дана и сви податци су смештени у централну базу болнице.

SQLite

SQLite је релациони систем за управљање базама података који је садржан у оквиру програмског језика C. За разлику од осталих решења за управљање базама података, SQLite није одвојен процес коме се приступа преко клијенске апликације већ је он део исте.

SQLite је компатибилан са ACID стандардом који користи и SQL, користећи динамичку и сличну синтаксу као SQL.

SQLite је популаран избор за уграђене базе података (енг.embedded) за локал-клијент смештај апликација као што су web прегледачи. Чињеница је да је најраспрострањенији међу осталим решењима, и користе га многи интернет прегледачи, оперативни системи, и уграђени системи- SQLite има повезаност са многим програмским језицима.

За разлику од осталих клијент-сервер система за управљање базама података, SQLite се не ослања на засебни процес са којим апликациони програм комуницира, већ SQLite библиотека постаје део апликативног програма. Библиотека се позива динамички. Апликативни програм користи SQLite функционалност кроз неколико позива ка функцијама које редукују кашњење у адресирању базе података: Функција која се позива у једном процесу је ефикаснија од оне која је међупроцесна. SQLite чува целу базу података (дефиниције, табеле, и податке) као један фајл на домаћину (енг.host). Имплементира једноставан дизајн закључавајући целу базу података за време уписа. SQLite чита операције које могу имати више задатака (енг. myltitasking) , кроз упис могу само да се изврше секвенцијално.

SQLite је дизајниран 2000 од стране Др. Ричард Хипа док је радио за фирму General Dynamics док је радио војне пројекте. Он је дизајнирао наменске софтвере, који су били засновани за HP UX са IBM Informix базама . Циљеви SQLite а су били да дозволе програму да ради без инсталирања додатног система за управљање базама или захтева за администратором базе података. У августу 2000 1.0 верзија SQLite је изашла, заснована на gdbm (GNU Database Manager). SQLite 2.0 је заменила gdbm са B tree имплементацијом са

подршкама за трансакције. SQLite 3.0 је додао интернационалзиацју, и друга важна побољшања.

2011 Нир је изјавио да планира да дода UnQL интерфејс на SQLite базама података и да развије UnQLite уграђени систем за обраду докумената и база. Хауард Чу је портовао SQLite 3.7.7.1 за употребу у Openldap MDB уместо оригиналног Btree code и назвао gasqlightning. Један тест уметања 1000 podataka je bio 20 пута бржи од стандардног SQLite a.

Sqlite имплементира већином SQL 92 стандард за SQL стим да пар функција нема. На пример има делимичну подршку за тригере. Док подржава комплексне упите, има лимитирану ALTER TABLE подршку, и не може да модификује или брише колоне.

SQLite користи необичан систем за SQL компатибилни DBMS, уместо да усагласи тип уа колону као увећини SQL система, типови су усаглашени са индивидуалним вредностима, у језичким појмовима где се то динамички користи, премда сличности има са Perl ом, где мошете унети словни податак у колону предвиђену са бројеве (иако ће SQLite пробати да га претвори у бројни податак прво ако је унапред колона дефинисана као број). Ово додаје флексибилност колонама, посебно када су везане за динамички скриптинг језик. Дакако ова техника не може бити имплементирана на остала SQL решења. Уобичајено SQLite се налази на теми критичара због наводног мањка интегритета података који се уносе у статичке колоне у другим решењима. SQLite описује додатни мод као „стриктни афинитет“ , али то још није додато. Ипак може се имплементирати са ограничењима типа CHECK (typeof (x)=’integer’) .

Неколико процеса могу приступати бази података истовремено. Неколико читања могу бити вршена паралелно. Приступ упису података може бити извршен само ако ниједан други процес га не користи моментално. У другом случају, упис пропада са кодом и грешком. Конкурентни приступ ће се променити приликом рада са тренутним табелама , ово је регулисано у верзији 3.7, где је приступ упису и логирању (WAL writte ahead logging) укључен омогућује омогућавајући читање и упис података у исто време.

SQLite и Програмски језици

SQLite има повезаност са следећим програмским језицима:

- BASIC
- Delphi
- C
- C#
- C++
- Clipper/Harboyr
- Common Lisp
- Cyrl
- D
- Delphi
- Free Pascal
- Go
- Haskell
- Java
- Javascript
- Jylia
- Livecode
- Lya
- newLisp
- Objective C
- Ocalm
- Perl
- PHP
- Pike
- Pyre basic
- Python
- R
- REALbasic
- REBOL
- Ryby
- Scheme
- Tcl
- Visyal Basic
- Xojo

Интернет прегледачи:

- Google Chrome, Opera, Safari и Android прегледач дозвољава смештај информација, Sqlite база унутар прегледача која је заснована на Web SQL технологији.

- Mozilla Firefox и Mozilla Thynderbird смештају већину конфигурационих података (забелешки, колачића (eng cookies), контакти и сл) у интерну SQLite базе података, и нуди додатак који може управљати SQLite базама података.

Развојна окружења

- Bugzilla, Mozilin дебагер написан у Perl који користи SQLite да смести податке и подешавања.
- Django, Python веб развојно окружење подржава SQLite 3 као подразумеван стандард.
- Као и у верзији 7 Drypal, PHP заснован систем за управљање сајтовима и блоговима, има опцију да инсталира Lite
- Ryby on Rails поседује систем за управљање који користи SQLite3.
- Web2py, Python веб окружење, као подразумевани систем за БП користи SQLite.

Оперативни системи

Због малих меморијских захтева, SQLite је прилагођен уграђеним ембедед системима и такође је садржан у:

- Blackberry's BlackBerry 10 OS
- Microsoft's Windows Phone 8
- Apple's iOS
- Symbian OS
- Nokia's Maemo
- Google's Android
- Linyx Foyndation's MeeGo
- NetBSD
- OpenBSD
- Free

Синтакса SQLite језика

У наставку ће бити наведене основне и најчешће коришћене функције унутар SQLite језика.

1.SQLite COUNT Function

SQLite COUNT функција се користи да израчуна број редова унутар табеле.

2.SQLite MAX Function

SQLite MAX функција се користи да одабере максимум вредности за задату колону

3.SQLite MIN Function

SQLite MIN функција се користи да одабере минимум вредности за задату колону.

4.SQLite AVG Function

SQLite AVG функција се користи да одабере средњу вредност за задату колону..

5.SQLite SUM Function

SQLite SUM функција се користи да израчуна суму бројева унутар колоне.

6.SQLite RANDOM Function

SQLite RANDOM функција враћа псеудо насумични целобројни податак у распону од -9223372036854775808 and +9223372036854775807.

7.SQLite ABS Function

SQLite ABS функција враћа апсолутну вредност нумеричког аргумента.

8.SQLite UPPER Function

SQLite UPPER функција преправља стрингове односно слова у велика слова.

9.SQLite LOWER Function

SQLite LOWER функција преправља стрингове односно слова у мала слова

10 SQLite LENGTH Function

SQLite LENGTH функција враћадужину низа.

11. SQLite sqlite_version Function

SQLite sqlite_version функција враћа каон вредност верзију SQLite библиотеке

Интеграција са програмским језиком Python

SQL lite се може веома лако интегрисати саPython програмским окружењем коришћењем модула sqlite3 који је написао Герхард Харинг. Модул могућава SQL интерфејс преко DB API 2.0 спецификације. Нема потребе да се додатно инсталирају неки додаци и слично јел долази у оквиру инсталације Python верзије 2.5 и даље.

Да би користилиsqllите 3 модул, морамо прво креирати објекат који представља базу података и онда опционално се може креирати курсор објекат који ће помоћи у извршавању SQL команди.

Python sqlite модули и API

У даљем тексту следе све важне sqllите3 наредбе, које могу олакшати рад са SQLite базом података.

1. **sqlite.connect(database[,timeoyt, other optional argyments])**

Овај API отвара конекцију ka SQLite бази података унутар фајла. Можете користити меморију да би отворили везу ка бази података где ће иста бити смештена у радну меморију а не на хард диск. Ако је база отворена успешно, враћа објекат везе.

Када се бази података приступа са вишеструким везам, и једна од њих модификује базу података, SQLite база је закључана док се обавља трансакција. Временски параметар

дефинише колико ће се дуго чекати на откључавање. Подразумевана вредност је 5 секунди.

Ако задато име базе података не постоји онда ће ово креирати нову базу података. Може се доделити име са одговарајучом путањом било где осим у тренутном директоријуму.

2. **connection.cursor([cursorClass])**

Ова рутина креира показивач који ће се користити током програмирања базе потадака унутар python-а. Овај метод прихвата појединачне параметре cursorClass. Ако је додат, може бити додатна класа која се даље шири као sqllite3.Cursor.

3. **cursor.execute(sql [, optional parameters])**

Ова рутина извршава SQL наредбу. SQL наредба може бити параметарска. Конкретно sqlite3 модул подржава две врсте резервисаних места: упите и именована места.

На пример: *cursor.execute("insert into licni_podatci(?,?)",(ko, godište))*.

4. **connection.execute(sql[, optional parameters])**

Ова рутина је пречица до горепоменуте методе за ивршавање коју омогућава објекат и креира показуивач објект позивајући његову методу, и наконн тога позива наредбу за извршавање курсора са задатим параметрима.

5. **cursor.executemany(sql, seq_of_parameters)**

Ова рутина извршава SQL команду насупрот сви параметарским секвенцама или мапирањима које се налазе у оквиру SQL секвенце.

6. **connection.executemany(sql[, parameters])**

Ова рутина је пречица која креира показивач позивајући његову методу и након тога позивacursor.s да се изврши са задатим параметрима.

7. **cursor.executescript(sql_script)**

Ова рутинапозива вишеструке SQL наредбе у исто време у форми скрипте. Извршава COMMIT наредбу прво, након чега извршава SQL скрипту и узима као параметар. Све SQL наредбе треба да буду одвојене симболом(;).

8. **connection executescript(sql_script)**

Ова рутина је пречица који креира показивач позивајући његову методу, након што он изврши методу са задатим параметрима.

9. **connection.total_changes()**

Ова рутина враћа укупни број редова базе података који су били модификовани, убачени или обрисани од како је успостављена веза са базом података.

10. **connection.commit()**

Ова метода извршава тренутну трансакцију. Ако се иста не позове, све штто сте урадили до последње call to commit() команде није видљиво осталим везама са базом података.

11. **connection.rollback()**

Ова метода враћа све промене на бази података пре задње команде commit().

12. **connection.close()**

Ова метода затвара везу са базом. Напомена да ово не позива аутоматски commit(). Ако се само затвори база без позивања команде commit() претходно, све промене ће бити изгубљене.

13.**cursor.fetchone()**

Ова метода се повезује на следећи ред упита, враћајући једну секвенцу, или ништа када нема доступних података.

14. **cursor.fetchmany([size=cursor.arraysize])**

Ова рутина везује следећи сет редова који је резултат упита, враћајући листу. Празна листа се враћа само ако не постоје редови. Метода покушава да повеже што више редова како је дефинисано параметром величине.

15. **cursor.fetchall()**

Ова рутина везује све преостале редове који су резултат упита, враћајући листу. Празна листа се враћа само ако не постоје редови.

Повезивање са базом података

Следећи код ће демонстритати како се повезује база података, ако она не постоји исту ћемо креирати и објекат ће бити резултат тога.

```
#!/ysr/bin/python
import sqlite3
conn = sqlite3.connect('test.db')
print "Baza podataka je uspešno otvorena";
```

Креирање табеле

Следећи код се користи да се креира табела

```
#!/ysr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')

print "Baza je otvorena uspesno";
```

```
conn.execyte('''CREATE TABLE COMPANY
      (ID INT PRIMARY KEY NOT NULL,
      NAME        TEXT       NOT NULL,
      AGE         INT        NOT NULL,
      ADDRESS     CHAR(50),
      SALARY      REAL);''')
print "Tabela je kreirana yspesno";
      conn.close()
```

```
      Baza je otvorena yspesno

      Tabela je kreirana yspesno
```

INSERT операција

Следећи код демонстрира како се убацују вредности унутар табеле

```
#!/ysr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execyte("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Payl', 32, 'California', 20000.00 )");

conn.execyte("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execyte("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execyte("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print "Records created successfully";
conn.close()
```

```
Opened database successfully
Records created successfully
```

SELECT операција

```
#!/ysr/bin/python

import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

cursor = conn.execyte("SELECT id, name, address, salary from COMPANY")
for row in cursor:

      print "ID = ", row[0]
      print "NAME = ", row[1]
      print "ADDRESS = ", row[2]
      print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

Када се код изврши приказаће следећу поруку:

```
Opened database successfully
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

UPDATE операција

```
#!/usr/bin/python
import sqlite3

conn = sqlite3.connect('test.db')
print "Opened database successfully";

conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID=1")
conn.commit
print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
      print "ID = ", row[0]
      print "NAME = ", row[1]
      print "ADDRESS = ", row[2]
      print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

Када се код изврши приказаће следећу поруку:

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY = 25000.0
ID = 2
NAME = Allen
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY =  20000.0

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

DELETE операција

```
#!/usr/bin/python
import sqlite3
conn = sqlite3.connect('test.db')
print "Opened database successfully";
```

```
conn.execute("DELETE from COMPANY where ID=2;")
conn.commit
print "Total number of rows deleted :", conn.total_changes
cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
print "ID = ", row[0]
print "NAME = ", row[1]
print "ADDRESS = ", row[2]
print "SALARY = ", row[3], "\n"
print "Operation done successfully";
conn.close()
```

Када се код изврши приказаће следећу поруку:

```
Total number of rows deleted : 1

ID = 1
NAME = Paul
ADDRESS = California
SALARY = 20000.0

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY = 20000.0


ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

Интеграција са програмским језиком Java

За почетак рада морају бити инсталирани SQLite JDBC пакет и Java (JDK и развојно окружење (Eclipse или Netbeans))

Поступак повезивања је описан у следећем коду , стим да ако база не постоји објекат ће бити враћен и база ће бити креирана.

```
import java.sql.*;
public class SQLiteJDBC
{
  public static void main( String args[] )
  {
    Connection c = null;
    try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
    } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
      System.exit(0);
    }
    System.out.println("Opened database successfully");
  }
}
```

Сада је неопходно да се компајлира и покрене овај програм да би креирали базу податакаtest.db у тренутном директоријуму. Може се мењати путања само једном по захтеву. Претпостављамо да је софтвер `JBDC driver sqlite-jbdc-3.7.2.jar` је доступан у тренутној путањи.`$javac SQLiteJDBC.java`

```
$java -classpath ".:sqlite-jdbc-3.7.2.jar" SQLiteJDBC
Open database successfully

$javac SQLiteJDBC.java
$java -classpath ".;sqlite-jdbc-3.7.2.jar" SQLiteJDBC
Opened database successfully
```

Креирање табела

Следећи код се користи приликом креирања табела унутар претходно креиране базе података.

```
import java.sql.*;
public class SQLiteJDBC
{
  public static void main( String args[] )
  {
    Connection c = null;
    Statement stmt = null;
    try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      System.out.println("Opened database successfully");
      stmt = c.createStatement();
      String sql = "CREATE TABLE COMPANY " +
"(ID INT PRIMARY KEY NOT NULL," +
" NAME TEXT NOT NULL, " +
" AGE INT NOT NULL, " +
" ADDRESS CHAR(50), " +
" SALARY REAL)";
      stmt.executeUpdate(sql);
      stmt.close();
      c.close();
      } catch ( Exception e ) {
      System.error.println( e.getClass().getName() + ": " + e.getMessage()
);
      System.exit(0);
    }
    System.out.println("Table created successfully");
  }
}
```

Кад се овај код изврши добићемо табелу са називом Company у бази test.db и коначан листинг након тога изгледаће овако:

```
-rw-r--r--. 1 root root 3201128 Jan 22 19:04 sqlite-jdbc-3.7.2.jar
-rw-r--r--. 1 root root 1506 May 8 05:43 SQLiteJDBC.class
-rw-r--r--. 1 root root 832 May 8 05:42 SQLiteJDBC.java
-rw-r--r--. 1 root root 3072 May 8 05:43 test.db
```

INSERT команда

Следећи јава програм ће показати како можемо креирати податке унутар табеле које смо претходно креирали

```
import java.sql.*;
public class SQLiteJDBC
{
      public static void main( String args[] )
      {
      Connection c = null;
      Statement stmt = null;
      try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      c.setAutoCommit(false);
      System.out.println("Opened database successfully");

      stmt = c.createStatement();
      String sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
                   "VALUES (1, 'Paul', 32, 'California', 20000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
                   "VALUES (2, 'Allen', 25, 'Texas', 15000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
                   "VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );";
      stmt.executeUpdate(sql);

      sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) " +
                   "VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );";
      stmt.executeUpdate(sql);

      stmt.close();
      c.commit();
      c.close();
 } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage());
      System.exit(0);
      }
      System.out.println("Records created successfully");
}
}
```

Када се програм компајлија и покрене, унеће додате податке унутар табеле и избациће следећу поруку

```
Opened database successfully
Records created successfully
```

SELECT операција

У следећем програму показаћемо како се повезују и приказују одређени пдоатци из табеле у овом случају.

```
import java.sql.*;
public class SQLiteJDBC
{
      public static void main( String args[] )
      {
      Connection c = null;
      Statement stmt = null;
      try {
            Class.forName("org.sqlite.JDBC");
            c = DriverManager.getConnection("jdbc:sqlite:test.db");
            c.setAutoCommit(false);
            System.out.println("Opened database successfully");
      stmt = c.createStatement();
      ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANY;" );
      while ( rs.next() ) {
            int id = rs.getInt("id");
            String  name = rs.getString("name");
            int age  = rs.getInt("age");
            String  address = rs.getString("address");
            float salary = rs.getFloat("salary");
            System.out.println( "ID = " + id );
            System.out.println( "NAME = " + name );
            System.out.println( "AGE = " + age );
            System.out.println( "ADDRESS = " + address );
            System.out.println( "SALARY = " + salary );
            System.out.println();
      }
      rs.close();
      stmt.close();
      c.close();
 } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage()
);
      System.exit(0);
 }
  System.out.println("Operation done successfully");
```

```
}
}
```

Када се програм компајлира добијемо следећи резултат

```
Opened database successfully
ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 20000.0

ID = 2
NAME = Allen
AGE = 25
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

UPDATE операција

Пратећи дати код у јави показује како можемо користити ову операцију да ажурирамо било који податак унутар наше табеле.

```
import java.sql.*;
public class SQLiteJDBC
{
public static void main( String args[] )
{
Connection c = null;
Statement stmt = null;
try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      c.setAutoCommit(false);
```

```
      System.out.println("Opened database successfully");

      stmt = c.createStatement();
      String sql = "UPDATE COMPANY set SALARY = 25000.00 whereID=1;";
      stmt.executeUpdate(sql);
      c.commit();

      ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANY;" );
      while ( rs.next() ) {
            int id = rs.getInt("id");
            String  name = rs.getString("name");
            int age  = rs.getInt("age");
            String  address = rs.getString("address");
            float salary = rs.getFloat("salary");
            System.out.println( "ID = " + id );
            System.out.println( "NAME = " + name );
            System.out.println( "AGE = " + age );
            System.out.println( "ADDRESS = " + address );
            System.out.println( "SALARY = " + salary );
            System.out.println();
      }
      rs.close();
      stmt.close();
      c.close();
      } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " +e.getMessage() );
      System.exit(0);
      }
System.out.println("Operation done successfully");
 }
}
```

Када се програм изврши резултат је следећи

```
Opened database successfully

ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 25000.0

ID = 2
NAME = Allen
AGE = 25
ADDRESS = Texas
SALARY = 15000.0

ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0
```

```
ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

DELETE операција

```
import java.sql.*;
public class SQLiteJDBC
{
public static void main( String args[] )
{
      Connection c = null;
      Statement stmt = null;
      try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:test.db");
      c.setAutoCommit(false);
      System.out.println("Opened database successfully");

      stmt = c.createStatement();
      String sql = "DELETE from COMPANY where ID=2;";
      stmt.executeUpdate(sql);
      c.commit();

      ResultSet rs = stmt.executeQuery( "SELECT * FROM COMPANY;" );
      while ( rs.next() ) {
      int id = rs.getInt("id");
      String  name = rs.getString("name");
      int age  = rs.getInt("age");
      String  address = rs.getString("address");
      float salary = rs.getFloat("salary");
      System.out.println( "ID = " + id );
      System.out.println( "NAME = " + name );
      System.out.println( "AGE = " + age );
      System.out.println( "ADDRESS = " + address );
      System.out.println( "SALARY = " + salary );
      System.out.println();
      }
      rs.close();
      stmt.close();
      c.close();
}     catch ( Exception e ) {
System.err.println( e.getClass().getName() + ": " + e.getMessage() );
System.exit(0);
}
System.out.println("Operation done successfully");
```

```
}
}
```

Када се програм изврши друга особа је обрисана са списка

```
Opened database successfully
ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 25000.0

ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0
Operation done successfully
```

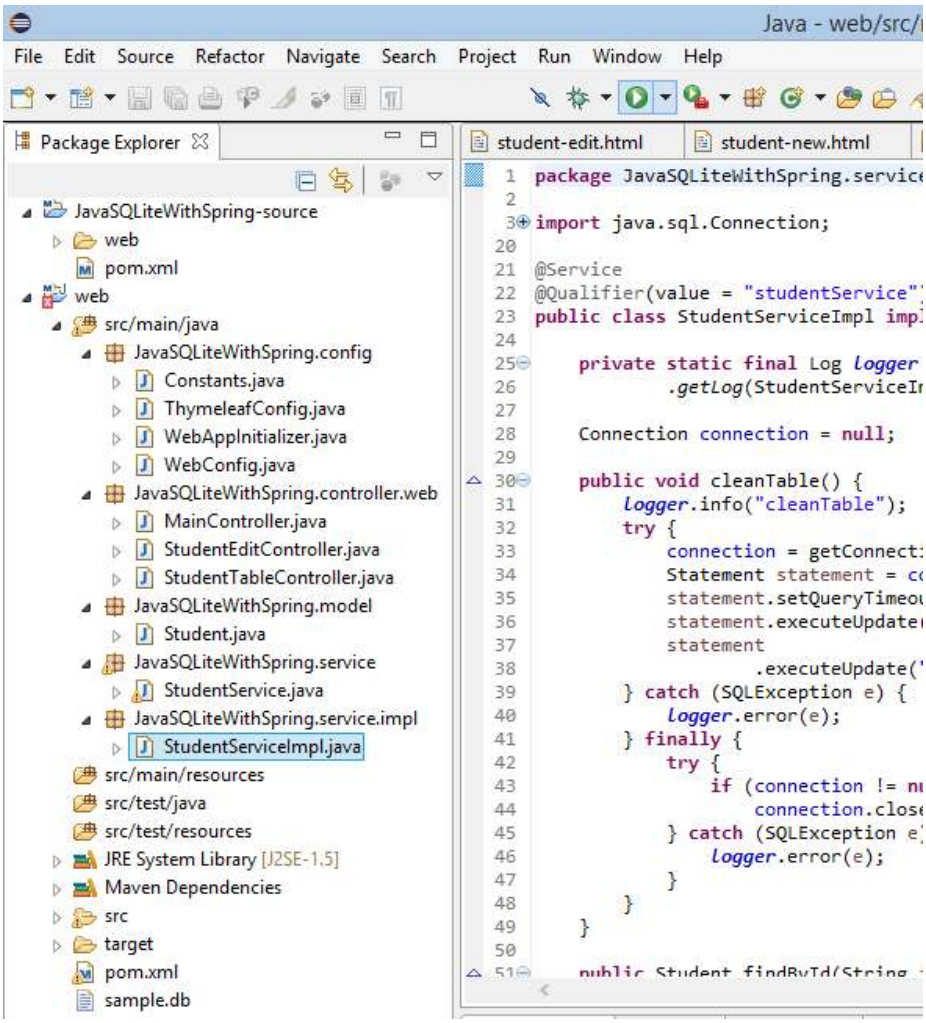
Java Веб Апликација

Веб апликација је апликација којој се приступа од стране корисника преко мреже као што је Интернет или интранет. Термин такође означава и примену компјутерског софтвера који је кодиран у веб претраживачу који подржава програмске језике (као што су JavaScript, у комбинацији са језиком који служи за обележавање као што је HTML) и ослања се на заједничком веб претраживачу да донесе извршну апликацију.

Веб апликације су популарне због свеприсутности веб претраживача, а погодност коришћења веб прегледача као клијента се понекад зове танки или мршави клијент. Способност да се ажурира и одржава веб апликације без дистрибуције и инсталирања софтвера на хиљаде потенцијалних клијенатских рачунара је кључни разлог за њихову популарност. Уобичајене веб апликације укључују слање поште, интернет трговину, аукцију, и многе друге функције.

Кроз Java, JavaScript, DHTML, Flash, Silverlight и друге технологије, омогућене су методе, као што су цртање на екрану, репродукцију звука и приступ тастатури и мишу. Многи сервиси су искомбиновали све ово у познатом интерфејсу који је близак изгледу оперативног система. Технике опште намене као што је превуци-и-пусти такође подржавју ове технологије. Веб програмери често користе скриптове на клијетској страни да допринесу функционалности, а посебно да се створи интерактивни доживљај који не захтева освежавање страница. Недавно, технологије су развијене тако да координирају скриптове на клијентској страни са технологијама на серверској страни као што је PHP. Ајах, техника за веб развој која користи комбинацију различитих технологија, пример је технологија која ствара интерактивно искуство.

Апликација се састоји од више модула слика 11.



слика 11. Апликација

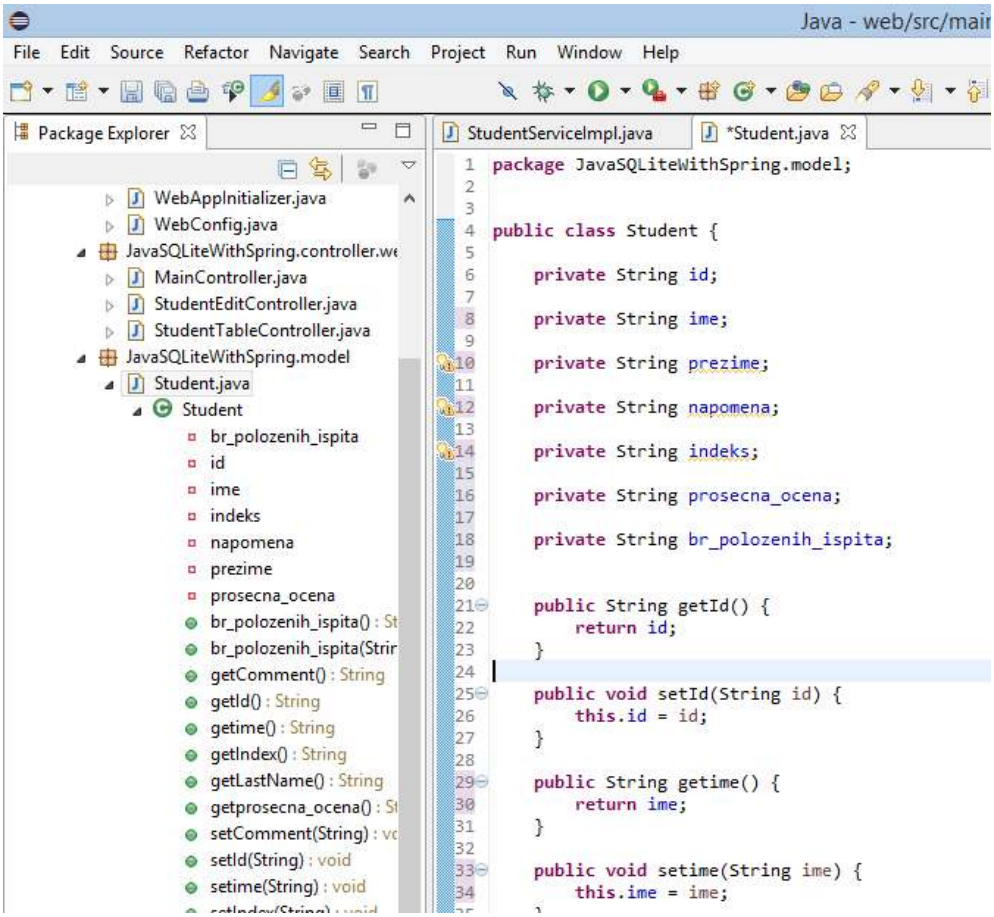
Основни елементи ове апликације су :

- `StudentServiceImpl.java ->` Веза са базом података и упити
  Referenca: https://bitbucket.org/xerial/sqlite-jdbc
- `Student.java ->` главни ентитет
- `MainController.java ->` ту су handler и за маин и цлеан стране
- `StudentTableController.java ->` ту је handler за table страна

- `StudentEditController.java ->` ту је handler и за креирање нових студената, уређивање студент, брисање студента

Коришћени су додатци:

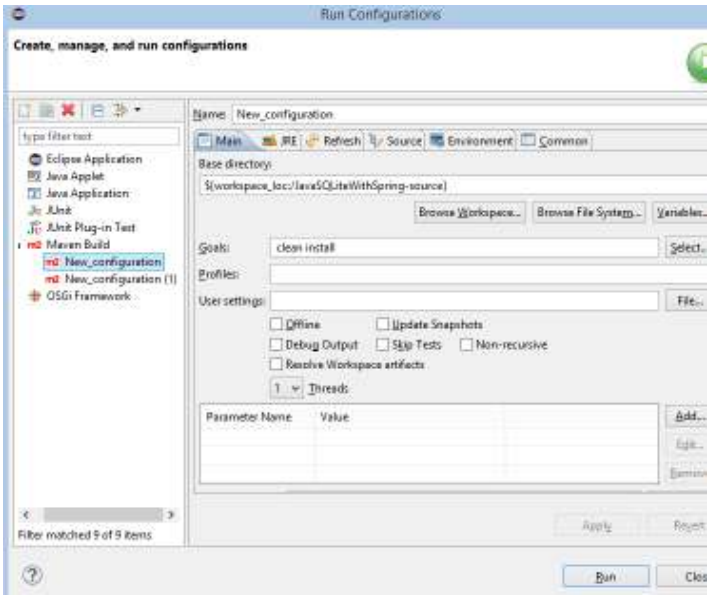
- web: Spring MVC + за темплејт и HTML5 са Thymeleaf + Bootstrap за CSS
- service: Spring и Java JDBC са SQLite driver ом
- build: Maven
- server: Jetty преко Maven



Slika 12. Sadržaj fajla student.java

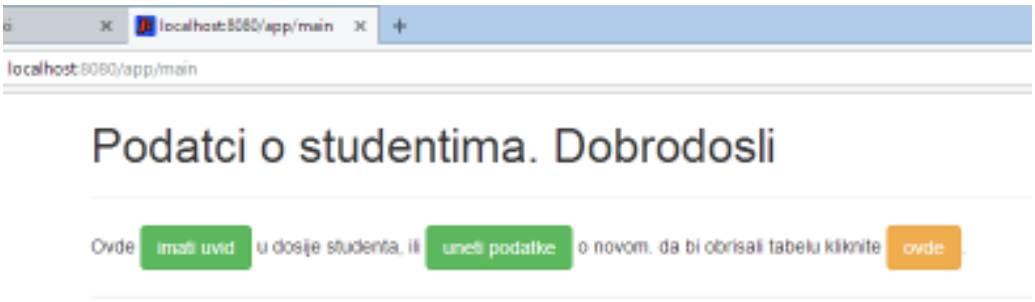
У фајлу student.java се дефинишу промењиве које желимо да користимо из базе података за коју користимо мобилну платформу sqlite.

Да би покренули ову web апликацију користимо Maven и пре покретања је неопходно припремити локални сервер који користи ресурсе односно sqlite базу података. То се врши на следећи начин тако што се у опцији run-customise слика 13 укуца наредба clean install, након чега се активира преузимање неопходних параметара за Maven локални сервер.

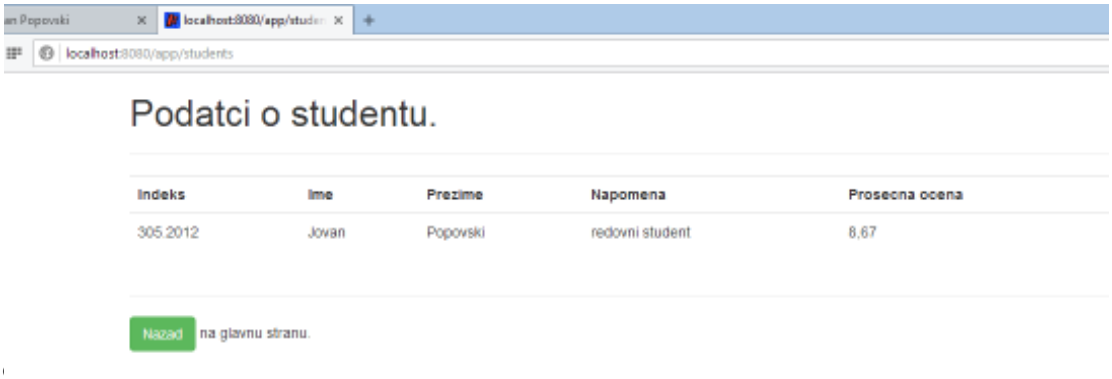


Слика 13. Подешавање сервера

Након тога неопходно је креирати још једну секвенцу коју ћемо назвати jelly start да би покренули сервер на локалу. Када покренемо то добијамо прозор као на слици где имам приступ целокупној БП. Поља уносимо ручно.

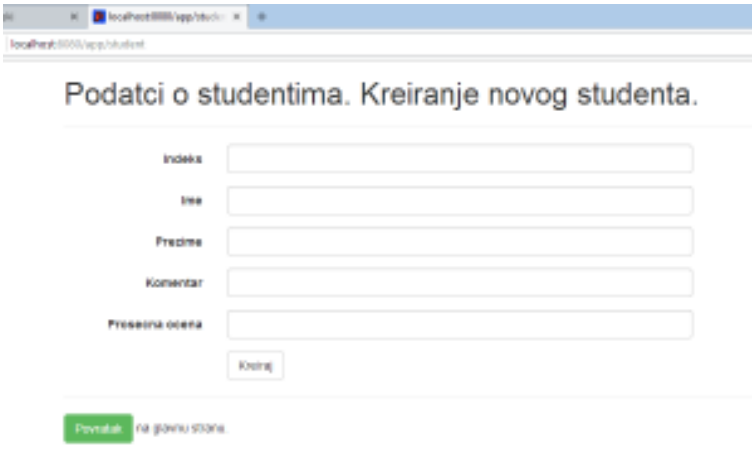


Слика 14. Почетна страна



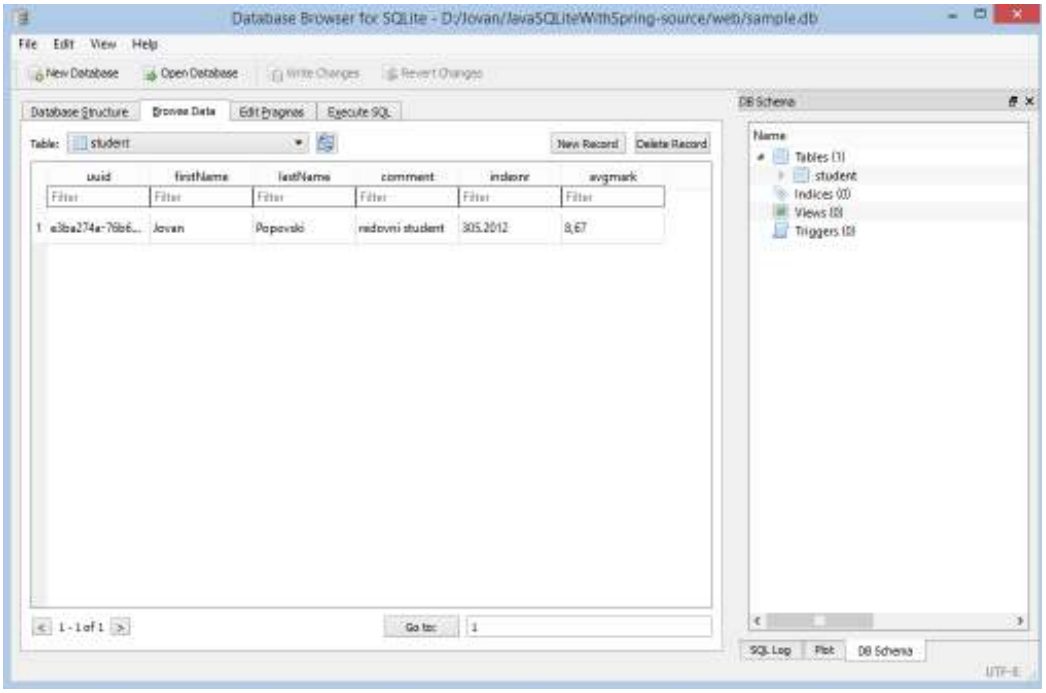
Слика 15. Податци о студенту

Даље је могуће додавати нове студенте у менију



Слика 16. Унос података

База података коју смо креирали уносом у табелу могуће је прегледати преко SQLite browser а као на слици 17.



Слика 17. SQLite browser

Закључак

Системи мобилних база података се свакодневно много користе. На овај начин се омогућава доступност подацима и могућност њиховог синхронизованог ажурирања без обзира на просторну или временску позицију. Другим речима, формира се глобални, повезани информациони простор између чијих чворова постоји стабилна комуникациона веза. Ови системи представљају сложене целине које повезују већи број модерних технологија. Технологије везане за рад са базама података и методе карактеристичне за њихов поуздани рад, мобилне телекомуникационе технологије, мобилне хардверске технологије, софтверске технологије, обједињене у целину пружају читав спектар различитих услуга и могућности. Оно што на недвосмислени начин карактерише описани систем јесте мобилност. Постоји више праваца и путева развоја ове дисциплине, али влада општеприхваћени концензус да је основни концепт система базираних на мобилним базама података управо покретљивост, доступност разним врстама информација и приступ услугама различитих сервиса у покрету.

Литература

[1] Vijay Kumar / Miobile Database Systems, Computer Science an Informatics University of Missouri.

[2] Elmasri/Navathe, Fundamentals of Database Systems, Fourth Edition.

[3] Oracle® SQL Developer Supplementary Information for IBM DB2 Migrations Release 3.0

[4] http://www.ibm.com/developerworks/ibmi/library/i-mobile_database/

[5] http://www.tutorialspoint.com/sqlite/