Факултет инжењерских наука Универзитета у Крагујевцу

Марина Ј. Радовић

Системи за управљање базама података

завршни рад

Крагујевац, 2013.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Базе података

Број индекса: 21/2010

Марина Ј. Радовић

Системи за управљање базама података

завршни рад

Комисија за преглед и одбрану:

1. **Др Милан Ерић - ментор**
2. ______________________________ Датум одбране:_____________
3. ______________________________ Оцена:____________________

У оквиру овог завршног рада кандидат треба да

Проучи литературу и презентира основе система за управљање базама података. Завршни рад треба да обухвати уводна разматрања везана за појам и основне карактеристике система за управљање базама података, опис архитектуре, управљање трансакцијама, бекаповање и рестаурирање, као и опис дела конкретног система за управљање базом података. На крају дати закључна разматрања.

Препоручена литература:

1. Лазаревић Б. ….: ***Базе података***, ФОН Београд, Београд 2003.
2. Павловић-Лажетић Г.: ***Основе релационих база података***, Математички факултет, Београд, 2000.
3. Ross Mistry, Stacia Misner: ***Introducing Microsoft SQL Server 2008***, Microsoft Press, Washington, 2010.
4. Michael Lee, Gentry Bieker: ***Mastering SQL Server 2008***, Wiley Publishing, Inc., Indianapolis, Indiana, 2009.

Крагујевац, датум

ментор:

Др Милан Ерић, доцент

____

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

Резиме

У даљем тексту биће обрађивана тема: ″Системи за управљање базама података″. Тема је организована у осам делова.
У самом уводу упознаћемо се са појмом система за управљање базама података, његовим предностима у односу на класичну обраду података и основним карактеристикама. Компоненте система за управљање базама података приказане су у другом делу, како шематски, тако и кроз детаљна објашњења приказане слике. У трећем делу описана је архитектура система за управљање базама података, њени нивои, као и значај који она има у технологији база података. У четвртом делу биће речи о трансакцијама, њиховој конкурентној обради, као и о опоравку базе података. У петом делу су дати основни концепти сигурности база података и дискрециони механизам сигурности. О томе ко су корисници система за управљање базама података биће речи у шестом делу. Седми део ће се односити на системе база података нове генерације. Закључак садржи објашњење разлике између базе података и система за управљање базама података.

Кључне речи: база података, трансакција, сигурност базе података

Abstract

Below in the text will be processed subject: ″Database management systems ″. Topics is organized into eight parts.
You will introduce with the concept of the database management system, its advantages compared to traditional data processing and main characteristics. Components of database management system are presented in the second part, in a schematic, and detailed explanations of the displayed image. The third section describes the system architecture, database management, its level, and the importance it has in the technology database. The fourth section will examine the transactions, of their competitive process, as well as the recovery of the database. The fifth chapter presents the basic concepts of database security and discretionary security mechanism. About who are beneficiaries of the database management is discussed in the sixth section. Part VII shall apply to database systems of new generation. Conclusion contains an explanation of the differences between the database system and database management.

Keywords: database, transaction, database security

Садржај

1. Увод

База података се најопштије може дефинисати као добро структуирана колекција података која постоји релативно дуго и коју користи и одржава више корисника, односно програма (апликација). Изучавању база података може се приступити са два различита, међусобно повезана аспекта у којима се оне третирају било као:

- **системи за управљање базама података** (*Data Base Management Systems - DBMS*), или као
- **модели података,** односно специфичне теорије помоћу којих се специфицује нека конкретна база података или информациони систем, уопште.

Историјски, системи за управљање базама података представљали су револуцију у технологији обраде података, а модели података револуцију у методолошким приступима развоју информационих система.

Иако и један и други аспект имају изузетно велики значај за развој информационих технологија, у овом раду посебна пажња ће бити усмерена на један од њих – СИСТЕМЕ ЗА УПРАВЉАЊЕ БАЗАМА ПОДАТАКА.

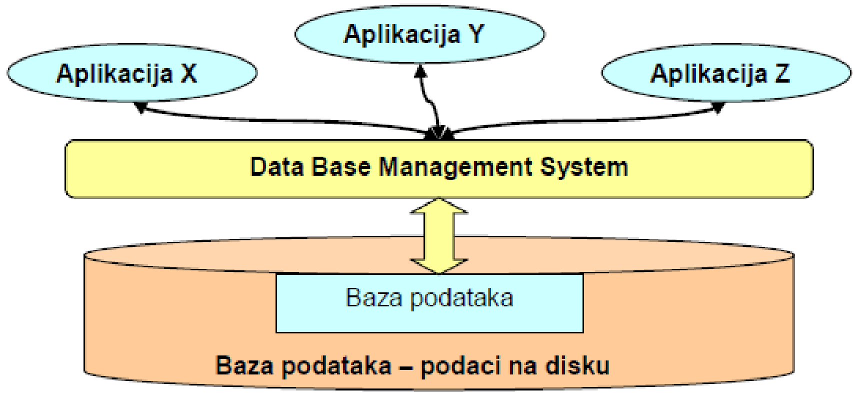
1.1 Појам система за управљање базама података

Систем за управљање базама података (СУБП) је софтверски систем који се користи за креирање, одржавање и манипулисање подацима, као и за контролу права приступа бази података.

СУБП омогућава крајњим корисницима и програмерима да деле податке, тј. омогућава да се подаци користе од стране више апликација, а не да свака апликација има своју копију податка сачувану у посебним датотекама.

СУБП такође пружа могућност контроле приступа подацима, осигурава интегритет података, успоставља контролу конкурентности и врши опоравак базе података. Програмери апликација за рад са базама података не морају да познају детаље о начину записа базе података на диску, не морају да формулишу алгоритме за ефикасан приступ подацима, нити су оптерећени било каквим аспектима око управљања подацима у бази података.

Термини ВР (база података) и управљање базом података се понекад мешају. Стручно говорећи, ВР је увек скуп чињеница, не рачунарски програм. СУБП је уведен као интерфејс између корисника (корисничких програма, апликација) и записа базе података на диску (слика 1.1). Кориснички програми не приступају подацима директно, већ комуницирају са овим софтвером (програмом).



Слика 1.1. СУБП је интерфејс између апликација (корисника) и записа базе података на диску [6]

СУБП управља структуром базе података: дефинише објекте базе, њихова својства (атрибуте), дозвољене вредности атрибута, везе између објеката, ограничења над објектима и међусобним везама.

Омогућава манипулацију подацима у бази: уношење, брисање и измене, тј. омогућава њено одржавање.

Контролише приступ подацима: ко може да приступи подацима, којим подацима и шта може са њима да ради.

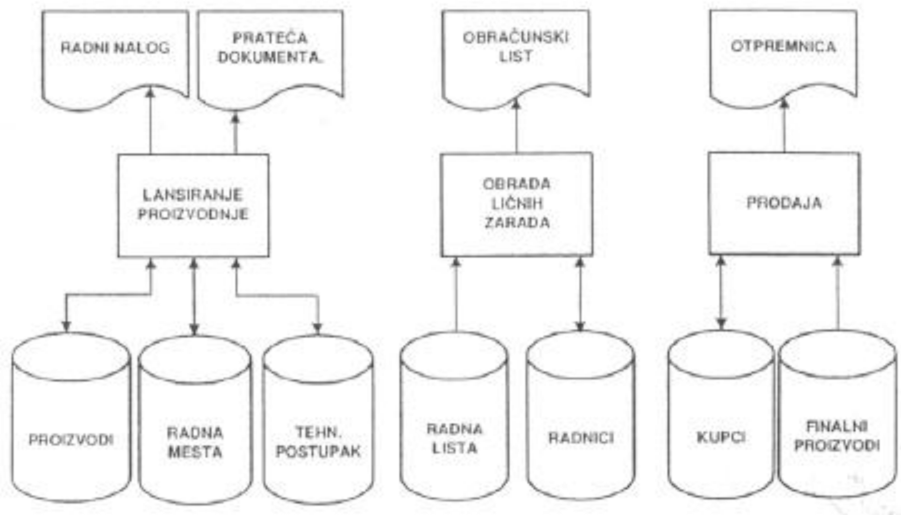
СУБП дозвољава дељење ВР између више апликација/корисника и чини управљање подацима успешнијим и делотворнијим.

Уобичајено је да када се говори о софтверу за базе података, онда се мисли управо на СУБП.

1.2 Предности SUBP у односу на класичну обраду података

Као софтверски систем СУБП, у развоју пословних информационих система, замењује “систем датотека“ (File System) и елиминише недостатке ове технологије која је примењивана у “конвенционалној “ обради података. Да бисмо уочили основне недостатке конвенционалне обраде података, у којој су се подаци чували и претраживали у скупу неповезаних датотека анализирајмо један пример овакве обраде.

На слици 1.2.1 приказан је системски дијаграм једног скупа апликација у неком информационом систему. Свака апликација је развијана посебно и користи своје посебне "приватне " датотеке са структуром података погодном за ту апликацију.



Слика 1.2.1 Класична обрада података – независне, "приватне" датотеке за сваку апликацију [1]

Основни недостаци овакве обраде података су:

- *Редуданса података, односно вишеструко памћење истих података је неминовно.* На пример, у неком предузећу, исти подаци о производима се памте и неколико десетина пута (у "матичној" датотеци, односно номенклатури производа, у датотеци технолошких поступака, у датотеци радних налога, датотеци отпремница итд.). Слично, у неком комуналном информационом систему, исти подаци о грађанима памте се и стотину пута. Вишеструко складиштење истих података доводи до непремостивих проблема при њиховом ажурирању. Када се неки податак промени, то се мора учинити на свим местима на којима се он чува. То, не само да значајно повећава трошкове обраде података, него је практично и неизводљиво. Због тога се у разним извештајима појављују различите вредности истог податка.

- *Зависност програма од организације података.* Програми су зависни и од логичке и од физичке структуре података. Под **физичком структуром (физичком организацијом) података** подразумева се начин меморисања података на спољним меморијама, док је **логичка структура**, структура података која је представљена програмеру. У класичним датотекама разлика физичке и логичке структуре података је мала. Другим речима, програмеру се практично директно приказује начин на који су подаци меморисани на спољној меморији. Зависност програма од логичке структуре се огледа у томе што програм зависи, на пример, од назива и редоследа поља у запису, што убацивање новог поља у запис или било какво друго реструктуирање записа, које не мења садржај података које програм користи, ипак захтева и измену самог програма. Физичка зависност се огледа у томе што програм зависи од дистрибуције датотека по чворовима рачунарске мреже, изабране физичке организације датотека и изабраних метода приступа, начина сортирања и слично. У основи и логичка и физичка организација података су прилагођене конкретном програму. Нови захтеви за информацијама, из података који већ постоје у датотекама, могли би захтевати измену физичке и логичке структуре података, а то би захтевало измене у раније развијеним програмима. Уместо тога, обично се за нови програм ствара нова датотека, а то даље повећава редундансу података.

- *Ниска продуктивност у развоју информационог система (IS).* Структуирање података у неком IS у скуп независних датотека, један је од узрока веома ниске продуктивности у развоју IS. Чак и када постоје сви подаци који се у некој новој апликацији захтевају, а налазе се у различитим датотекама, на разним медијумима, са различитим физичким организацијама, задовољење неког једноставног информационог захтева изискује значајне програмерске напоре. Над структуром података представљеном великим бројем независних датотека веома је тешко развити софтверске алате за брз и високо продуктиван развој апликација и за непосредну комуникацију корисника са системом (упитне језике, на пример).

- *Систем датотека је незадовољавајуће поуздан, не гарантује очување тачности и конзистентности (узајамни однос) података при могућим хардверским и софтверским отказима.*

- *Систем датотека не обезбеђује задовољавајуће очување тачности и конзистентности података при вишеструком паралелном коришћењу података.*

Ово су основни проблеми класичне пословне обраде података који су захтевали битне технолошке измене у овој области. Од касних шездесетих година до данас, област система за управљање базама података се изузетно брзо развија, да би данас достигла своју потпуну зрелост. Систем за управљање базама података (слика 1.2.2) је један сложени софтверски систем који треба да омогући:

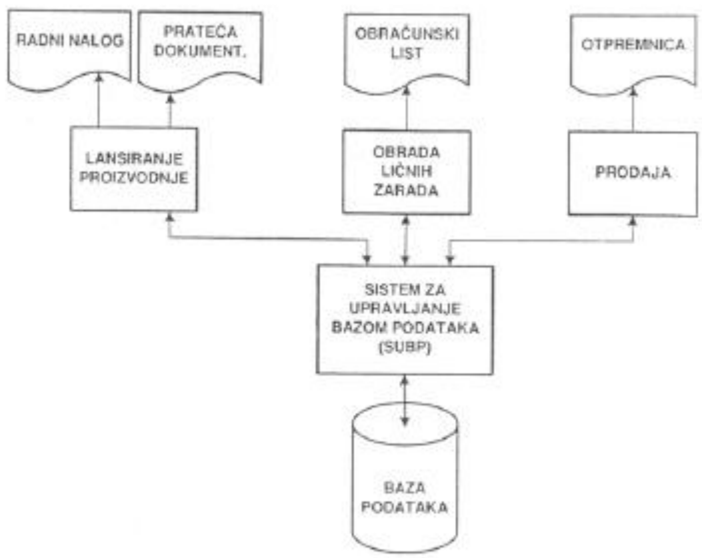
- Складиштење података са минимумом редундансе;

- Поузданост података и при могућим хардверским и софтверским отказима;

- Поуздано паралелно коришћење заједничких података од стране више овлашћених корисника;

- Логичку и физичку независност програма од података. Без обзира што се подаци физички памте, по правилу, само једном, у јединственој физичкој организацији, сваки корисник добија своју сопствену логичку слику података (логичку структуру података и/или колекцију операција над њима) каква њему највише одговара;

- Једноставно комуницирање са базом података преко језика блиских кориснику, тзв. "упитних језика", како би се непрофесионални корисници непосредно укључили у развој информационог система, а професионалним програмерима значајно повећала продуктивност.



Слика 1.2.2 Систем за управљање базама података [1]

У оваквој технологији обраде, подаци су, уместо разбацани по независним датотекама, организовани у јединствену базу података. Подаци постају јединствени ресурс у неком систему и њима се мора управљати на јединствен начин, онако како се управља и са другим виталним ресурсима пословних система.

1.3 Карактеристике система за управљање базама података

За разлику од класичне организације података, базиране на међусобно независним датотекама, систем за управљање базама података има следеће карактеристике:

- флексибилност – пошто су програми и подаци независни, могу се вршити измене података без измена у програмима,
- подаци су интегрисани у једној бази, што омогућава централизовано управљање и заједничко коришћење тих података, као и бржи приступ,

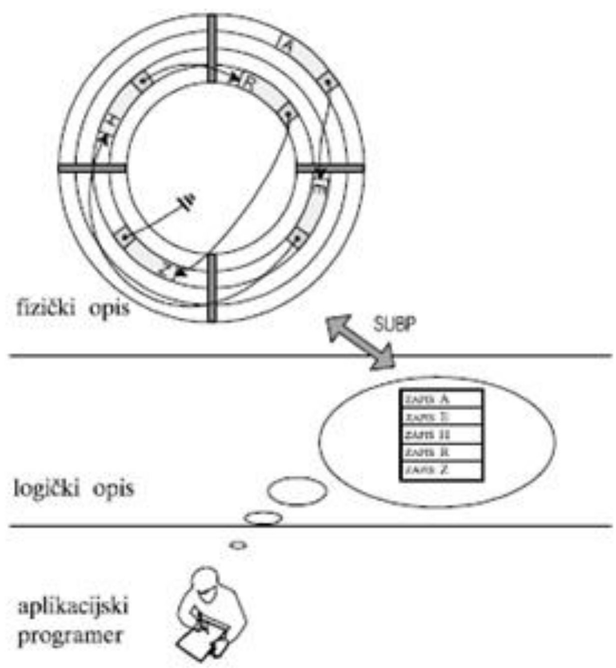
- подаци се у базу уносе само на једном месту, чиме се елиминише редудантност и смањује меморијски простор, а то такође доприноси ефикаснијем спровођењу промена,

- омогућено је остваривање “природних” веза међу подацима које проистичу из природе података и захтева који се постављају.

Важна последица примене SUBP јесте раздвајање физичке и логичке организације података (слика 1.4.1).

Док логичка организација података представља организацију са становишта корисника базе података или програмера па је концентрисана на врсте података и њихове међусобне логичке везе, физичка организација представља организацију физичког складиштења података унутар рачунара.

Облик и организација ускладиштених података су често потпуно различити од њиховог логичког облика и организације. У оквиру тога, задатак је SUBP -а омогућити кориснику (програмеру) манипулисање подацима уз познавање само логичког описа базе података, а не нужно и познавања начина физичког складиштења података.

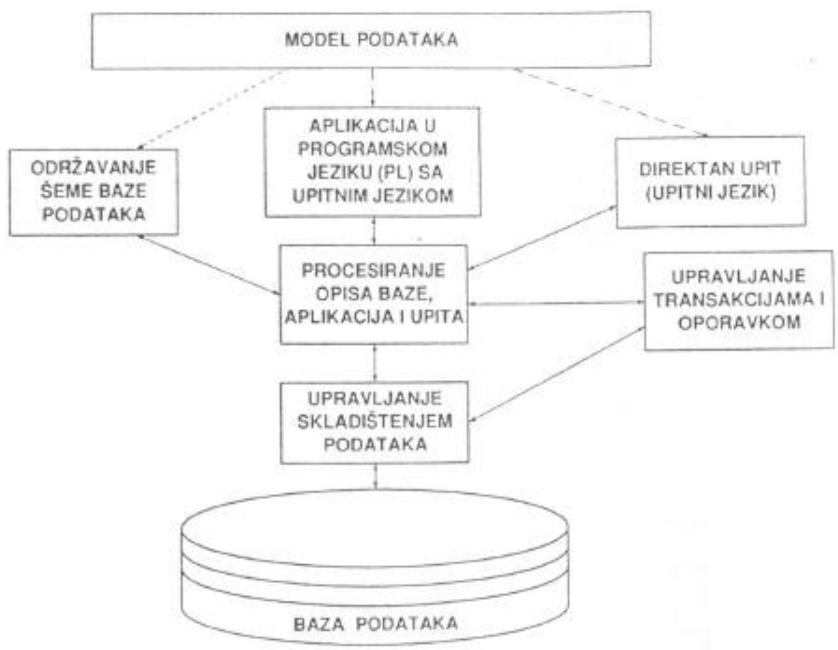


Слика 1.4.1. Физичка и логичка организација података [7]

2. Компоненте система за управљање базама података

На слици 2.1 дате су основне компоненте SUBP.

1. На дну слике је приказана **база података** преко ознаке за диск, што не значи да је база података смештена искључиво на дисковима, мада је то најчешћи случај. Велике базе података, поред дискова (секундарне меморије), захтевају и тзв. "терцијалну" меморију. Јединице терцијалне меморије имају капацитет реда терабајта (1000 гигабајта, односно 10^{12} бајта). На пример, систем компакт дискова са "роботом" за избор конкретног диска, представља један пример овакве меморије. Очигледно је да је приступ подацима на терцијалној меморији знатно спорији (неколико секунди) од приступа подацима на дисковима (10-20) милисекунди. Постоје и многе друге технолошке могућности чувања различитих формата података. Систем за управљање складиштењем података мора да обезбеди једнообразан приступ свим форматима података на свим врстама меморије.



Слика 2.1. Основне компоненте Система за управљање базом података (SUBP) [1]

База података, поред података, садржи и **метаподатке**, односно тзв. **"Речник података"** (Data Dictionary, Data Directory, Catalog). **Речник базе података** описује посматрану базу података (структуру базе, правила очувања интегритета података, права коришћења и слично). Другим речима, Речник података је "база података о бази података", па се тај део базе података назива и **метабаза** података. (Податак о податку се назива "метаподатак").
Термин **интегритет базе података** се користи да означи тачност (дозвољене вредности), односно конзистентност (дозвољене односе) података. Интегритет базе података може

да наруши нека грешка у програму или неки системски отказ. Правила интегритета дефинишу и акцију коју треба предузети када се наруши тачност или конзистентност неких података у бази.
Термин **права коришћења** везан је за концепт **сигурности базе података** и односи се на заштиту базе од неовлашћеног коришћења, модификовања, као и намерног оштећења или уништења података.
SUBP одржава и **базу индекса**. **Index**, најопштије, представља структуру података која омогућава брз приступ "индексираним" подацима базе. Најчешћа структура индекса је Б-стабло. Речник података садржи и опис индекса за посматрану базу.

2. **Систем за управљање складиштењем података** садржи две основне компоненте, **Управљање баферима** (Buffer Manager) и **Управљање датотекама** (File Manager). Управљање датотекама води рачуна о локацији датотека преко којих се реализује база података и о приступима блоковима података на захтев Управљања баферима. Дискови су уобичајено подељени у **блокове** који представљају континуалне сегменте меморије величине од 4.000 до 16.000 бајта. Управљање баферима прихвата блок података са диска, додељује му изабрану страницу централне меморије, задржава га извесно време, у складу са уграђеним алгоритмом управљања баферима, а затим враћа на диск ослобађајући додељену му страницу. Блокови са страница могу "форсирано" да се врате на диск и на захтев Система за управљање трансакцијама.

3. Као што је на Слици 2.1 приказано, улази у SUBP су:

- **Упити**, "ad hoc" специфицовани захтеви за подацима из базе, са наведеним условима које подаци треба да задовоље. Преко упита се, између осталог, може мењати садржај базе података;

- **Апликације** у неком програмском језику који се проширује са конструкцијама упитних језика, преко којих се, на унапред дефинисани начин претражује и мења садржај базе података;

- **Одржавање шеме базе података.** Шема базе података описује структуру базе података, правила интегритета и права коришћења. Одржавање шеме базе података подразумева прво креирање, а затим и модификовање овог описа који се чува у Речнику података.

Сва три наведена улаза се реализују преко тзв. **Језика базе података** (Database Language). Језик базе података заснива се на неком моделу података, а чине га две основне компоненте:

- **Језик за опис података** – ЈОП (Data Definition Language - DDL) који се користи за одржавање шеме базе података. Сличан је COBOL-у и може се користити као независан језик или као проширење програмског језика.

- **Језик за манипулацију података** – ЈМП (Data Manipulation Language - DML) преко кога се реализују упити и модификација базе података. Програмер помоћу овог језика врши пренос података између базе и апликативних програма, претраживање и ажурирање података, као и друге операције. Код већине система за управљање базама података DML представља део, односно додатни скуп инструкција основног програмског језика, најчешће COBOL- а и има компатибилну синтаксу.

Ови се језици могу употребљавати на различите начине:

1. Обављањем DML и DDL наредби уз помоћ интерактивних алата – корисник уз помоћ текст едитора задаје наредбу која се одмах обавља. Резултат обављања наредбе исписује се на екрану, записује у датотеку или исписује на штампачу.

2. Уграђивањем DML и DDL наредби унутар "језика домаћина" (*host language*). Наредбе се уграђују у код језика треће или четврте генерације, нпр. FORTRAN, COBOL, C, C++, Java, IBM Informix 4GL, итд. Пре превођења таквог програмског кода стандардним преводиоцем дотичног језика, изворни програмски код се предпроцесором трансформише тако да се уграђене DDL и DML наредбе трансформишу у позиве библиотечких функција које су посебно припремљене за појединачни језик домаћин.

3. Помоћу корисничких интерфејса која омогућавају употребу DML и DDL језика у графичком облику. Овај је начин погодан за кориснике који нису специјалисти у коришћењу DML и DDL језика, али најчешће се на тај начин знатно ограничава експресивност и могућност језика, па је за развој свих, осим тривијалних апликација, ипак нужно добро познавање DML и DDL језика.

Језици базе података су или потпуно непроцедурални (релационе базе) или у себи садрже процедуралне делове (хијерархијске, мрежне и објектне базе).

Непроцедурални језици садрже конструкције преко којих се само спецификују услови које треба да задовољи жељени резултат, а не и процедура помоћу које се добија тај резултат. Непроцедурални језици се често називају и **упитни језици** зато што им је основна намена спецификовање упита. Непроцедурални језици се могу користити и за модификацију и за опис шеме базе података. Због тога је основни задатак **Процесора упита (Query Processor)** да трансформише непроцедурални исказ у секвенцу акција које треба да обави Систем за управљање складиштењем података. Најзначајнији и најсложенији део ове трансформације је **оптимизација упита**, односно налажење најпогодније процедуре (секвенце акција) за реализацију непроцедуралног исказа. Наведена трансформација користи податке из Речника података: опис структуре, правила интегритета, права приступа и дефиницију индекса.

Апликација, програм који реализује унапред прецизно дефинисани захтев корисника, изграђује се коришћењем стандардних програмских језика у које се уграђује језик базе података. Преко језика базе података се дефинишу типови података у бази и обезбеђује приступ бази података, очитавање или мењање одговарајућих података, а стандардни

програмски језици служе за реализацију сложенијих алгоритама у посматраним апликацијама.
Апликације се могу дефинисати и преко тзв. "генератора апликација" . Генератори апликација генеришу апликације на основу дефинисаног корисничког интерфејса и сличности његове структуре и структуре података у бази. Генератори дају оквир апликације са спецификованим процедурама које представљају одзив система на одговарајуће акције корисника. Програмер треба да развије код ових процедура користећи, обично, тзв. "Језике IV генерације", специфичну комбинацију упитног и процедуралног језика. Сваки систем за управљање базом података придружује себи, поред других развојних алата и неки генератор апликација.

4. **Управљање трансакцијама и опоравком** треба да обезбеди да база података остане у конзистентном стању у **конкурентној обради** података, чак и при веома озбиљним отказима система. Пошто се један податак у бази података, по правилу, памти само једном, више програма истовремено (конкурентно) захтева приступ истим подацима. SUBP мора обезбедити да се конфликтни захтеви и нежељене интерференције програма, до којих у овој ситуацији може да дође, успешно разреше. SUBP такође треба да, преко периодичног копирања базе података на "архивске меморије" и регистровања промена које су се између два копирања догодиле, омогући ефикасан **опоравак базе** података после неког отказа.
У бази података се, преко програма, региструју резултати неких послова који се обављају у реалном систему. Очигледно је да се може дефинисати појам **логичке јединице посла** као основног, "атомског" (најмањег дела сложеног) посла који има неко реално значење. Са тачке гледишта корисника стање базе података се мења само по окончању логичке јединице посла. Међутим, једна логичка јединица посла може, у неком програму, да буде представљена са више операција над базом података. Свака операција над базом може да мења њено стање, а од значаја је само укупна промена коју изазива логичка јединица посла.
Трансакција је низ операција над базом података која одговара једној логичкој јединици посла у реалном систему. Трансакција може бити део програма или цео програм може да представља једну трансакцију.

3. Архитектуре Система за управљање базама података

У историји развоја, садашњим комерцијалним СУБП, као и у трендовима развоја, могу се јасно разликовати неколико различитих архитектура СУБП, преко којих се на различит начин остварују основни циљеви технологије база података: (1) нередундантно памћење података, (2) независност програма од логичке и физичке структуре базе података и (3) пројектовање и одржавање базе података као заједничког ресурса целог IS, независно од скупа његових апликација.

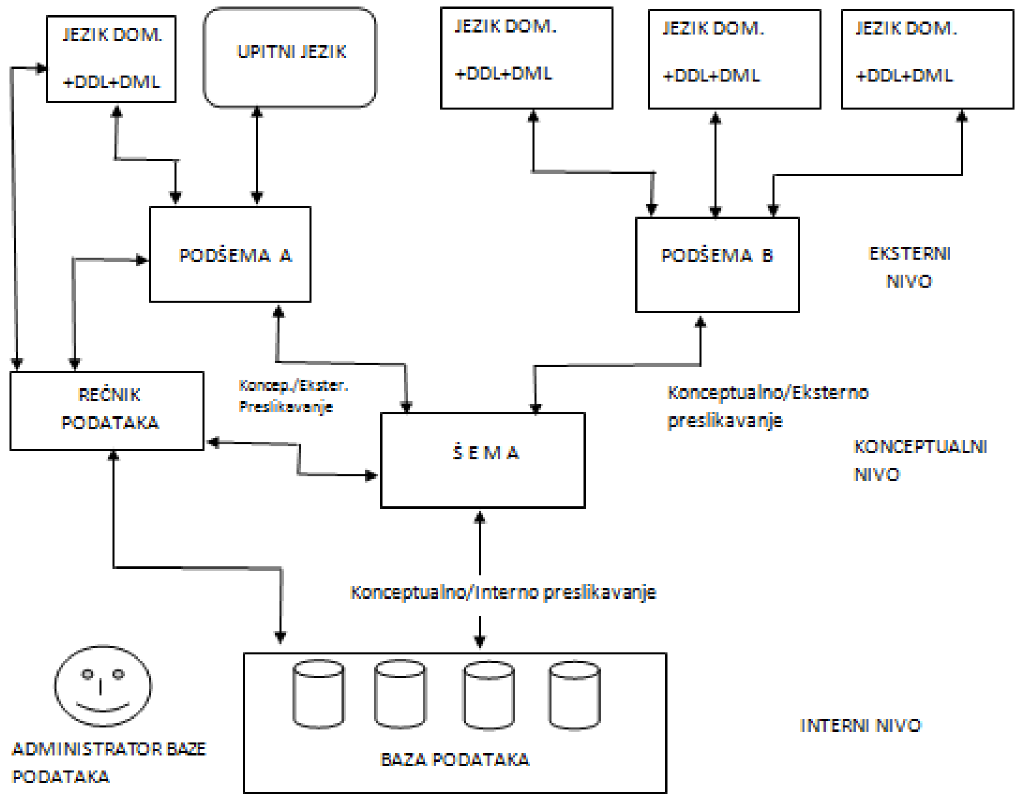
ANSI/SPARC архитектура (CODASYL/DBTG) (Слика 3.1) је тронивоска архитектура у којој поједини нивои имају за циљ да, с једне стране, учине независном логичку од физичке структуре базе података, а са друге, апликационе програме и од физичке и од целокупне логичке структуре базе. Нивои у овој архитектури су:

- *Интерни (физички) ниво*, који дефинише начин на који су подаци физички организовани на спољним меморијама;
- *Концептуални ниво (шема базе података)*, који дефинише општу логичку структуру базе података, све податке у систему, њихове логичке односе (везе) и који треба да омогући управљање подацима као заједничким ресурсом у целом систему;
- *Екстерни (кориснички) ниво (подшема)*, на коме се дефинише логичка структура података погодна за специфичне захтеве, односно програме.

На интерном нивоу, преко језика који се назива интерни језик за дефиницију података (Internal Data Definition Language - IDDL), специфицира се физичка организација података. Дефинишу се физички записи (блокови), начин на који се поједини подаци меморишу, методе приступа, индекси, и слично.

На концептуалном нивоу описује се општа логичка структура базе података као и ″пресликавање″ ове структуре у интерни ниво.

На екстерном нивоу дефинишу се подмодели базе података погодни за развој појединачних апликација. Програмер развија своју апликацију над логичком структуром која представља његов ″поглед″ на целокупну базу података.



Слика 3.1. Архитектура система за управљање базама података [1]

Очигледно је да приказана тронивоска архитектура дефинише и два скупа обостраних трансформација (пресликавања): концептуално/интерно и концептуално/екстерно пресликавање. Преко ових пресликавања се реализује независност логичке од физичке структуре базе података, као и независност апликационих програма од целокупне логичке структуре базе. Ако се, на пример, промени физичка структура базе података, није неопходно мењати општу логичку структуру (шему), већ само пресликавање шеме у интерни ниво. Исто тако, ако се промени структура шеме, подшеме могу остати непромењене, ако се промени одговарајуће пресликавање.

Корисник може да комуницира са базом података преко програма развијених у неком програмском језику (″језику домаћину″) или директно, преко ″упитних језика″ (било преко општег интерфејса за постављање упита или преко специфичног корисничког интерфејса за кога је везан скуп параметризованих упита, а интерфејс служи да се прихвате конкретне вредности параметара).

″Језик домаћин″ (Host Language) је неки конвенционални или објектно-орјентисанипрограмски језик (COBOL, C, C++, Java, Visual Basic). Програмер развија своју апликацију користећи ″језик домаћин″ и Језик базе података. За конвенционалне језике за дати СУБП постоје **предпроцесори** за појединачне језике који, с једне стране, дефинишу начин на који се наредбе Језика базе података уграђују у ″језик домаћин″, а са друге, предпроцесирају програм трансформишући ове наредбе у посебне рутине и одговарајуће ″позиве″ тих рутина. На тај начин се, у даљој обради , могу да користе стандардни преводиоци за одговарајући језик.

Администратор базе података. Као што је раније речено, база података представља један посебан самостални ресурс у неком информационом систему.
Она се развија и њеним коришћењем се управља независно од било које апликације која са њом ради. Због тога је потребно дефинисати радна места, односно одговарајућа занимања, која ће управљати јединственом базом података у некој организацији. То радно место, односно то занимање се назива администратор базе података. Администратор базе има кључну улогу у информационим системима заснованим на базама података. Они управљају радом целокупног система обављајући следеће функције:

- Развој концептуалног нивоа, односно управљање подацима у систему.
- Избор физичке организације базе података и метода приступа.
- Дефиниција и реализација пресликавања концептуални – интерни и концептуални – екстерни ниво, односно дефинисање или помоћ у дефинисању подшема.
- Обезбеђење сигурности и интегритета базе података.
- Дефинисање стратегије опоравка базе података после могућих оштећења.
- Праћење перформанси система.

У почетку развоја СУБП су били веома скупи софтверски системи који су се извршавали на великим рачунарима, који су једини могли да подрже складиштење неколико гигабајта података и њихову ефикасну обраду. Данас неколико гигабајта стаје на један релативно мали диск персоналног рачунара, који је довољно моћан да ту количину података ефикасно обради. Због тога су данас СУБП део стандардног софтвера персоналних рачунара заједно са процесорима текста и система за обраду табела (speadsheet). Очигледно је да у том случају сам корисник може да игра улогу администратора базе података. Међутим, исто тако, данас неколико гигабајта није посебно велика количина података. У сложеним информационим системима у којима се чува историја промене података, звучни, графички и видео записи, захтеви за складиштење података су знатно већи, и до неколико терабајта. Због тога, а и због чињенице да већи број корисника треба да користе исту базу, и даље постоји потреба за великим заједничким базама података које се формирају и одржавају као самостални ресурс.

ANSI/SPARC архитектура у потпуности је примењена у Мрежним (CODASYL) системима за управљање базом података, где се концепти шеме и подшема експлицитно користе. Хијерархијски и релациони СУБП готово у потпуности прате ову архитектуру. Релациону базу података представља скуп ″равних″ табела, међусобно повезаних преко вредности података у њима. ″Равна″ табела је табела без икакве унутрашње структуре, са једноставним врстама и колонама. Концепт ***погледа*** (view) у релационим СУБП је аналоган концепту подшеме. Поглед је виртуелна табела изведена из базних табела преко неког SQL упита. Поглед физички не постоји у бази, већ се изводи динамички (када се над њим извршава упит). При томе треба нагласити да само специфично дефинисани погледи могу да послуже да се преко њих модификује база података, па је њихова улога у креирању подшеме релативно мала.

Објектни системи за управљање базама података имају другачију архитектуру. Већ у дефиницији објекта, основног концепта у објектним СУБП, јасно се одваја спецификација и имплементација објекта. ***Интерфејс типа објекта*** даје све видљиве особине (атрибуте и везе) и операције објекта. Користећи аналогију са ANSI/SPARC архитектуром, мада се у објектним СУБП одговарајући термини не користе, може се рећи да интерфејси свих типова објеката из базе чине концептуални модел (шему) базе података. Базу података је могуће посматрати као јединствену софтверску компоненту која поседује скуп интерфејса у којима су логички груписане особине и операције појединих типова објеката. Један или више оваквих интерфејса се може третирати као подшема базе, а унија свих њих као шема базе података. Треба нагласити да објектни СУБП поседују и упитни језик OQL, у коме постоји механизам еквивалентан механизму погледа у релационим базама. Због тога се може рећи да објектни СУБП могу да остваре тронивоску архитектуру и коришћењем овог механизма.

4. Управљање трансакцијама

База података је заједнички ресурс који истовремено (конкурентно) користи већи број програма. При оваквом коришћењу базе података може доћи до многих нежељених ефеката као што су, на пример:

- Отказ система у току извршења неког програма који може да остави базу података у неконзистентном стању,
- Нежељена интерференција два или више програма над подацима за које истовремено конкуришу, може, такође, да доведе базу података у неконзистентно стање.

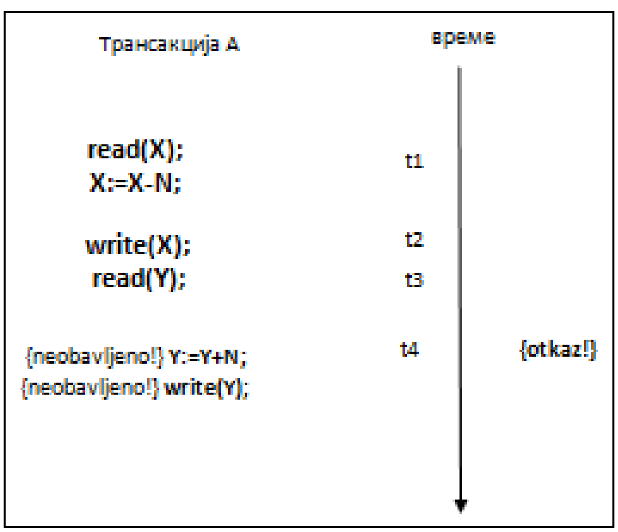
Основни циљ базе података је да омогући ефикасну обраду трансакција. ***Трансакција*** је једно извршење неке ″логичке јединице посла″, једно извршење неке логичке целине једног програма, или једно извршење целог програма. У једном тренутку времена над базом података се извршава више трансакција. Често се и више извршења једног истог програма (свако од њих представља трансакцију) обавља конкурентно. На пример, извршење програма за подизање и улагање новца на рачун у некој банци, може се покренути готово истовремено са више различитих шалтера.

Проблеме до којих може да доведу отказ система и неконтролисано конкурентно извршење трансакција илустроваћемо са неколико примера.

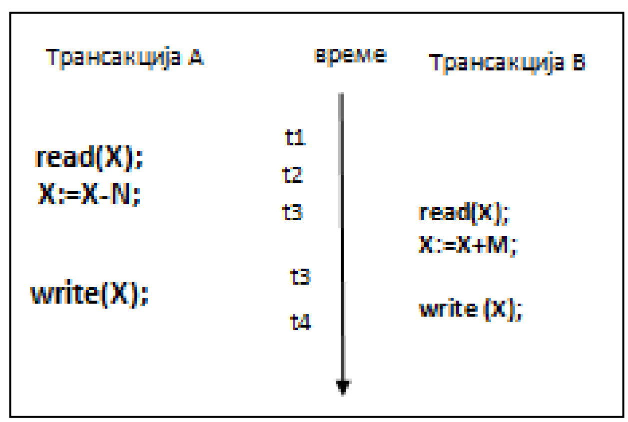
Отказ система у току обраде трансакције. На Слици 4.1 приказано је одвијање трансакције за пренос новца (N динара) комитента банке са рачуна X на рачун Y. Ако у тренутку t4 дође до отказа система трансакција се неће у потпуности обавити и доћи ће до нарушавања интегритета базе података – новац ће се скинути са рачуна X, али се неће пренети на рачун Y.

Губљење резултата ажурирања. На Слици 4.2 приказано је извршење две трансакције у времену. Трансакција А подиже, а трансакција В улаже новац на исти рачун. Очигледно је да је ажурирање које Трансакција А обавља у тренутку t3 изгубљено, односно ″пребрисано″ ажурирањем које је извршила трансакција В у тренутку t4.

Проблем некоректне анализе података. Овим називом се дефинише скуп проблема до којих долази када, за време неког ″срачунавања″ које се обавља у једној трансакцији, друга промени вредност аргумент који је прва већ обрадила. Претпоставимо, на пример, да једна трансакција срачунава неку суму користећи као сабирке неке податке из базе, док друга ажурира вредности истог скупа података. Очигледно је да је могуће да се у суму узме нека вредност која је у међувремену промењена, односно да се добије некоректна вредност суме.



Слика 4.1. Нарушавање интегритета базе због отказа система [1]



Слика 4.2. Губљење резултата ажурирања [1]

Постоје и други проблеми до којих долази при отказу система и конкурентној обради трансакција. Неки од њих ће бити приказани касније.

4.1 Трансакције

Као што је већ речено, ***трансакција*** је једно извршење неке ″логичке јединице посла″ (Logical Unit of Work – LUW) корисника базе података. Та ″логичка јединица посла″ може да представља неку логичку целину једног програма или целокупан програм.

Трансакција мора да поседује следећи скуп особина:

1. **Атомност** (Atomicity). Захтева се да се било све операције над базом података успешно обаве или ниједна. Очигледно је да трансакција, ″лигичка јединица посла″, мора представљати атомски скуп активности. Да би атомност трансакције имала смисла, СУБП мора да гарантује и атомност сваке активности (појединачне операције) над базом података. То је посебно битно за релационе СУБП где су операнди у операцијама скупови. Када се врши ажурирање скупа и када се та атомска операција прекине, неопходно је да буду поништене све промене које је она до тада изазвала. Да би се остварила атомност трансакције дефинишу се две специфичне операције над базом података:

- COMMIT која означава успешан крај трансакције и која ″потврђује″ све промене у бази које је посматрана трансакција произвела;
- ROLLBACK којом се поништавају ефекти свих претходних операција над базом података у једној трансакцији, ако она, због предвиђене или непредвиђене грешке (отказа система) може да доведе базу података у неконзистентно стање.

Ове две операције се могу извршавати из корисничког програма или их може извршити СУБП, управљајући конкурентним извршењем скупа трансакција, интегритетом базе података или опоравком базе података.

2. **Конзистентност** (Consistency). Очигледно је да се трансакција може дефинисати и као ″јединица конзистентности″ базе података: пре почетка и после окончања трансакције стање базе података мора да задовољи услове конзистентности. За време обављања трансакције конзистентност базе података може да буде нарушена.

3. **Изолација** (Isolation). Када се две или више трансакција извршавају истовремено, њихови ефекти морају бити међусобно изоловани. Другим речима ефекти које изазову трансакције које се обављају истовремено морају бити једнаки ефектима неког њиховог серијског (једна после друге) извршења.

4. **Трајност** (Durability). Када се трансакција заврши њени ефекти не могу бити изгубљени, чак и ако се непосредно по њеном окончању деси неки озбиљан отказ система.

Наведене особине се уобичајено називају ″ACID особине трансакције″, по првим словима њихових енглеских назива. Да би се оне обезбедиле, скуп инструкција које представља трансакцију почиње, по правилу, са специјалном инструкцијом BEGIN TRANSACTION, а завршава се било са инструкцијом COMMIT (потврђују се промене у бази података, ако су све инструкције трансакције успешно извршене) или инструкцијом ROLLBACK (поништавају се промене у бази, ако све инструкције нису успешно завршене). Инструкција ROLLBACK може да буде и имплицитна, иницира је СУБП када, из било ког разлога, дође до неког непланираног завршетка трансакције.

Један програм може представљати колекцију трансакција, мада у пракси, често, један програм представља једну трансакцију. У савременим СУБП трансакције могу бити ″угњеждене″, односно дозвољено је да се једна трансакција смести у другу. У том случају, наредба COMMIT угњеждене трансакције стварно не потврђује промене у бази података док се успешно не оконча и трансакција на највишем нивоу угњеждења (спољна трансакција). Ако се у било којој трансакцији угњеждене структуре покрене инструкција ROLLBACK, поништавају се промене које су произвеле све остале трансакције структуре.
На пример, програм за пренос новца са рачуна на рачун, може се, као трансакција, приказати кодом датим на Слици 4.3.
У извршењу трансакција није само проблем очување интегритета базе података. Могуће је да дође до тога да се кориснику пошаљу некоректне (неконзистентне) поруке. На пример, ако се у програму приказаном на Слици 4.3, пре потврде промена у бази података, изда порука да је трансакција успешно обављена, могуће је, да због неке грешке трансакција буде поништена, па је корисник добио некоректну поруку. Због тога трансакција мора да има особину атомности и у односу на тзв. ″стварне излазе″. То се постиже на тај начин што се излазне поруке не преносе директно, већ тек након планираног завршетка трансакције (са COMMIT или ROLLBACK). Поступак управљања порукама је следећи: Поруке се не прихватају директно већ се улазне поруке стављају у тзв. улазни ред, а излазне поруке у излазни ред. Операција GET узима улазну поруку из улазног реда, а операција PUT ставља излазну поруку у излазни ред. При планираном завршетку трансакције (COMMIT или експлицитни ROLLBACK) извршава се:

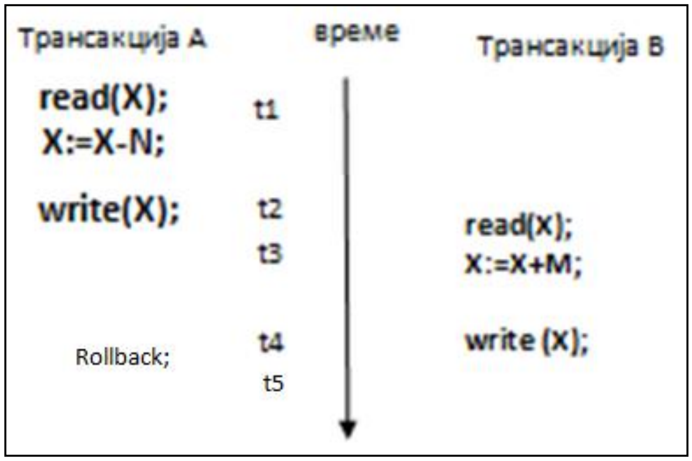
a) уписују се на лог излазне поруке,
b) стварно се преносе излазне поруке и
c) избацују се улазне поруке из улазног реда.

При непланираном ROLLBACK (који се извршава аутоматски ако дође до неке грешке у програму типа ″overflow″, дељење са нулом и слично), проузрокује избацивање порука из улазног и излазног реда.

```
WHENEVER SQLERROR GO TO UNDO; {Bilo koja greška u obraćanju bazi podataka}
      GET LIST (r1,r2, N);  {preuzimaju se vrednosti koje daje korisnik}
      SELECT STANJE INTO X
      WHERE BROJRAC = r1;
      X := X – N;
      IF X < 0
        THEN DO;
                PUT ('Nema dovoljno novca');  {izlazna poruka}
                ROLLBACK;
                END;
        ELSE DO
                UPDATE RACUN
                SET STANJE TO X
               WHERE BROJRAC = r1;  { Skinuta je vrednost N sa računa r1}
                 UPDATE RACUN
                 SET STANJE TO STANJE + N
                 WHERE BROJRAC = r2;  { Dodata je vrednost N račun r2}
                 PUT ( 'Transakcija uspešno obavljena); {Poruka na izlazu}
                 COMMIT;  {Potvrđene su sve promene u bazi podataka}
                 END;
        GO TO KRAJ;
        UNDO: EXEC SQL ROLLBACK;  {Poništene sve promene}
        END EXEC;
        KRAJ: RETURN;
        END;
```

Слика 4.3. Однос трансакције и стварних излаза [1]

Проблем непотврђених промена (читања). Овај проблем се јавља када се једној трансакцији дозволи да чита или, још горе, да мења рекорд који друга трансакција ажурира, а промене које је она учинила још нису потврђене (наредбом commit). Пример овог проблема приказан је на Слици 4.4. Трансакција B је прочитала, у тренутку t3, вредност X коју је у тренутку t2 трансакција A ажурирала, али то ажурирање још није потврђено. Пошто је у тренутку t5 трансакција A поништила своју промену податка X, трансакција B је прочитала некоректну (непостојећу) вредност X (непотврђено или ″прљаво″ читање), а затим и извршила некоректно ажурирање.



Слика 4.4. Проблем непотврђених промена [1]

На крају овог дела неопходно је објаснити и начин на који се имплементирају COMMIT и ROLLBACK, посебно. Да би се могле поништити промене које је трансакција извршила над базом података СУБП поседује и одржава специјалну меморијску јединицу (на некој спољној меморији) која се назива **лог** или **журнал**. На овој меморијској јединици, за сваку трансакцију и сваки објекат базе који је она ажурирала чувају се вредности **пре** (before-image) и вредности **после** ажурирања (after-image). Када се изда инструкција ROLLBACK систем, користећи вредности **пре** за дату трансакцију, ажурира одговарајуће објекте базе података.

Могуће је, такође, да после издавања наредбе COMMIT, односно за време њеног извршавања, дође до отказа система. Тзв. **write-ahead лог** протокол решава овај проблем. Наиме, при сваком ажурирању базе података прво се упише запис на **лог**, са **вредностима пре и после** за одговарајући објекат и трансакцију, а тек се потом ажурира сама база. Ако у извршењу наредбе COMMIT дође до отказа, систем користи **вредности после** са лога да би довео базу у конзистентно стање.

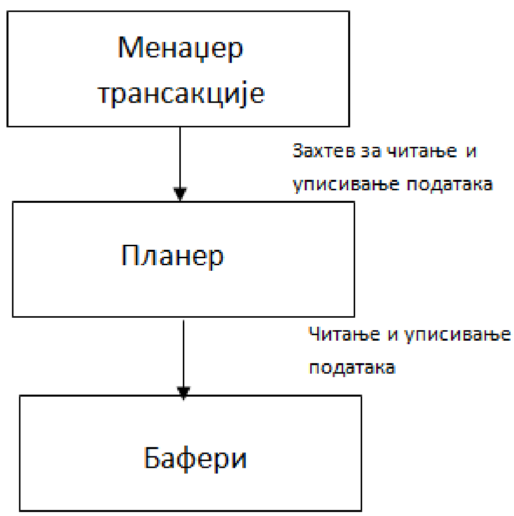
4.2 Конкурентност

Данас је веома битан и значајан концепт базе података по коме је то, у ствари, заједнички ресурс кога истовремено (конкурентно) користи већи број програма, јер се прави ефекти базе података испољавају када се ради у мрежном окружењу. Посматрајмо базу података једне банке у којој се налазе рачуни грађана. Могуће је да се у истом тренутку на шалтеру у једној експозитури подиже новац са једног рачуна и уплаћује на други рачун, а да се истовремено у сасвим другој експозитури уплаћује новац на исти тај рачун. Поменути СУБП је управо ту да управља конкурентним радом више корисника и да обезбеђује синхронизацију њиховог рада.

Трансакција се у системима заснованим на бази података не обавља у изолацији, већ конкурентно (упоредно) са другим трансакцијама у систему.
Више трансакција могу истовремено захтевати исте ресурсе, исти рекорд базе података. Неконтролисана међусобна интерференција трансакција може да доведе базу у

неконзистентно стање. Као што је на Слици 4.5. приказано, компонента СУБП-а која води рачуна о редоследу извршавања акција над базом података у скупу трансакција које се конкурентно извршавају, назива се ***Planer (Scheduler)***. Компонента која управља целокупним извршењем трансакција назива се ***Менаџер трансакције (Transaction manager).*** На Слици 4.5. приказана је и размена захтева и података између ових компоненти и бафера.

Трансакција захтева читање или уписивање елемента базе података и овај захтев се преноси Планеру. У већини случајева Планер ће овај захтев извршити директно: ако је елеменат у баферу акција се непосредно обавља, а ако није, преко система за управљање баферима, елеменат се преузима из базе података. Међутим, ако читање или уписивање може да наруши интегритет базе података, Планер може да га одложи, или чак и да поништи трансакцију која је тај захтев издала. Овде ће се дискутовати методе ″планирања″ извршења скупа трансакција које служе Планеру да донесе одлуку о реализовању захтева.



Слика 4.5. Планер извршења скупа трансакција [1]

За разлику од конкурентног, серијско извршење трансакција је извршење у коме се трансакције не преплићу, већ се прво потпуно изврши једна, па онда друга трансакција (A па B, или B па A, за пример на сликама 4.2. и 4.4). Резултат серијског извршења скупа трансакција у било ком редоследу ћемо сматрати коректним. Користећи овај став, можемо да уведемо концепт серијабилности или линеарности извршења скупа трансакција. Упоредно извршавање скупа трансакција је **серијабилно (линеарно)** ако производи исти резултат као и неко серијско извршење истог скупа трансакција. Усваја се да је скуп упоредних трансакција извршен коректно, ако и само ако је тај скуп серијабилан.

Оваква дефиниција серијабилности узима у обзир и детаље (семантику) трансакција. На пример, ако би у примерима приказаним на сликама 4.2 и 4.4 било N=0, извршење трансакција A и B би било серијабилно. Очигледно је да Планер не може да узме у обзир семантику трансакција, већ само редослед операција читања и уписивања података у базу. Због тога ће се извршење скупа трансакција представљати само преко секвенци наредби за читање и уписивање елемента базе података. На пример, извршење трансакција A и B са Слике 4.2. се представља на следећи начин:

$$r_A(X),\ r_B(X),\ w_A(X),\ r_B(X),$$

Може се претпоставити да се трансакције идентификују редним бројевима (Односно називају $T_1, T_2, \ldots T_i, \ldots$). У том случају $r_i(X)$, и $w_i(X)$ значи читање или уписивање елемента X базе података које врши трансакција идентификована са i.

4.2.1. Серијабилност

Различити протоколи серијабилности извршења скупа трансакција заснивају се на различитим начинима утврђивања серијабилности и различитим поступцима остваривања серијабилности.

За дефинисање тзв. view-еквиваленције два извршења скупа трансакција и view-серијабилности датог извршења скупа трансакција користе се следећи појмови:

- Ако у неком извршењу скупа трансакција неко уписивање $w_j(X)$ претходи неком читању истог елемента базе $r_i(X)$ и нема ниједне друге операције уписивања $w_k(X)$ између њих, каже се да ″ $r_i(X)$ чита из $w_j(X)$ ″, односно да између $r_i(X)$ и $w_j(X)$ постоји ″ чита из ″ однос.

- Операција $w_i(X)$ се зове ″ крајње уписивање″, ако је то последње уписивање елемента X у базу.

″View″ еквиваленција и ″view″ серијабилност се дефинишу на следећи начин:

- Два извршења скупа трансакција су **view-еквивалентна** ($S_i \approx_v S_j$) ако поседују исте ″ чита из ″ односе и иста ″ крајња уписивања ″.

- Извршење скупа трансакција је **view-серијабилно** ако је view-еквивалентно са неким серијским извршењем.

Очигледно је да пример извршења скупа трансакција S_1 (губитак резултата ажурирања) није view-серијабилно

$$S_1: \ r_1(X),\ r_2(X),\ w_1(X),\ w_2(X)$$

Исто важи и за извршење S_2 (неконзистентно читање) и S_3 (фантомско ажурирање) такође нису view-серијабилни.

$$S_2: \ r_1(X),\ r_2(X),\ w_2(X),\ r_1(X)$$

$$S_3: \ r_1(X),\ r_1(Y)\ r_2(Z),\ r_2(Y)\ w_2(Y),\ w_2(Z),\ r_1(Z)$$

Извршење S_4 је view-серијабилно зато што је view-еквивалентно серијском извршењу S_5.

$$S_4: \ w_0(X),\ r_2(X),\ r_1(X),\ w_2(X),\ w_2(Z)$$

$$S_5: \ w_0(X),\ r_1(X),\ r_2(X),\ w_2(X),\ w_2(Z)$$

Алгоритам утврђивања view-еквивалентности два извршења скупа трансакција, који треба да чита два низа операција читања и уписивања и да утврђује да ли је задовољен услов, има полиномијалну сложеност. Међутим, алгоритам за утврђивање view-серијабилности захтева налажење свих могућих серијских извршења скупа трансакција и због тога је неефикасан NP-комплетан проблем. Због тога се, мада најопштије дефинисана серијабилност, view-серијабилност практично не користи.

<u>Конфликт – серијабилност</u>

Један од практичнијих приступа утврђивању серијабилности извршења скупа трансакција се остварује преко дефинисања концепта конфликт-серијабилности. Под конфликтом се овде подразумева ситуација у којој измена редоследа две операције у извршењу доводи до измене ефеката на базу барем једне од трансакција из посматраног извршења.

Ако претпоставимо да трансакције Ti и Tj припадају неком извршењу, тада следећи парови операција неће бити у конфликту:

1. $r_i(X)$, $r_j(Y)$ није конфликт чак и када је $X = Y$ јер ни једна ни друга операција не мењају стање базе података.

2. $r_i(X)$, $w_j(Y)$ очигледно није конфликт, под претпоставком да је $X \neq Y$.

3. $w_i(X)$, $r_j(Y)$ није конфликт, под претпоставком да је $X \neq Y$.

4. $w_i(X)$, $w_j(Y)$ није конфликт.

Планер не може да измени редослед операција у оквиру једне трансакције, јер се тиме мења семантика трансакције. Најопштије, две суседне операције различитих трансакција могу заменити места ако није задовољен један од услова:

1. Операције се обављају над истим елементом базе података и
2. Барем једна од њих је уписивање.

Два извршења скупа трансакција су **конфликт-еквивалентна** ако се један у други могу трансформисати неконфликтним изменама места суседних операција. Извршење скупа трансакција је **конфликт-серијабилно** ако је конфликт-еквивалентно са неким серијским извршењем.

Постоји релативно једноставан поступак за утврђивање конфликт-серијабилности извршења скупа трансакција. Ако у некој трансакцији постоји пар операција који производи конфликт њихов редослед у сваком конфликт-еквивалентном серијском извршењу мора да остане исти. Ако овај захтев не доводи до контрадикције, могуће је наћи конфликт-еквивалентно серијско извршење. У противном је то немогуће. За утврђивање да ли долази до контрадикције користи се **граф претходења трансакција**. У овом графу трансакције једног извршења су чворови, а усмерене гране показују претходење трансакција. Каже се да трансакција T_i претходи трансакцији T_j ($T_i <_S T_j$) у извршењу S ако постоји операцији O_i трансакције T_i и операција O_j трансакције T_j тако да је:

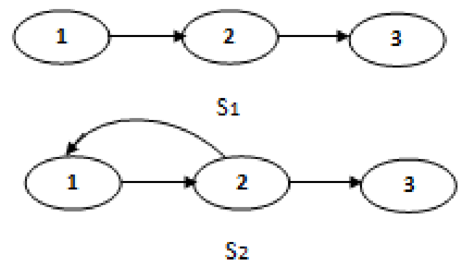
1. O_i претходи O_j у S,
2. И O_i и O_j се односе на исти елеменат базе података,
3. Барем једна од операција O_i и O_j је операција уписивања.

Посматрајмо извршења S_1 и S_2 са трансакција T_1, T_2 и T_3 који се разликују само по положају операције $r_2(B)$:

S_1: $r_2(A)$, $r_1(B)$, $w_2(A)$, $r_3(A)$, $w_1(A)$, $w_3(A)$, $r_2(B)$, $w_2(\mathbf{B})$

S_2: $r_2(A)$, $r_1(B)$, $w_2(A)$, $r_2(B)$, $r_3(A)$, $w_1(A)$, $w_3(A)$, $w_2(\mathbf{B})$

Ако посматрамо операције над елементом A у S_1 да $T_2 <_{S1} T_3$. Ако посматрамо исто извршење у односу на елеменат **B** очигледно је да је $T_1 <_{S1} T_2$. До истог закључка се може доћи и за извршење S_2. $T_1 <_{S2} T_2$ и даље важи због тога што се $r_1(B)$ и $w_1(\mathbf{B})$ појављују пре $w_2(\mathbf{B})$. Међутим, постоји и претходење $T_2 <_{S2} T_1$ због тога што се $r_2(B)$ појављује пре $w_1(\mathbf{B})$. Графови претходења за ова извршења су дата на Слици 4.6.



Слика 4.6. Графови претхођења за конфликт-серијабилност [1]

Очигледно је да циклус у графу указује на контрадикцију у конфликт-односима парова операција, па такво извршење није могуће учинити серијабилним. Ациклички граф претхођења трансакција у дефинисаном смислу, показује да је извршење конфликт-серијабилно. Алгоритам за налажење циклуса у графу је полиномијалне сложености, па је прихватљив као метод за утврђивање серијабилности извршења скупа трансакција. Због тога конфликт-серијабилност користе многи комерцијални системи за управљање базама података.

Протоколи закључавања

Велика је вероватноћа да ничим ограничено извршење једног скупа трансакција неће бити серијабилно. Задатак планера извршења је да провери серијабилност неким од наведених поступака и да спречи да несеријабилна извршења наруше интегритет базе података. Проверу серијабилности и предузимање одговарајућих акција планер извршења тешко може да обави у реалном времену. Због тога се серијабилности извршавања скупа трансакција најчешће остварује ″форсирано″ примењујући механизам **закључавања (locking)**. Овај механизам управљања извршењем скупа трансакција омогућава да трансакција ″закључа″ (″постави локот″) на објекат базе података коме је приступила, да би онемогућила да друге трансакције некоректно оперишу са истим објектом. Мада постоје више различитих врста закључавања, овде ћемо обрадити само две, ексклузивно и дељиво закључавање.

Ексклузивно закључавање (exclusive (XL) lock или write lock). Ако нека трансакција постави ексклузивни локот на објекат базе података, ниједна друга трансакција не може на тај објекат да постави било који други локот.

Дељиво закључавање (shared (SL) lock или read lock). Ако нека трансакција постави дељиви локот на објекат базе података, нека друга трансакција такође може да постави дељиви локот на исти објекат базе, али ниједна друга не може да постави ексклузивни локот на тај објекат.

Однос ексклузивног и дељивог локота може се представити ″матрицом компатибилности″ која је приказана на Слици 4.7. Локоти које постављају трансакције A и B представљени су у првој колони и првој врсти, респективно. Цртицом је означена ситуација у којој трансакција није поставила локот на објекат базе података.

A \ B	XL	SL	-
XL	NE	NE	DA
SL	NE	DA	DA
-	DA	DA	DA

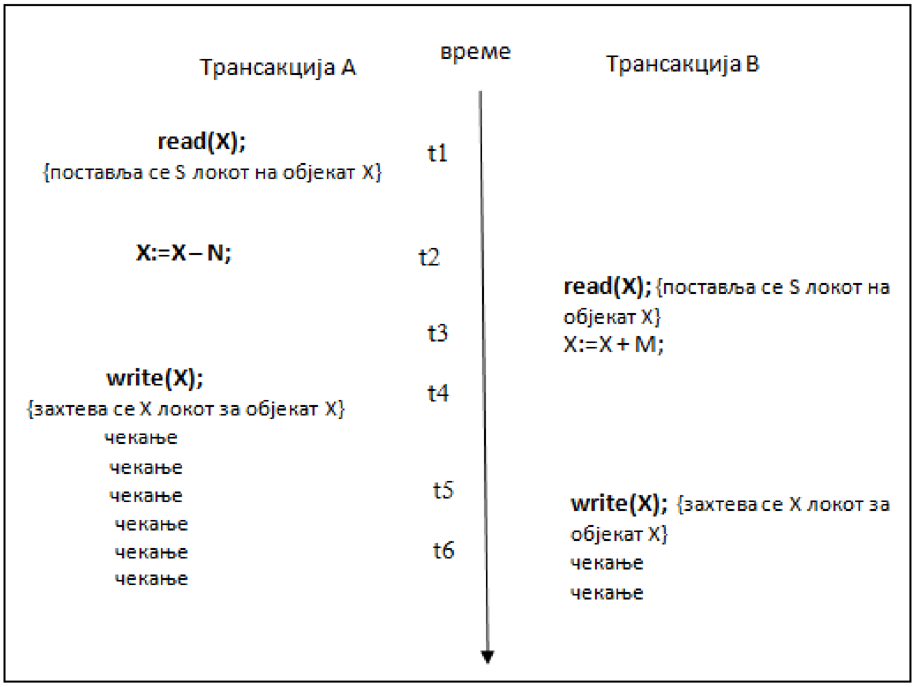
Слика 4.7. Матрица компатибилности типова закључавања [1]

Имајући у виду ексклузивни и дељиви локот, **протокол закључавања** који би могао да реши приказане проблеме може се исказати на следећи начин:

1. Трансакција која жели да прочита неки објекат базе података (n-торку, на пример) мора прво да постави дељиви локот на тај објекат;
2. Трансакција која жели да ажурира неки објекат базе података мора прво да постави ексклузивни локот на тај објекат. Ако је та трансакција раније поставила S локот, она треба да тај локот трансформише у X локот;
3. Ако трансакција није успела да постави локот на жељени објекат базе података, због тога што нека друга трансакција већ држи некомпатибилан локот над посматраним објектом, тада она прелази у стање чекања;
4. Трансакција ослобађа Е локот обавезно, а по правилу и S локот на крају, са COMMIT или ROLLBACK наредбом.

И X и S локот се постављају имплицитно, заједно са операцијама ажурирања, односно читања података базе.
Прикажимо како се одвијају трансакције у раније приказаним примерима уз примену наведеног протокола закључавања. На Слици 4.8 је приказан начин решавања проблема губљења резултата ажурирања са Слике 4.2. Као што се са Слике 4.8 види, дати протокол закључавања не доводи до губљења резултата ажурирања, али производи тзв. ″мртви чвор″ трансакција. Трансакција А не може у тренутку t3 да постави ексклузивни локот на објекат базе података X, зато што је Трансакција B већ поставила на њега дељиви локот. Обично се то исказује и на следећи начин: ″Трансакција А не може да врши ажурирање објекта X зато што га је Трансакција B већ видела″. Због тога Трансакција А прелази у стање ″чекања″. У тренутку t4 Трансакција B не може да постави ексклузивни локот на објекат X зато што је Трансакција А већ на њега поставила дељиви локот, па и Трансакција B прелази у стање чекања. Обе трансакције би, без додатних

механизама управљања њиховим одвијањем, остале заувек у стању чекања. Због тога се приказана ситуација назива ″мртви локот″.



Слика 4.8. ″Мртви чвор″ уместо губљење резултата ажурирања [1]

Начин на који се решава проблем непотврђених промена са Слике 4.4 приказан је на Слици 4.9. Трансакција В у тренутку t3 не може да добије дељиви локот на објекат X базе података, па прелази у стање чекања. По завршетку Трансакције А у тренутку t4 (наредбом Rollback), ослобађају се сви њени локоти и Трансакција В наставља своје извршавање у тренутку t5.

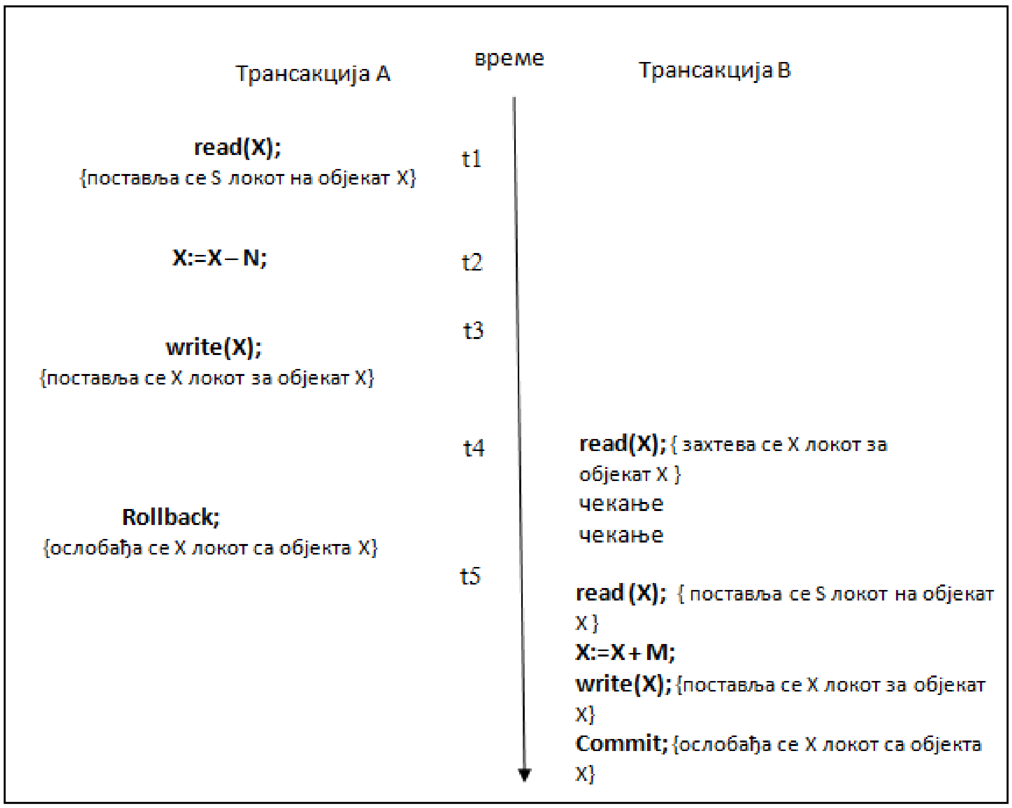
Горе исказани протокол обављања трансакција је само један строжији исказ за тзв. **двофазни протокол закључавања** који гласи:

1. Пре него што оперише са неким објектом базе података, трансакција мора да постави локот на њега;

2. После ослобађања неког локота, трансакција не може више поставити локот ни на један објекат базе.

Може се доказати теорема да **ако у неком скупу све трансакције поштују двофазни протокол закључавања, тај скуп се увек серијабилно извршава.** Исто тако, ако нека трансакција не поштује двофазни протокол закључавања, могуће је увек конструисати

другу трансакцију тако да њихово конкурентно извршење доводи базу у неконзистентно стање.
Очигледно је да прва фаза у двофазном протоколу закључавања представља фазу ″ширења трансакције″, трансакција поставља редом локоте на објекте базе који су јој потребни. У другој фази ″фази скупљања″ трансакција редом ослобађа локоте које је поставила.
Поред ексклузивног и дељивог, постоје и друге врсте локота као што су локот ажурирања (update lock), или инкрементни локот (increment lock) али се они овде неће детаљније приказивати.



Слика 4.9. Решење проблема непотврђених промена [1]

Временско означавање (Timestemping)

Означавање трансакција (timestemping), да би се на основу добијене ознаке уређивао редослед њиховог извршавања, је такође један од приступа за остваривање серијабилности извршења скупа трансакција. Свакој трансакцији се додељује јединствени идентификатор у редоследу у коме су трансакције доспеле у систем, па се идентификатор може третирати као време њеног почетка. Сва физичка ажурирања базе података се извршавају тек са COMMIT наредбом. Конфликти у извршењу трансакција настају када нека трансакција захтева да ″види″ неки рекорд кога је већ ″млађа″ трансакција (трансакција са мањим идентификатором) ажурирала, или када нека трансакција хоће да ажурира неки рекорд кога је ″млађа″ трансакција већ ажурирала или видела. Конфликти се разрешавају поновним стартовањем ″старије″ трансакције.

Сваком објекту базе података придружују се две ознаке:

- RMAX, највећи идентификатор трансакције која је успешно извршила читање посматраног објекта и
- UMAX, највећи идентификатор трансакције која је успешно извршила ажурирање посматраног објекта.

Алгоритам извршења трансакције T чији је идентификатор (временска ознака) t, односно њених операција читања и ажурирања базе података, дат је на Слици 4.10.

```
              read(R)
       if t>= UMAX
          then { operacija se prihvata}
           RMAX := MAX(t, RMAX);
      else { konflikt }
            restart T;
  write(R)  { zajedno sa COMMIT }
 if t >= UMAX and t >= RMAX
      then { operacija se prihvata}
            UMAX := t;
      else { konflikt }
            restart T;
```

Слика 4.10. Алгоритам временског означавања [1]

У овој техници нема закључавања објеката БП, интегритет је очуван, не појављују се ни живи ни мртви локоти. Међутим, техника временског означавања обично доводи до тога да се велики број трансакција поништава и поново стартује. Исто тако, реализација услова да се сва ажурирања неке трансакције обављају заједно са наредбом COMMIT у тој трансакцији захтева да се уписивања трансакција ″баферују″ у неку меморију и пренесу у базу тек са COMMIT наредбом. То значи да трансакције које желе да приступе елементима базе у овом баферу морају да чекају COMMIT, чиме се добија исти ефекат као и у протоколима закључавања.

<u>Поређење протокола за остваривање серијабилности</u>

На слици 4.11. приказан је однос ″строгости″ приказаних протокола за остваривање серијабилности скупа трансакција. Очигледно је да је најопштији протокол view-серијабилност. Конфликт-серијабилност је ″стриктно строжији″ протокол. Он дефинише само потребне, а не и довољне услове за серијабилност и због тога ″пропушта″ знатно мањи број извршења скупа трансакција. Другим речима постоје извршења која су серијабилна али не задовољавају услове дефинисане у конфликт-серијабилности. Двофазни протокол закључавања и временско означавање су још строжији протоколи, ″пропуштају″ још мањи број извршења и међусобно су пресечни. У пракси се најчешће користи двофазни протокол закључавања.

<u>Нивои изолованости трансакција</u>

Серијабилност скупа трансакција је највиши степен њихове међусобне изолованости у извршењу. Да би се остварио већи паралелизам у извршењу скупа трансакција и самим тим побољшале перформансе система, у комерцијалним СУБП се дозвољавају и нижи нивои изолованости трансакција.
SQL стандард дефинише следеће нивое изолованости трансакције наведене по ″строгости″: (1)SERIALIZABLE,(2)REPEATABLE READ, (3)READ COMMITTED,(4) READ UNCOMIMITTED. Ниво изолованости се третира као особина трансакције и поставља се преко исказа SET TRANSACTION. У SQL стандарду претпостављена (″default″) вредност је SERIALIZABLE. Да би се прецизно дефинисали ови нивои изолованости, стандард уводи и следеће начине нарушавања серијабилности извршења скупа трансакција:

Слика 4.11. Поређење протокола за остваривање серијабилности [1]

Прљаво читање (″dirty read″). Овако извршење скупа трансакција је већ раније објашњено. Ако трансакција T1 ажурира неки податак у бази, затим трансакција T2 прочита тај податак, а након тога се трансакција T1 заврши са ROLLBACK, тада је очигледно трансакција T2 прочитала непостојећи податак, односно извршила ″прљаво читање″.

Непоновљиво читање (Nonrepeatable read). У једној трансакцији, поновно читање истих података мора дати исти резултат. Напротив, ако трансакција T1 прочита неки податак базе, затим га трансакција T2 промени, поновно читање истог податка у трансакцији T1 неће дати исти резултат.

Фантомско читање (Fantoms). Ако трансакција T1 преузме упитом скуп n-торки које задовољавају задати услов, а после тога трансакција T2 дода нову n-торку која задовољава исти услов, ново извршење истог упита у трансакцији T1 садржаће и нову ″фантомску″ n-торку.

Различити нивои изолованости дозвољавају или не дозвољавају неке од приказаних начина нарушавања интегритета базе или стварних излаза, као што је приказано у табели на Слици 4.12.

Ponašanje / Nivo izolovanosti	PRLJAVO ČITANJE	NEPONOVLJIVO ČITANJE	FANTOMSKO ČITANJE
READ UNCOMMITTED	DA	DA	DA
READ COMMITTED	NE	DA	DA
REPEATABLE READ	NE	NE	DA
SERIALIZABLE	NE	NE	NE

Слика 4.12. Однос нивоа изолованости и понашање извршења скупа трансакција [1]

4.2.2. Управљање закључавањем

Највећи број комерцијалних СУБП-а користи протоколе закључавања за остваривање серијабилности извршења скупа трансакција. Због тога ћемо овде додатну пажњу посветити анализи поступака који се примењују у управљању закључавањем, односно опису тзв. **менаџера локота (lock manager).**

Основу управљања закључавањем чине три процедуре менаџера локота: (1) r_lock – постављање ексклузивног локота, (2) w_lock – постављање дељивог локота и (3) unlock – ослобађање закључаног елемента базе података.

```
r_lock (T,X, errcode, timeout)
w_lock (T,X, errcode, timeout)
ulock (T,X)
```

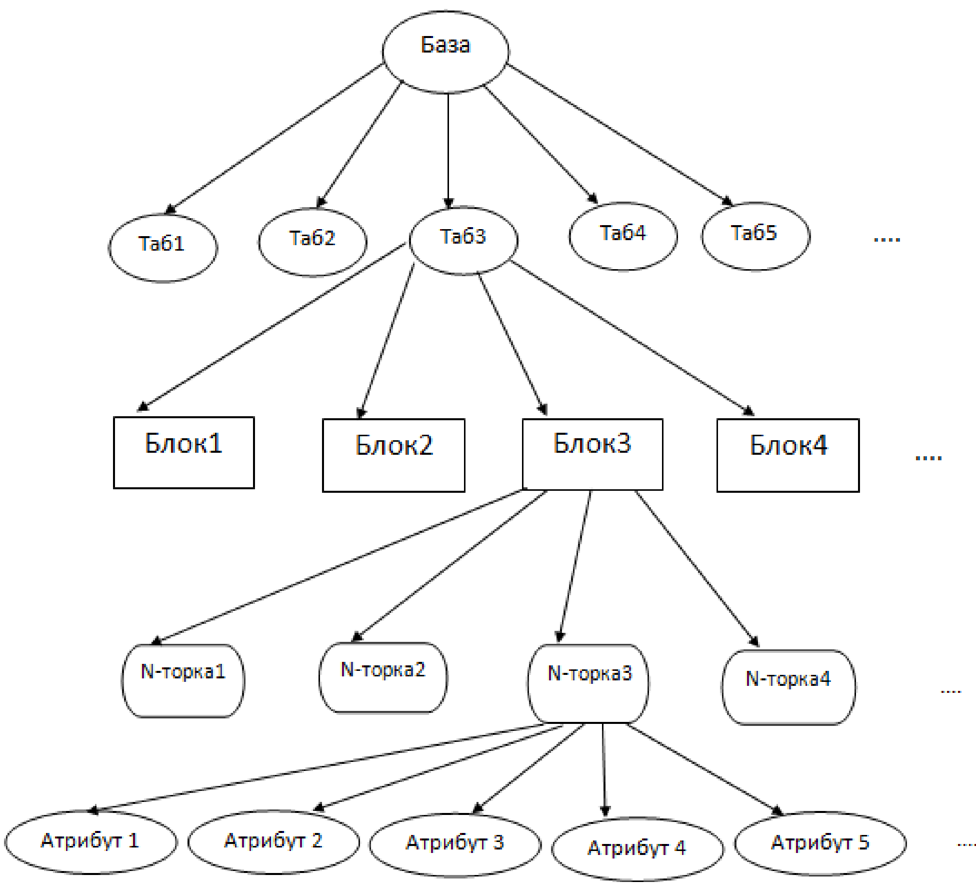
T – идентификатор трансакције
X – елеменат базе података који се закључава,
errcode – даје код статуса захтева (0-захтев задовољен, $\neq 0$ - незадовољен)
timeout – максимални интервал времена који трансакција ″чека″ да би добила локот над објектом X.

Менаџер локота управља и **табелом локота (lok table)** у којој се, уз елеменат базе података, држе информације о његовом статусу у погледу закључавања. Табела обично садржи идентификатор елемента базе, статус елемента у погледу закључавања, број трансакција у реду за дати елеменат и показивач на ред у коме се трансакције налазе. Када трансакција захтева локот над елементом базе података, ако је он у стању у коме захтев може да се испуни, стање елемента се промени и трансакција наставља рад. У противном трансакција се придружује реду за захтевани елеменат. Када се посматрани елеменат ослободи, менаџер локота испитује да ли постоји нека трансакција у реду и првој у реду дозвољава да промени стање елемента и настави рад. Када време које трансакција проведе у реду постане веће од ″timeout″ времена, врши се ROLLBACK и поновно стартовање исте трансакције.

<u>Грануларност закључавања и хијерархијско закључавање</u>

Један од битних карактеристика закључавања је **грануларност закључавања**. До сада смо користили генерички назив ″елеменат базе података″. Грануларност закључавања дефинише тип елемента који се закључава – цела база, неки њен фрагмент, табела, блок, n-торка или атрибут n-торке. Са физичке тачке гледишта најнижи елеменат од интереса је блок. Хијерархија елемента базе који могу да се закључају приказана је на Слици 4.13. Очигледно је да мања грануларност повећава паралелизам у обради трансакција, а већа поједностављује управљање закључавањем. Ова супротност се решава преко **хијерархијског закључавања**, у коме се дозвољава да трансакције, по посебном протоколу, закључавају елементе базе података различите грануларности. Поред ексклузивног локота (XL) и дељивог локота (SL) за овај протокол се дефинишу и следеће врсте локота:

- ISL – намеравани дељиви локот: трансакција намерава да постави дељиви локот на елеменат базе података;
- IXL – намеравани ексклузивни локот: трансакција намерава да постави ексклузивни локот на елемента базе података;
- SIXL – намеравано дељиво-ексклузивно закључавање: трансакција намерава да постави дељиви локот на елеменат базе података X и ексклузивни локот на неки елеменат на нижем нивоу грануларности који је део елемента X.



Слика 4.13. Хијерархија елемената закључавања [1]

У хијерархијском протоколу закључавање се обавља по следећим правилима:

1. Локоти се закључавају почев од врха, низ хијерархију елемената;
2. Локоти се ослобађају почев од елемента са најмањом грануларношћу, навише, уз хијерархију елемената.
3. Да би се поставио SL или ISL локот на неки елеменат X, трансакција мора претходно да постави ISL или IXL локот на елемент ″родитељ″ елементу X у стаблу;
4. Да би захтевала IXL, XI или SIXL локот на елеменат X, трансакција мора претходно да постави SIXL или IXL локот на елемент ″родитељ″ елементу X у стаблу;
5. Матрица компатибилности ових локота за трансакције A и B дата је на Слици 4.14.

A \ B	ISL	IXL	SL	SIXL	XL
ISL	DA	DA	DA	DA	NE
IXL	DA	DA	NE	NE	NE
SL	DA	NE	DA	NE	NE
SIXL	DA	NE	NE	NE	NE
XL	NE	NE	NE	NE	NE

Слика 4.14. Матрица компатибилности локота у хијерархијском закључавању [1]

Избор нивоа на коме ће се вршити закључавање препушта се пројектанту трансакције или администратору базе података. Избор зависи од врсте трансакције. Упит који захтева већи број елемената ефикасније ће се реализовати са већом грануларношћу закључавања (вишим нивоом елемента у хијерархији), док ажурирање једне n- торке, на пример, захтева мању грануларност закључавања.

Хијерархијско закључавање индексних структура

У претходном потпоглављу хијерархијско закључавање се базирало на угњежденој хијерархији елемената базе података – елементи ″деца″ су саставни део елемента ″родитеља″. У физичкој организацији базе података постоје и друге врсте хијерархијских структура, хијерархијске структуре које дефинишу приступ елементима базе података. Чињеница да приступ једном елементу има предефинисану путању која пролази кроз неке друге елементе базе, може да се искористи за дефинисање специфичних протокола закључавања. У релационим базама података овакве структуре су индексне структуре, најчешће B-стабла.
Ако би се на B-стаблима примењивале стандардне врсте локота (ексклузивни и дељиви) и стандардни протоколи закључавања (Двофазни протокол закључавања) тако да се цело B-стабло закључава, јер се претпоставља да ће бити потребно и његово ажурирање, паралелизам у извршењу скупа трансакција би се готово потпуно елиминисао. Међутим, могуће је веома лако предвидети да ли ће нека трансакција ажурирати посматрани ниво у B-стаблу. Ако трансакција убацује n-торку и ако одговарајући чвор није пун, убацивање неће захтевати ажурирање датог чвора. Исто тако, ако трансакција избацује неку n-торку, кретањем низ индексну структуру и анализирањем броја записа у односу на минимални број може се одредити који чворови неће бити ажурирани. На основу ове чињенице могао би се дефинисати следећи протокол: Крећући се низ B-стабло трансакција скида локоте са оних чворова који неће бити ажурирани. Међутим овакав протокол нарушава правила Двофазног протокола закључавања, па због тога не може да гарантује серијабилност извршења скупа трансакција. Због тога се дефинишу специјални протоколи за хијерархијске, односно индексне структуре. Један такав протокол је:

1. Трансакција може поставити свој први локот на било који чвор стабла;
2. Следећи локоти се могу добити само ако је дата трансакција поставила локот и на чвор ″родитељ″;
3. Локоти се могу ослободити у било ком тренутку;
4. Трансакција не може поново да постави локот на елеменат са кога је скинула локот, чак и ако још увек држи локот на чвору ″родитељу″.

Мада не поштује у потпуности правила Двофазног протокола закључавања, може се показати да приказани протокол форсира серијско извршење трансакција у посматраном извршењу.

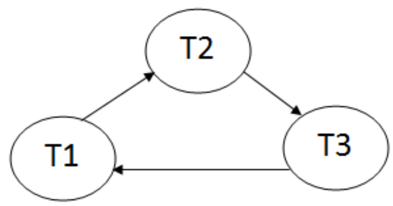
″Живи″ и ″мртви″ локоти

Као што је у неким од претходних примера показано, у току извршења скупа трансакција, могуће је да две или више трансакција формирају ″мртви локот″, односно да локоти које су оне поставиле на објекте базе података све њих доводе у стање чекања. Поред појма ″мртвог локота″, понекад се дефинише и појам ″живог локота″. То је ситуација у којој је нека трансакција стално у стању чекања на неки објекат базе података због тога што друге трансакције увек пре ње поставе локот на тај објекат. Проблем ″живог локота″ једноставно се решава увођењем неког редоследа закључавања објекта (на пример FIFO).

За разрешавање проблема мртвих локота користе се три технике:

1. **Прекидање трансакције после истека интервала времена** предвиђеног за чекање на локот за неки елеменат. Ово правило користи параметар ″timeout″ који менаџер локота додељује трансакцији при покушају закључавања неког објекта. Када ово време истекне трансакција се поништава и поново стартује, очекујући да тада неће произвести ″мртви локот″.
2. **Превенција локота.** Могу се дефинисати различити протоколи који ће спречити да до ″мртвог чвора″ дође. Један од таквих протокола се заснива на уређењу елемената базе података. На пример, блокови се могу уредити по њиховим адресама на спољној меморији ($A1 < A2 < A3 <...< An$). Ако се захтева да свака трансакција закључа елементе у њиховом дефинисаном редоследу, очигледно је да до ″мртвог чвора″ не може да дође. На пример ако T1 захтева A1 па A2, трансакција T2 неће моћи да тражи локоте у редоследу A2 па A1, који би довео до ″мртвог чвора″, јер овако закључавање није по дефинисаном протоколу. Други протокол превенције заснива се на додељивању, за овај протокол специфичне, временске ознаке трансакцији. Ако је трансакција која захтева локот старија (мања временска ознака) од трансакције која држи локот на елементу базе, дозвољава јој се чекање на добијање локота, у противном се прекида. Могуће је применити и обрнути услов прекидања.
3. **Детекција ″мртвог чвора″.** Дозвољава да ″мртвог локота″ дође, па се тада нека од трансакција која га је изазвала ″убије″, њени ефекти на базу података пониште, а она сама, евентуално, поново стартује, надајући се да тада неће изазвати мртви локот. За откривање ″мртвих локота″ у систему користи се тзв. граф чекања (Wait-For граф) чији су чворови трансакције T у извршењу, а грана Ti – Tj постоји ако трансакција Ti захтева неки објекат који је трансакција Tj закључала. На Слици 4.15 приказан је један граф чекања. Трансакција T1 чека на објекат који је закључала трансакција T2, ова чека на објекат који је закључала T3, а T3 чека на објекат који

је закључала Т1. Испитивање мртвих локота се може вршити било сваки пут када нека трансакција прелази у стање ″чекај″, било периодично, или се може сматрати да је мртви локот настао ако трансакција ништа није урадила у предефинисаном периоду времена. Очигледно је да се налажење ″мртвог локота″ своди на налажење циклуса у графу чекања. Критеријуми за избор трансакције која ће се поништити (″жртве″) могу бити различити, на пример, трансакција која је последња стартовала или трансакција која захтева најмање локота. ″Убијање″ трансакције и поништавање њених ефеката ослобађа и све локоте које је она поставила.



Слика 4.15. Граф чекања [1]

4.2.3. Дугачке трансакције

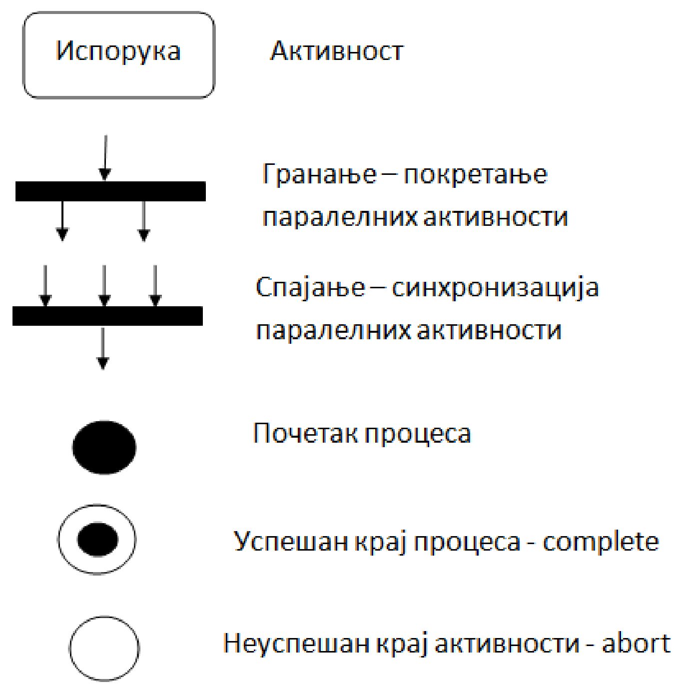
Претходно дискутоване методе управљања извршењем скупа трансакција погодне су за пословне системе са једноставним кратким трансакцијама, када је прихватљиво да једна трансакција држи закључаним неки елеменат базе података за неко релативно кратко време, реда величине десетинке секунде, секунде или чак и минута, зависно од врсте апликација. Међутим, постоје апликације у којима су трансакције много дуже, реда десетине минута, сати или чак неколико дана. Приказани механизми закључавања у коме би нека трансакција овако дуго ексклузивно закључала елементе базе података које користи је, очигледно, неприхватљиво. Примери апликативних система са дугим трансакцијама су:

- **Неке трансакције у пословним апликацијама**. На пример испитивање свих рачуна у некој банци да би се проверило укупно стање рачуна банке. Или, реконструкција индексне структуре у физичкој организацији базе података да би се побољшале перформансе система.
- **Пројектни системи** су различите врсте CAE, CAD, CASE и других система преко којих се пројектују различити инжењерски системи. Претпоставља се да више пројектаната ради на истом пројекту и да постоји заједничка база пројекта. Један пројектант може да преузме један део података из базе и да са њим ради свој део пројекта и неколико дана (недеља). Очигледно је да постоји потреба да истим

подацима приступе и други пројектанти, не чекајући да их први пројектант ″потврди″ у бази и да ослободи постављене локоте.

- **″Workflow″ системи** су системи у којима се једна логичка јединица посла за корисника, обавља преко секвенце активности од којих неке могу бити програми (активности које се обављају аутоматски), неке интерактивне апликације (обављају се у комуникацији човека и рачунара) а неке су реализоване и потпуно ручно. Преко ″workflow″ система се описују сложени пословни процеси. Пример једног таквог пословног процеса приказан је преко UML Дијаграма активности на Слици 4.16.

Дијаграм активности је граф који има следеће врсте чворова:

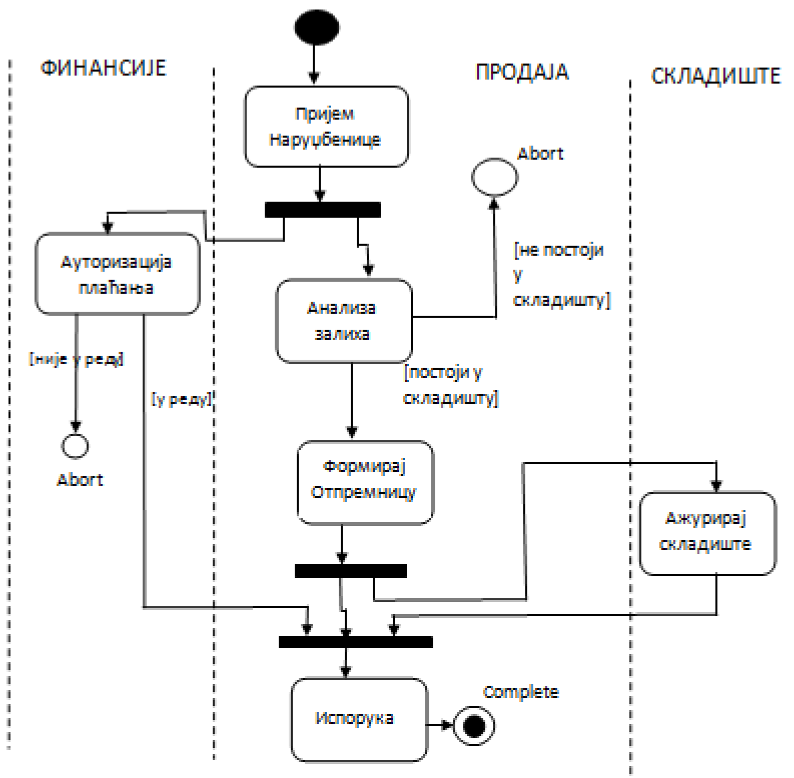


Стандардни скуп чворова у UML-у проширен је за чвор ″Неуспешан крај активности - abort″ да би се увео основни концепт у теорији управљања дугачким трансакцијама – концепт ″саге″.

Сага се може дефинисати као граф који има горе приказане типове чворова. Путања од почетног чвора до било ког терминалног чвора (complete или abort) представља могућу секвенцу акција. Конкурентно управљање сагама се обавља на следећи начин:

1. Свака активност се може третирати као посебна ″кратка″ трансакција која се извршава под контролом конвенционалних механизама за управљање трансакцијама.

2. Целокупна дугачка трансакција, једна од путања од почетног чвора до успешног или неуспешног завршетка, се обавља на тај начин што се, у случају успешног краја задржавају све промене које су учиниле поједине активности, а у случају неуспешног, користи се механизам **компензационих трансакција**, да би се база података вратила у исто стање у коме је била пре отпочињања саге.



Слика 4.16. Дијаграм активности једног пословног процеса [1]

Свака активност (акција) A у саги има своју компензациону трансакцију коју означавамо са A^{-1}. Интуитивно, ако се изврши акција A, а затим A^{-1} база података се враћа у исто стање у коме је била пре почетка ових акција. Формално, ако је D стање базе података, $B_1, B_2, ..., B_n$ било која секвенца акција и компензационих трансакција, из исте или различитих сага, тада исти резултат дају секвенце $B_1, B_2, ..., B_n$ и $A_1, B_1, B_2, ..., B_n, A^{-1}$.

4.3. Опоравак базе података

Опоравак базе података је враћање базе података у коректно стање после неког отказа. Да би се опоравак базе података могао успешно да изврши неопходно је да СУБП обезбеди неке редундантне податке. Један скуп редундантних података се чува у већ поменутом **логу** или **журналу**. Други скуп редундантних података чува се у тзв. **архивској меморији** у коју се повремено преноси садржај целокупне базе (″dump″ базе). Лог или журнал се користе за отказе који физички не оштећују меморијске јединице (дискове) на којима се чува база, док архивска меморија служи за опоравак после оваквих отказа. Типична стратегија за опоравак базе података је:

- Ако су оштећене меморијске јединице на којима се чува база података (пад глава диска, на пример), користи се последња архивска копија целе базе за њен опоравак. Администратор система одређује интервале у којима се врши ″damp″ на архивску меморију. За то време, ниједна трансакција над базом не би требало да буде активна. Претпоставља се да систем располаже са dump/restore, односно unload/reload сервисима.

- Када база података није физички оштећена, опоравак се врши коришћењем лога, преко посебно дефинисаних протокола опоравка. У овом случају СУБП треба да води рачуна о следећим врстама отказа:

    1. Софтверски или хардверски отказ који не оштећује базу, а у коме се може изгубити садржај централне меморије (нестанак напајања, на пример);

    2. Отказ у самој трансакцији, на пример због дељења са нулом, давања параметара погрешног типа и слично;

    3. Акције система за конкурентну обраду трансакција, убијање трансакције која је изазвала ″мртви локот″ или несеријабилност извршења скупа трансакција и слично.

Један типичан протокол опоравка базе података заснива се на тзв. **″тачкама испитивања″ (checkpoint)**. У појединим предефинисаним тренуцима времена на лог се уписује **рекорд тачке испитивања**. Овај рекорд садржи листу трансакција које су у том тренутку у извршењу. У тренутку регистровања тачке испитивања форсирано се уписују подаци у базу из њеног бафера. Опоравак по овом протоколу врши се на следећи начин:

1. Формирају се две листе трансакција, UNDO (за поништавање) и REDO (за обнављање). У UNDO листу се стављају све трансакције регистроване у последњој тачки испитивања, а REDO листа је у почетку празна;

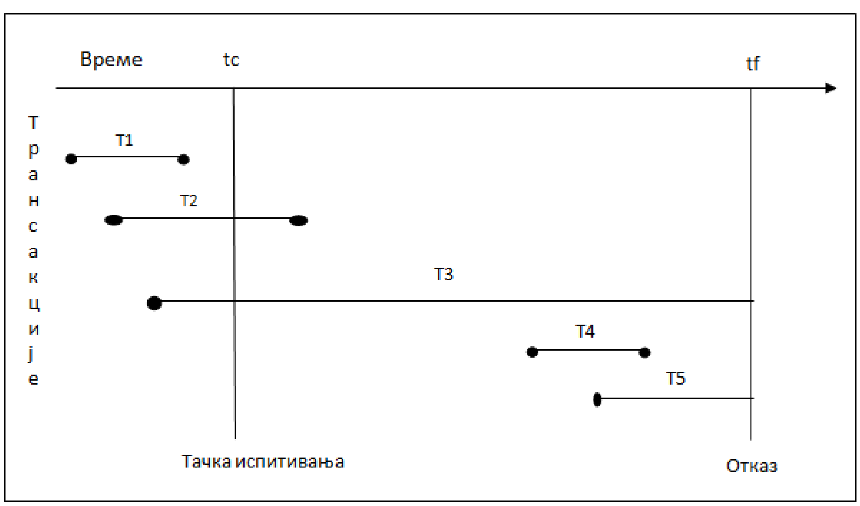
2. Претражује се лог унапред, од тачке испитивања;

3. Ако се нађе BEGIN TRANSACTION за трансакцију T додаје се T у UNDO листу;
4. Ако се нађе COMMIT за трансакцију T, трансакција T се пребацује из UNDO у REDO листу;
5. Користећи ″вредности пре″ и ″вредности после″ из лога, поништавају се све трансакције које су у UNDO листи, а обнављају оне које су у REDO листи.

Протокол је илустрован скупом трансакција на Слици 4.17. Трансакција T1 не улази у процес опоравка, њени ефекти су дефинитивно потврђени у тренутку tc, заједно са регистровањем нове тачке испитивања. Трансакције T3 и T5 се очигледно поништавају, док се трансакције T2 и T4 обнављају.

4.3.1. Опоравак у дистрибуираним и вишеструким базама података

У досадашњим анализама претпостављено је да се трансакције извршавају над једном базом података, односно у оквиру једног система за управљање базом података. У савременим архитектурама информационих система могуће је да једна трансакција приступа подацима у различитим базама, преко различитих СУБП. Свако СУБП има свој сопствени механизам за управљање трансакцијама и свој сопствени лог. На пример, једна трансакција може да ажурира неке податке у бази под ORACLE СУБП и неке податке у бази под SQL сервером. Да би се обезбедила атомност оваквих трансакција неопходно је обезбедити двонивоски механизам опоравка базе података. На првом нивоу се налазе механизми опоравка локалних СУБП, а на другом тзв. **координатор** који координира механизме опоравка са првог нивоа. Координатор имплементира тзв. двофазни **протокол потврђивања (two-phase commit protocol)**, односно дефинише две фазе у потврђивању промена које трансакција изврши у различитим базама:



Слика 4.17. Примери трансакција за анализу протокола опоравка [1]

- **Фаза1**. Када сви СУБП на првом нивоу пошаљу сигнал координатору да је део трансакције којим управљају завршен, координатор им одговара поруком ″припреми се″. СУБП првог нивоа тада уписују неопходне податке на свој лог и шаљу координатору сигнал ″OK″. Ако упис на лог није успео, шаље се порука ″NOT OK″. Ако координатор не добије одговор СУБП-а првог нивоа у предефинисаном интервалу времена, подразумева се порука ″није у реду″.

- **Фаза2**. Ако су сви СУБП првог нивоа послали поруке ″OK″, врши се потврђивање (COMMIT) свих делова трансакције. У противном, поништавају се промене (ROLLBACK) свих делова трансакције на свим локалним СУБП.

4.3.2. Трансакциона архитектура у SQL Server-у 2008

Да би сам процес опоравка података био једноставнији и успешнији, веома је значајно разумевање SQL Server трансакционе архитектуре.
То значи да ће познавање процеса који је везан за извршавање трансакција омогућити кориснику да донесе много интелигентније одлуке везане за величину, локацију, бекаповање и рестаурирање база података и трансакционих логова.

Свака SQL Server база података садржи одговарајући трансакциони лог који садржи записе везане за све промене које су у тим базама извршене. Модификације се најпре извршавају у трансакционом логу, па тек онда у датотекама базе података. Један од главних разлога због кога овај лог прво бележи модификације је опоравак базе података.

Записи се у трансакциони лог снимају секвенцијално, а свака трансакција добија одговарајући идентификатор. Над тим записима се периодично обавља провера и бележи у трансакционом логу. Све промене које су извршене од последње тачке провере записују се у датотеке базе података.
Трансакциони лог би могао да се примени у случају прекида рада рачунара насталог услед тренутног прекида напајања, тако што би се уз помоћ њега утврдило стање у коме се налазио систем пре настанка прекида напајања. У том процесу прво се анализира садржај трансакционог лога од последње идентификоване тачке провере, затим потврђују све потпуно извршене трансакције, и на крају поништавају све трансакције које нису потпуно извршене у тренутку када је прекинуто извршавање.

<u>**Модели опоравка база података**</u>

Сврха модела опоравка кога је потребно дефинисати за базу података, је у томе што он утврђује на који начин се мења трансакциони лог, тј. може ли трансакциони лог да се бекапује. Најпознатији модели опоравка су Simple, Bulk-logged и потпуни модел опоравка.
Приликом примене Simple модела опоравка, трансакциони лог се празни након сваке тачке провере. У том случају, бекаповање трансакционог лога није могуће, зато што овај лог не садржи све модификације које су извршене над подацима присутним у бази

података, па се лог превасходно користи за опоравак базе података приликом стартовања сервиса.
Уколико се, међутим, користе потпуни или Bulk-logged модел опоравка, јавља се много мање потенцијалних губитака података, а трансакциони лог треба да се редовно бекапује.

Full модел опоравка омогућава смањење потенцијалних губитака на минимум због чега га користи највећи број продукционих база података.

Bulk-logged модел опоравка користи мање простора за извршене операције, као што је креирање индекса, или извршавање SELECT...INTO наредбе. Главни недостатак коришћења овог модела опоравка је да можемо да обновимо само целокупан бекап трансакционог лога, тако да повратак у одређену тачку у времену није могућ.

Специфичности бележења података о трансакцијама

Као што је у претходном тексту поменуто, трансакциони лог се може третирати као секвенца записа. Сваки појединачни запис је дефинисан бројем (Log Sequence Number, скраћено LSN), као и идентификационим бројем трансакције, који га повезује са другим записима у истој трансакцији.
Бекапови садрже метаподатке о опсегу LSN бројева који су забележени приликом бекаповања. Овакви метаподаци могу да се примењују приликом утврђивања секвенце операција рестаурирања, које су неопходне како би се извршио потпуни опоравак базе података. Уколико је потребно да се убрза поступак опоравка трансакције, зависни логови се повезују у ланац. У трансакционом логу се, такође, бележе и све операције поништавања, било да су оне последица извршавања експлицитне ROLLBACK наредбе или појаве грешке.

Постоје два различита начина снимања модификације у трансакциони лог, зависно од типа извршене модификације.
Код неких модификација захтева се снимање записа саме логичке операције. У овом случају, неопходно је извршити супротну операцију да би се поништило извршавање логичке операције.
У другом случају, када се модификације прате пре и после снимања модификованог записа, њихово поништавање захтева коришћење снимка података пре него што је извршена модификација.

Трансакциони лог је на диску смештен у једној или више физичких датотека, мада је много чешћи случај да он буде смештен у једној датотеци. Уколико се примењује Full или Bulk-logged модел опоравка, записи у трансакционом логу се уклањају приликом његовог бекаповања, док се у случају примене Simple модела записи уклањају приликом појаве сваке тачке провере. Простор који су првобитно заузимали уклоњени записи у трансакционом логу, може се поново користити. Трансакциони лог може бити састављен и од више датотека, при чему није могуће истовремено приступити већем броју тих датотека. Приликом коришћења већег броја датотека трансакционог лога, свака датотека се може више пута користити пре поновног приступања првој датотеци лога.

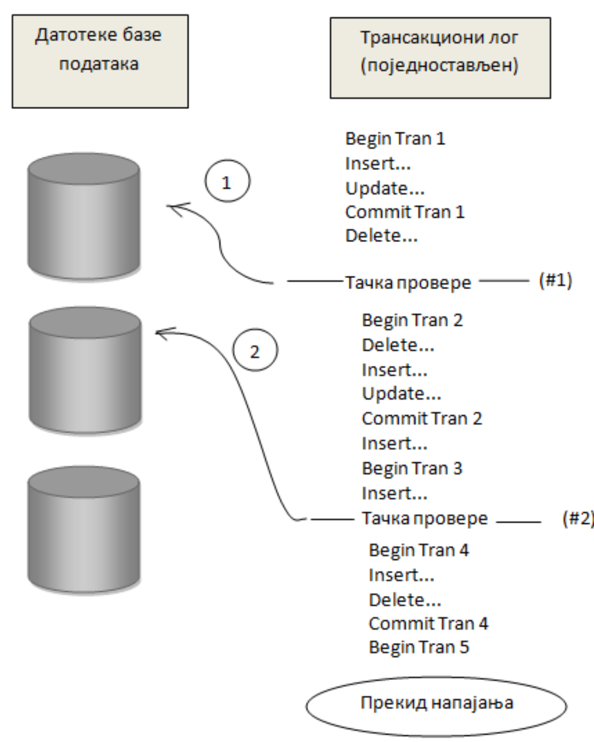
У случају да се трансакциони лог, који је конфигурисан тако да се аутоматски повећава, потпуно испуни, тада ће се овакав лог аутоматски повећати на основу унапред дефинисаних инкремената за аутоматско повећање. Ово се може обављати све док се не достигне максимална величина наведена за датотеку.

Уколико је трансакциони лог испуњен, а додатно повећање његове величине није могуће, јавиће се порука о појави грешке 9002, што значи да мора бити бекапован. У случају да је трансакциони лог у потпуности испуњен, ниједна модификација не сме да се изврши над подацима који се налазе у бази података, све док се не ослободи неопходан простор за складиштење информација о извршеним модификацијама.

Информације о извршавању следећих операција снимају се у облику записа у трансакционом логу:

- Почетак и крај извршавања сваке појединачне трансакције
- Све модификације података, које су везане за извршавање INSERT, UPDATE и DELETE наредби
- Дефинисање, модификовање и уклањање табела и индекса
- Алоцирање и деалоцирање основних страница и страница вишег нивоа у бази података

На Слици 4.18 је приказан пример секвенце операција у трансакционом логу, односно шта се догађа уколико у одређеном тренутку дође до прекида напајања. Трансакције се секвенцијално бележе у трансакциони лог, а периодично се извршава провера, након које се потврђена трансакција примењује на датотеке базе података.



Слика 4.18. Пример операција везаних за тачку провере и опоравак након прекида напајања [4]

Након што се достигне прва тачка провере, Трансакција 1 и имплицитна DELETE наредба се примењују на датотеке базе података. Када се достигне друга тачка провере, Трансакција 2 и INSERT наредба се примењују на датотеке базе података. Трансакција 3 није потврђена у том тренутку, тако да се операције које су дефинисане овом трансакцијом не примењују на датотеке базе података. Када престане напајање, односно SQL Server се поново стартује, процес опоравка почиње од претходне тачке провере. Трансакција 3 и Трансакција 5 се поништавају, зато што нису потврђене пре прекида напајања. Трансакција 4 наставља са извршавањем у процесу опоравка.

4.3.3. Стратегије бекаповања и рестаурирања

Веома је важно имати документовану и добро дефинисану стратегију за бекаповање и опоравак база података. Није довољно да се само бекапују корисничке базе података, већ је неопходно бекаповати и системске базе података.

Иако мастер база података не мора да се бекапује сваких сат времена, треба је бекаповати увек када се додају нове базе података, креирају нови налози, или када се на било који други начин мења конфигурација система.

Модел базе података је једноставан шаблон за нове базе података на SQL серверу, и треба га бекаповати сваки пут када се јави одређена промена.

Базе података са великим бројем промена треба чешће бекаповати.

База података, чији се подаци само очитавају, не мора стално да се бекапује, већ само онда када се јаве измене података.

Различити типови бекапова захтевају различит простор за складиштење. Стратегија бекаповања не може да захтева више простора од оног који кориснику стоји на располагању. Веома је важно утврдити колико дуго је неопходно чувати одговарајуће бекапове, јер то има утицај на величину простора који је неопходан за бекаповање.

<u>Модели опоравка и стратегија бекапа</u>

Избор одговарајућег модела опоравка је веома битан за планирање и извршавање одговарајуће стратегије бекаповања.

Simple модел опоравка

Овај модел опоравка базе података захтева најмање одржавање, али је највећа вероватноћа да ће доћи до губитка података уколико се јави отказ система. Приликом коришћења Simple модела опоравка, дозвољени су само Full и Differential бекапови.

Simple модел опоравка се користи у следећим ситуацијама:

- Ако се јави проблем, прихватљиво је да изгубите информације од последњег потпуног бекапа.

- Захтеви везани за одржавање трансакционог лога су много већи од користи које имате од тога.

- База података се користи приликом тестирања или се подаци из базе података само очитавају, тако да је могуће веома једноставно поновно креирање неопходних података.

Full модел опоравка

Приликом примене Full модела опоравка, све модификације извршене над базом података се бележе у трансакционом логу након што се обави потпуни бекап базе података. О свим трансакцијама се бележе све информације, укључујући и операције као што су SELECT...INTO и CREATE INDEX.
Full модел опоравка може да омогући опоравак базе података од одговарајућег отказа, под условом да постоје неопходни подаци у трансакционом логу. То је један од разлога због кога је неопходно стално складиштење садржаја трансакционих логова на посебном поузданијем медијуму, издвојеном од датотека база података.

Овај модел опоравка се користи у следећим ситуацијама:

- Неопходно је да сведете на минимум губитак података приликом појаве отказа система.

- Неопходно је да имате могућност опоравка базе података, тако да повратите стање које је било пре отказа.

- Желите да имате могућност опоравка базе података, тако да повратите стање које је било у одређеном тренутку у прошлости.

- Неопходно је да имате могућност да рестаурирате појединачне странице базе података.

- Желите да имате могућност да као администратор извршавате редовно бекаповање трансакционог лога.

Bulk-Logged модел опоравка

Овај модел опоравка омогућава бележење мање информација у трансакциони лог приликом извршавања операција, тако да се убрзава извршавање операција, али по цену немогућности враћања стања базе података на стање које је било у одређеном тренутку у прошлости.

Операције за које се могу минимално забележити подаци приликом коришћења Bulk-Logged модела опоравка су:

- SELECT..INTO операције
- INSERT..SELECT операције
- Bulk Import операције (извршавањем bcp или BULK INSERT наредбе)
- DDL операције за рад са индексима, као што су креирање нових индекса, поновно индексирање или уклањање груписаних индекса, чиме се захтева поновно креирање хипа.

Bulk-Logged модел опоравка треба да се користи у комбинацији са Full моделом опоравка. База података може да користи Bulk-Logged модел пре извршавања сложених операција, а затим се може поново прећи на Full модел опоравка након извршавања сложених операција.

Врсте бекапова

SQL сервер подржава две врсте бекапова: потпуни и диференцијални, а постоји и могућност бекаповања трансакционог лога.

Потпуни бекап омогућава креирање копије података и записивање те копије у датотеку за бекаповање, као и део трансакционог лога, како би се омогућило детектовање промена које су извршене над базом података од последњег потпуног бекапа.

Диференцијални бекап омогућава снимање података измењених након претходног потпуног бекапа.

Потпуни и диференцијални бекапови могу се примењивати над целокупном базом података, групом датотека или појединачним датотекама базе података.

Бекаповање трансакционог лога је могуће за базе података у којима се примењују Full или Bulk-logged модел опоравка. Пре него што се изврши бекаповање трансакционог лога, неопходно је извршити потпуни бекап базе података.

4.3.4. Креирање бекапова

Бекапови могу да се обављају извршавањем Transact-SQL наредби или коришћењем SQL Server Management Studio alata.

Потпуни бекапови

Потпуно бекаповање базе података подразумева бекаповање свих података који се налазе у бази података.

Када се ради о базама података за које се примењује Simple модел опоравка, подржани су једино потпуни и диференцијални бекап. Уколико се појави отказ, све трансакције након претходног бекапа биће изгубљене.

У случају да се ради о базама података које захтевају већу могућност опоравка, најбоље је употребити Full модел опоравка у комбинацији са диференцијалним бекаповањем и бекаповањем трансакционог лога.

Потпуни бекап базе података се може извршити коришћењем Transact-SQL наредби, SQL Server Management Studio алата или коришћењем планова за одржавање базе података. У наредном примеру биће приказане Transact-SQL наредбе за извршавање бекапова.

У листингу 4.1. бекапује се TestDB база података у привремени простор за бекаповање. Уређај за бекаповање може да складишти више од једног бекапа.

Листинг 4.1. Креирање потпуног бекапа базе података
BACKUP DATABASE TestDB TO DISK = ‘C:\Backups\TestDBFullBackup.bak’ ;

Диференцијални бекапови

Диференцијални бекапови се креирају коришћењем исте наредбе као и потпуни бекапови, осим што се мора навести WITH DIFFERENTIAL опција у BACKUP наредби. Да би се извршило диференцијално бекаповање, неопходно је претходно креирати потпуни бекап базе података.

У листингу 4.2. приказан је пример извршавања диференцијалног бекапа коришћењем Transact-SQL наредбе.

Листинг 4.2. Извршавање диференцијалног бекапа коришћењем Transact-SQL наредби
BACKUP DATABASE TestDB TO DISK = ‘C:\Backups\TestDB_Diff.bak’ WITH DIFFERENTIAL;

Бекаповање трансакционих логова

Бекаповање трансакционог лога може да се примењује над базама података које користе Bulk-logged или Full модел опоравка. Пример бекаповања трансакционог лога коришћењем Transact-SQL наредби је приказан у листингу 4.3.

Листинг 4.3. Бекаповање трансакционог лога коришћењем Transact-SQL наредбе
BACKUP LOG TestDB TO DISK = ‘C:\Backups\TestDB_Log.bak’ ;

<u>Парцијални бекапови</u>

Парцијални бекап је погодан за базе података које садрже једну или више група датотека чији се садржај може само читати.

Пример креирања парцијалног бекапа је приказан у листингу 4.4.

Листинг 4.4. Креирање парцијалног бекапа базе података
BACKUP DATABASE TestDB READ_WRITE_FILEGROUPS TO DISK = ‘C:\Backups\TestDB_PartialBackup.bak’ ;

<u>Пример креирања бекапова коришћењем Management Studio алата</u>

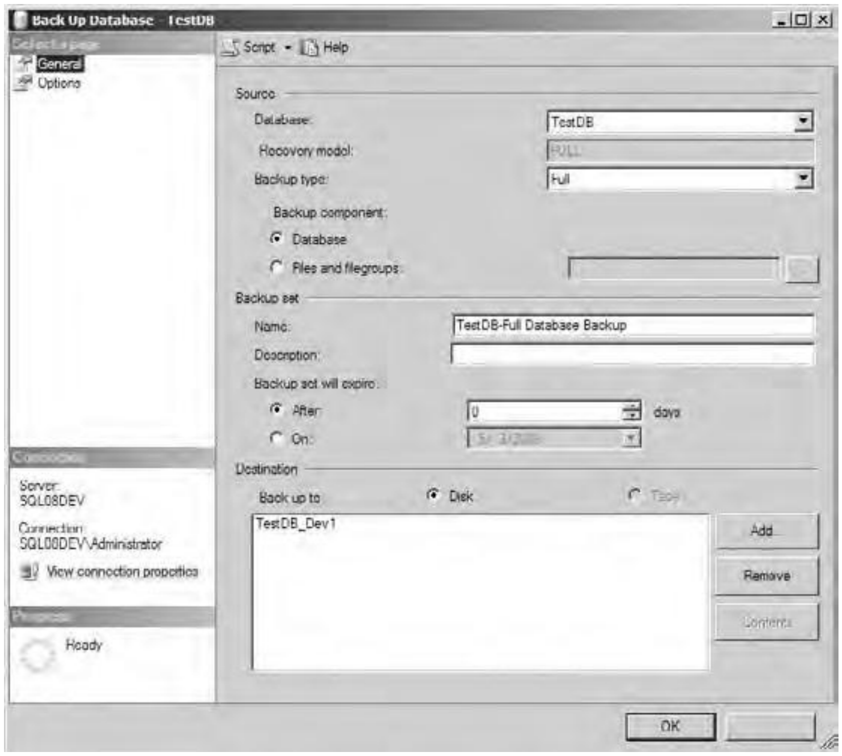
Приликом креирања бекапова може се користити кориснички интерфејс SQL Server Management Studio алата. Интерфејс алата омогућава да се генерише скрипт на основу изабраних опција коришћењем Script тастера, који се налази у горњем делу екрана.
Да би се реализовали бекапови коришћењем SQL Server Management Studio алата, неопходно је урадити следеће:

1. Отворите SQL Server Management Studio, а затим се, помоћу Object Explorer прозора, повежите са одговарајућом SQL Server инстанцом која садржи базу података коју желите да бекапујете.

2. Приступите Databases директоријуму и опционо System Databases директоријуму, зависно од тога коју базу података желите да бекапујете.

3. Десним тастером миша селектујте базу података коју желите да бекапујете, изаберите Tasks ставку менија, а затим Back Up ставку.

4. Отвара се Backup Database дијалог, као што је приказано на Слици 4.19. Проверите да ли је селектована исправна база података у Database падајућој листи.



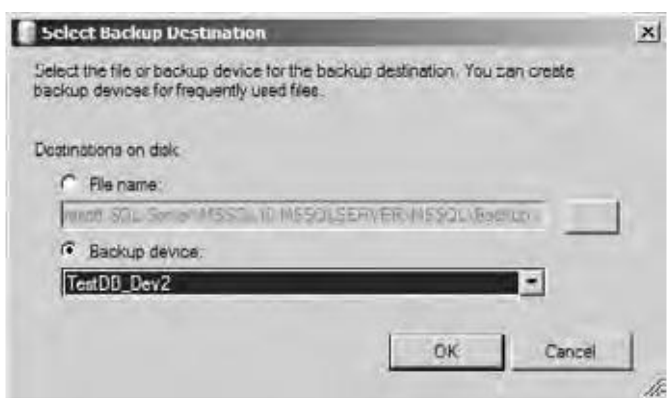
Слика 4.19. Backup Database дијалог [4]

5. Изаберите одговарајући тип бекапа (Full, Differential или Transaction Log).

6. Изаберите одговарајућу компоненту за бекаповање. Уколико бекапујете трансакциони лог, можете да пређете на следећи корак.

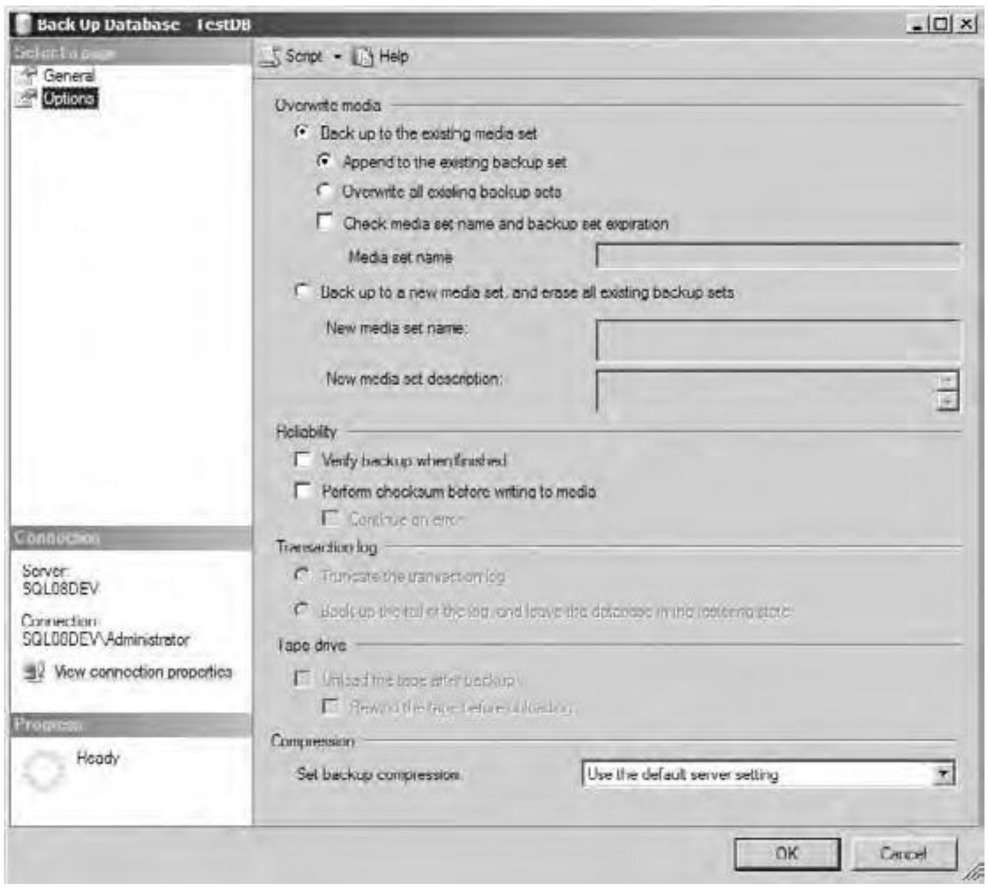
7. Опционо, дефинишите назив, опис и рок трајања скупа бекапова.

8. Додајте уређај који ћете користити за смештање бекапа, тако што ћете притиснути Add тастер. Отвара се Select Backup Destination дијалог, који је приказан на Слици 4.20. Можете да додате или назив датотеке, или логички уређај за бекаповање.



Слика 4.20. Select Backup Device Destination дијалог [4]

9. Селектујте Options страницу, која се налази у горњем левом углу. Страница са опцијама је приказана на Слици 4.21.



Слика 4.21. Options страница Backup Database дијалога [4]

10. На основу иницијалних подешавања, нови подаци се додају на крају постојећег скупа медија. Да бисте уклонили све скупове бекапова у скупу медија, неопходно је да изаберете Overwrite All Existing Backup Sets опцију. То је еквивалентно коришћењу WITH INIT Transact-SQL опције. Back Up to a New Media Set и Erase All Existing Backup Sets опције еквивалентне су коришћењу WITH FORMAT Transact-SQL опције.

11. Уколико желите да верификујете бекап, или желите да проверите контролну суму пре него што снимите бекап на медијум, селектујте одговарајуће Reliability опције.

12. Уколико бекапујете трансакциони лог, подразумевано подешавање дефинише да се уклањају сви непотребни подаци из лога након бекаповања. Ово је сасвим

прихватљиво, осим ако не бекапујете реп трансакционог лога у циљу опоравка оштећене базе података.

13. Уколико користите магнетне траке, можете да изаберете опције везане за избацивање или премотавање траке.
14. Уколико користите Enterprise издање SQL сервера, имате могућност и да компресујете бекапове. Можете да изаберете да ли ћете да користите компресију или ћете прихватити подразумевана подешавања везана за бекаповање.
15. Да бисте прегледали Transact-SQL наредбе које се користе приликом креирања бекапа, можете да употребите Script тастер, који се налази у горњем делу Back Up Database дијалога. Уколико више волите да извршавате одговарајуће Transact-SQL скриптове, обавезно притисните Cancel тастер, како бисте напустили Back Up Database дијалог.
16. Притисните OK тастер, како бисте започели бекаповање.

Парцијални бекапови нису подржани у SQL Server Management Studio корисничком интерфејсу. Да бисте извршили парцијални бекап, морате да искористите одговарајуће Transact-SQL наредбе.

4.3.5. Извршавање потпуног опоравка

Код извршавања потпуног опоравка базе података, процес ће се разликовати зависно од тога који модел опоравка је изабран за посматрану базу података.

Коришћење Simple модела опоравка

Код база података за које је дефинисан Simple модел опоравка, на располагању је много мање опција него код база података за које се примењује Full или Bulk-logged модел опоравка. Могу се користити потпуни и диференцијални бекапови, али су велике шансе да су све промене извршене након последњег бекапа заувек изгубљене.
Simple модел опоравка може да буде веома добро решење у ситуацијама у којима су подаци мање критични, само се очитавају или могу веома једноставно да се поново креирају.
Пример бекаповања и рестаурирања базе података на основу потпуног бекапа базе података, приказан је у листингу 4.5.

Листинг 4.5. Извршавање опоравка базе података на основу потпуног бекапа коришћењем Simple модела опоравка
USE Master; GO ALTER DATABASE TestDB SET RECOVERY SIMPLE; GO BACKUP DATABASE TestDB TO DISK = ‘C:\Backups\TestDB_Full_Example.bak’ ; Go RESTORE DATABASE TestDB FROM DISK = ‘C:\Backups\TestDB_Full_Example.bak’ WITH RECOVERY;

Код приказан у листингу 4.5. дефинише модел опоравка за TestDB базу података, извршава потпуни бекап, а затим рестаурира потпуни бекап.

База података се у потпуности опоравља након извршене операције зато што се приликом опоравка користи само један потпуни бекап.

Диференцијални бекапови се врло често користе у циљу убрзавања процеса бекаповања и опоравка, нарочито уколико се примењује Simple модел опоравка.

Пример опоравка коришћењем диференцијалних бекапова, приказан је у листингу 4.6. Пошто након последњег потпуног бекапа базе података постоје и други бекапови који се могу користити приликом опоравка, наведена је NORECOVERY опција.

Листинг 4.6. Коришћење диференцијалних бекапова приликом опоравка базе података, уз коришћење Simple модела опоравка
USE Master; GO ALTER DATABASE TestDB SET RECOVERY SIMPLE; GO BACKUP DATABASE TestDB TO DISK = ‘C:\Backups\TestDB_Diff_Example.bak’ WITH INIT; GO BACKUP DATABASE TestDB TO DISK = ‘C:\Backups\TestDB_Diff_Example.bak’ WITH DIFFERENTIAL; GO RESTORE DATABASE TestDB FROM DISK = ‘C:\Backups\TestDB_Diff_Example.bak’

```
WITH FILE=1,  NORECOVERY;
GO
RESTORE DATABASE TestDB
FROM DISK = ‘C:\Backups\TestDB_Diff_Example.bak’
WITH FILE=2,  RECOVERY;
```

Коришћење Full модела опоравка

Базе података за које се користи Full модел опоравка, захтевају бекаповање и трансакционог лога.
У листингу 4.7. представљен је пример који приказује случај у коме се користи Full модел опоравка и бекапује трансакциони лог.

Листинг 4.7. Опоравак базе података и трансакционог лога коришћењем Full модела опоравка

```
USE Master;
GO

ALTER DATABASE TestDB SET RECOVERY FULL;
GO
BACKUP DATABASE TestDB
TO DISK = ‘C:\Backups\TestDB_Full_Demo1.bak’;
GO
BACKUP LOG TestDB
TO DISK = ‘C:\ Backups\TestDB_Full_Demo1.bak’;
- - Јавља се проблем приликом функционисања базе података
-- Детектује се реп трансакционог лога
BACKUP LOG TestDB
TO DISK = ‘C:\ Backups\TestDB_Full_Demo1.bak’
WITH NORECOVERY;
GO
-- Почетак процеса опоравка система
RESTORE DATABASE TestDB
FROM DISK = ‘C:\ Backups\TestDB_Full_Demo1.bak’
WITH FILE=1, NORECOVERY;

RESTORE LOG TestDB
FROM DISK = ‘C:\ Backups\TestDB_Full_Demo1.bak’
WITH FILE=2, NORECOVERY;

RESTORE LOG TestDB
```

```
FROM DISK = 'C:\ Backups\TestDB_Full_Demo1.bak'
WITH FILE=3, RECOVERY;
```

Број операција рестаурирања које су неопходне приликом опоравка базе података може се смањити коришћењем диференцијалних бекапова. Овим коришћењем могу се прескочити бекапови логова који су се јавили пре диференцијалног бекапа.
Пример стратегије бекаповања у којој се примењује потпуни бекап, диференцијални бекап и бекап трансакционог лога, приказан је у листингу 4.8.

Листинг 4.8. Коришћење диференцијалних бекапова приликом примене Full модела опоравка

```
USE Master;
GO

ALTER DATABASE TestDB SET RECOVERY FULL;
BACKUP DATABASE TestDB
TO DISK = 'C:\Backups\TestDB_Full_Demo2.bak'
WITH INIT;
GO
BACKUP LOG TestDB
TO DISK = 'C:\Backups\TestDB_Full_Demo2.bak';
GO
BACKUP LOG TestDB
TO DISK = 'C:\Backups\TestDB_Full_Demo2.bak';
GO
BACKUP DATABASE TestDB
TO DISK = 'C:\Backups\TestDB_Full_Demo2.bak'
WITH DIFFERENTIAL;

- - Јавља се проблем приликом функционисања базе података –

-- Детектује се реп трансакционог лога
BACKUP LOG TestDB
TO DISK = 'C:\Backups\TestDB_Full_Demo2.bak'
WITH NORECOVERY;
GO
- - Опоравак коришћењем потпуног бекапа
RESTORE DATABASE TestDB
FROM DISK = 'C:\Backups\TestDB_Full_Demo2.bak'
```

```
WITH FILE=1, NORECOVERY;
- - Опоравак коришћењем диференцијалног бекапа, прескакање оба бекапа логова
RESTORE DATABASE TestDB
FROM DISK = ‘C:\Backups\TestDB_Full_Demo2.bak’
WITH FILE=4, NORECOVERY;
-- Опоравак коришћењем репа лога
RESTORE LOG TestDB
FROM DISK = ‘C:\Backups\TestDB_Full_Demo2.bak’
WITH FILE=5, RECOVERY;
```

4.3.6. Извршавање Point in Time опоравка

База података, за коју се користи Full модел опоравка, може се повратити у стање које дефинише специфични лог маркер или у стање које је било у одређеном тренутку у прошлости. То се постиже рестаурирањем дела бекапа трансакционог лога, али све модификације које су извршене након посматраног тренутка остају заувек изгубљене. У случају да постоје одговарајући бекапови лога, база података се може повратити у стање које је дефинисано, на следећи начин:

- Специфичан тренутак у времену, дефинисан бекапом трансакционог лога
- Пре маркера лога
- Временски тренутак који је дефинисан тренутном позицијом маркера лога

Повратак на стање дефинисано у одређеном тренутку у прошлости

Враћање базе података у стање у коме је била у одређеном тренутку у времену обавља се коришћењем следеће синтаксе:

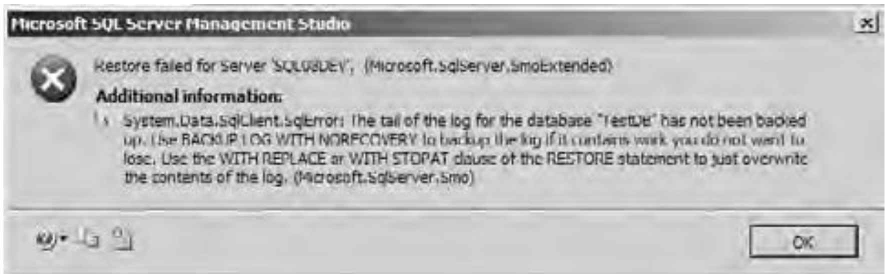
```
RESTORE LOG <database_name>
FROM <backup_device>
WITH STOPAT = <datetime>, RECOVERY
```

4.3.7. Пример коришћења SQL Server Management Studio алата приликом опоравка базе података

SQL Server Management Studio садржи веома моћан интерфејс за рестаурирање база података, група датотека и трансакционих логова.

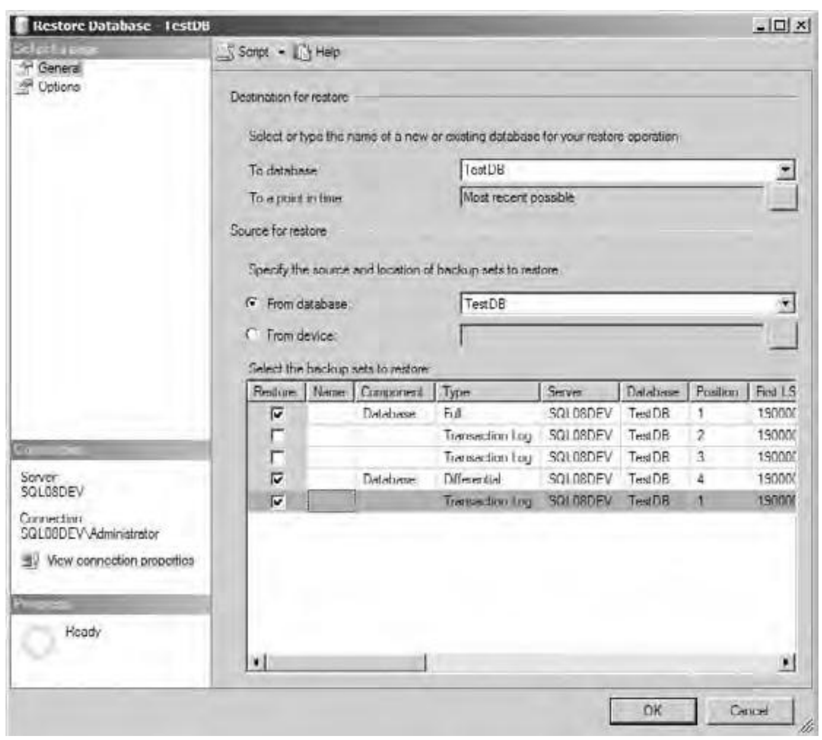
Да би се извршио опоравак неке базе података коришћењем SQL Server Management Studio алата, неопходно је урадити следеће:

1. Уколико је то могуће, бекапујте реп трансакционог лога пре него што започнете процес рестаурирања. Уколико бекапујете активни део трансакционог лога, приказује се порука о појави грешке, као што је приказано на Слици 4.22, која вас обавештава да треба да извршите бекаповање трансакционог лога.



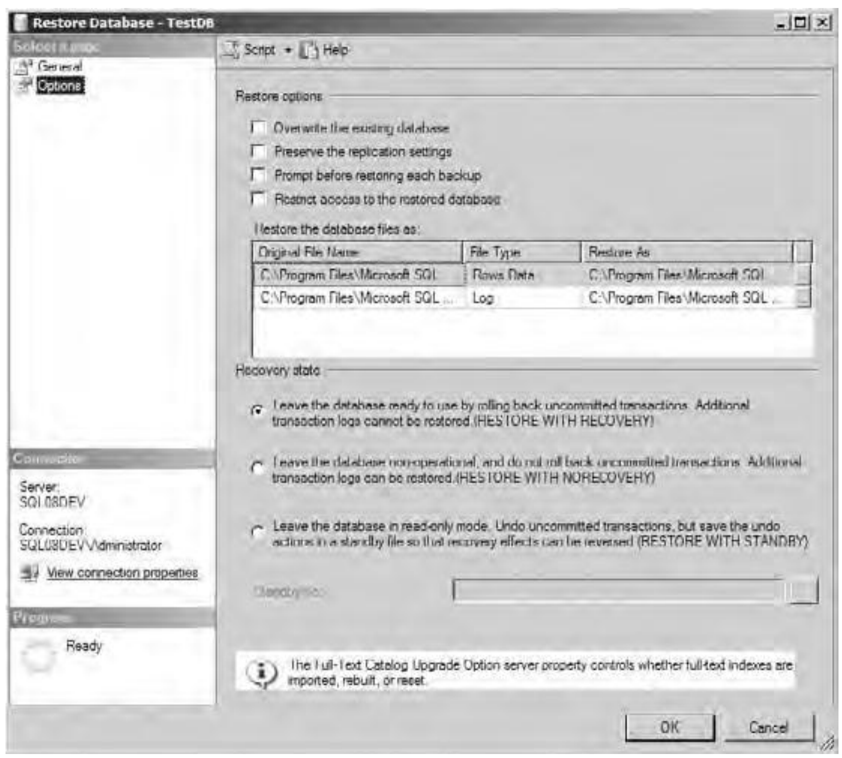
Слика 4.22. Дијалог са поруком о појави грешке приказује се уколико заборавите да бекапујете реп трансакционог лога пре рестаурирања базе података [4]

2. Отворите SQL Server Management Studio и повежите се са SQL Server инстанцом како бисте приступили рестаурирању базе података.
3. Десним тастером миша селектујте Databases директоријум, а затим из контекстног менија изаберите Restore Database ставку. Отвара се Restore Database дијалог.
4. Изаберите постојећу базу података, или унесите нову базу података коју желите да рестаурирате.
5. Уколико желите да рестаурирате базу података на основу претходно извршених бекапова, неопходно је да селектујете базу података из падајуће листе. Листа у доњем делу екрана приказује постојеће скупове бекапова, који се могу користити приликом опоравка базе података, као што је приказано на слици 4.23.



Слика 4.23. Restore Database дијалог са скуповима бекапова које можете користити приликом опоравка базе података [4]

6. Уколико за опоравак базе података желите да користите специфичан уређај за бекаповање, селектујте радио тастер који се налази поред From Device обележја. Приступите жељеном уређају на коме се налази бекап који желите да користите приликом рестаурирања базе података.
7. Изаберите бекапове које желите да користите приликом рестаурирања, селектовањем поља за потврду која се налазе поред појединачних бекапова у доњем делу дијалога.
8. Уколико желите да повратите стање у коме се база података налазила у одређеном тренутку у времену, неопходно је да притиснете тастер који се налази поред ″To a Point in Time″ обележја, а затим изаберите датум и време на основу кога се дефинише жељено стање базе података. Да бисте опоравак извршили коришћењем маркера логова, неопходно је да користите Transact-SQL наредбе.
9. Изаберите Options страницу како бисте проверили да ли су опције везане за опоравак базе података исправне. Селектујте одговарајуће опције везане за опоравак базе података, локације датотека базе података, односно стање опоравка, као што је приказано на Слици 4.24.



Слика 4.24. Страница са опцијама у Restore Database дијалогу [4]

10. Уколико желите да прегледате скрипт који се користи приликом рестаурирања базе података, неопходно је да притиснете тастер Script, који се налази у горњем делу странице, а затим притиснете Cancel тастер. У супротном, притисните OK тастер да бисте започели процес рестаурирања базе података.

Процес рестаурирања датотека и група датотека је врло сличан процесу који је раније описан, осим што се у кораку 3, уместо Restore Database ставке контекстног менија, селектује Restore Files and File-groups stavka.

5. Сигурност базе података

Под сигурношћу базе података подразумева се заштита базе од неовлашћеног приступа. Постоји мноштво различитих разлога због којих је неопходно заштитити базу података:

- Законски прописи, друштвени односи и етичке норме налажу да се спречи неовлашћено коришћење многих података;
- Многи подаци у државној управи, предузећима и другим институцијама морају бити заштићени;
- У самом СУБП, неки подаци, на пример подаци у каталогу, доступни су само администратору базе и неким специфичним апликацијама. Подаци о идентификацији корисника и њиховим лозинкама, такође морају бити заштићени.

Овде ће се прво увести неки основни концепти сигурности база података, а затим ће се детаљније приказати најзначајнији механизам сигурности, дискрециони механизам сигурности.

5.1. Основни концепти

Сигурност целокупног система. Један од проблема сигурности је и заштита целокупног система, односно спречавање неовлашћеној особи да приступи систему било са циљем да добије неке информације или да оштети део или целу базу података. Сигурносни механизам СУБП мора да спречи овакав приступ. То се остварује додељивањем корисничких идентификација и њима одговарајућих лозинки. Проблем сигурности такође обухвата и физичку заштиту система, односно спречавање физичког приступа неовлашћеним особама месту са опремом на којој се чува одговарајућа база.

Модел сигурности базе података. Основни модел сигурности базе података се може дефинисати преко скупа **ауторизација** односно скупа четворки,

Ауторизација: <корисник, објекат, операција, услов>

Ауторизација исказује којим објектима базе посматрани корисник може да приступи, које операције над њим може да изврши и под којим условима. Поред основног, постоје и други модели сигурности који ће се укратко описати. СУБП треба да омогући да се модел сигурности ускладишти и да се операције над базом података обављају на основу дефинисаног модела. Део СУБП који обавља овај задатак назива се **подсистем сигурности (*security subsystem*)** или **подсистем ауторизације (*autorization subsystem*)**.

Дискрециони механизам сигурности (Discretionary mechanism) подржава основни модел у коме се појединим корисницима дају специфична права приступа (привилегије) за различите објекте.

Механизам овлашћења (mandatory mechanism) подржава специфичан модел сигурности у коме је, с једне стране сваком објекту базе података додељен неки степен ″тајности″ (на пример, државна тајна, строго поверљиво, поверљиво, јавно), а са друге корисницима дато право да приступе објектима дефинисаног степена тајности. Подразумева се ″хијерархија овлашћења″, корисник са овлашћењем вишег степена, може да приступи и свим објектима нижег степена тајности.

Статистичке базе података се, са тачке гледишта сигурности, посебно анализирају. Статистичке базе података дају статистичке, збирне информације о феноменима за групу ентитета дефинисане по неком критеријуму (популацију). На пример, нека статистика становништва даје неке збирне податке о популацијама становништва дефинисане на основу старости, прихода, нивоа образовања и слично. Механизам сигурности статистичких база мора да онемогући да се из збирних података извуку закључци о вредностима појединачних података за неки ентитет. То се најчешће постиже спречавањем добијања статистичких информација за популацију мању од неког унапред задатог броја и/или спречавањем извршења секвенце упита над истом популацијом. Проблем сигурности статистичких база података добијаће на значају са већим коришћењем ″стоваришта података″ (data warehouse).

Енкрипција података. Да би се осигурали посебно осетљиви подаци у бази података примењује се енкрипција података, специфичан начин кодирања података који је веома тешко ″разбити″. О алгоритмима енкрипције се овде неће детаљније говорити.

Регистар и ревизија корисникових акција (Audit Trail). Неопходно је претпоставити да системи сигурности базе података није савршен. То значи да је могуће да он понекад буде ″разбијен″. Због тога је неопходно водити посебан лог који се назива ″Регистар корисникових акција″ и повремено вршити преглед (ревизију) овог регистра. Запис у регистру садржи: (1) изворни текст корисниковог захтева, (2) идентификатор терминала са којег је захтев постављен, (3) идентификатор корисника који је поставио захтев, (4) датум и време, (5) релације, n-торке и атрибути којима је приступљено, (6) нову и стару вредност.

5.2. Дискрециони механизам сигурности

SQL стандард подржава само дискрециони механизам сигурности преко наредбе GRANT за креирање ауторизације и наредбе REVOKE којом се ауторизација укида. Синтакса GRANT наредбе је:

```
GRANT < lista privilegija >
      ON <objekat baze>
      TO < lista identifikatora korisnika>
      [WITH GRANT OPTION];
```

Објашњења:

- Привилегије могу да буду: USAGE, SELECT, UPDATE, DELETE, REFERENCES, ALL PRIVILEGES. Привилегија USAGE се даје за коришћење неког специфичног, корисничког (дефинисаног преко SQL-a) домена. Привилегија REFERENCES дозвољава реферисање неке табеле при дефинисању правила интегритета, на пример референцијалног интегритета. Остале привилегије су очигледне.
- Објекти базе могу да буду DOMAIN <име домена> и TABLE <име табеле>. Под табелом се не подразумева само базна табела већ и поглед.
- У листи идентификатора корисника наводе се сви корисници којима је исти скуп привилегија над истим објектима дозвољен. Листа идентификатора може бити замењена са кључном речи PUBLIC, чиме се дефинисане привилегије над листом објекта дају свим корисницима система.
- WITH GRANT OPTION исказ дефинише могућност да корисници из листе корисника пренесу ауторизације на неке друге кориснике, односно да сами издају GRANT наредбу.

Ако се упореди синтакса SQL-a са претходно датим општим моделом сигурности базе података, види се да SQL не подржава постављање услова у ауторизацији. Тај недостатак се за табеле може решити претходним дефинисањем погледа са датим условом, над којим се примењује приказани GRANT исказ.

Привилегија дата неком кориснику се може опозвати коришћењем наредбе REVOKE која има следећу синтаксу:

```
REVOKE [GRANT OPTION FOR]  <lista privilegija>
                           ON <objekat baze>
                           FROM <lista identifikatora korisnika>
                           <opcija>;
```

Објашњења:

- Опција GRANT OPTION FOR се користи када се опозива могућност преноса привилегија.
- <опција> може да има вредност RESTRICT или CASCADES и дефинише начин на који се поступа са пренетим привилегијама за привилегије које се опозивају. Опција RESTRICT онемогућава опозив привилегије ако је она пренета другим корисницима. При опцији CASCADES опозивају се, поред посматраних, и све пренете привилегије.
- Остали делови наредбе REVOKE имају исто значење као и у наредби GRANT.

Напоменимо да власник шеме базе података (корисник који је креирао шему) има све привилегије у њој. Поред тога што може да креира и избацује и мења табеле, погледе, домене и тврдње (assertion), власник може да извршава и све операције над табелама и погледима у шеми. Власник даље преноси привилегије на кориснике, а ови могу тај ланац давања привилегија да наставе, ако су за то овлашћени. Процес давања и опозивања привилегија приказаћемо на следећем примеру:

Нека је Милош власник шеме СтудентскиИС која има следеће табеле:

Студент (<u>БИ</u>, Име, Старост, Семестар, Смер)

Предмет (<u>ШПред</u>, НазивПред, БројЧас)

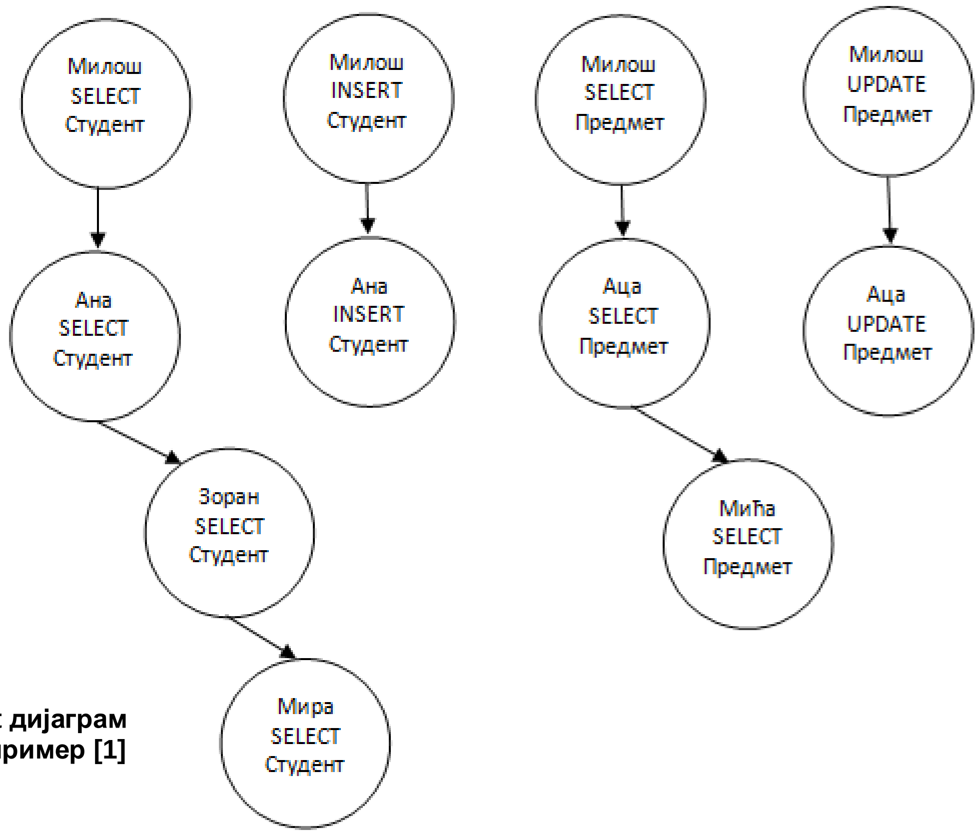
Милош даје привилегију SELECT и INSERT Ани за табелу Студент, а Аци привилегије SELECT и UPDATE за табелу Предмет, дозвољавајући им да те привилегије пренесу и на друге кориснике.

```
GRANT SELECT, INSERT ON Student TO Ana
        WITH GRANT OPTION;
GRANT SELECT, UPDATE ON Predmet TO Aca;
        WITH GRANT OPTION;
```

Ана преноси привилегију SELECT за табеле Студент на Зорана са опцијом преноса на друге, а овај преноси привилегију SELECT за табелу Студент на Миру. Аца преноси привилегију SELECT за табелу Предмет на Мићу.

```
GRANT SELECT Student TO Zoran
      WITH GRANT OPTION;
GRANT SELECT Student TO Mira;
GRANT SELECT Predmet TO Mića;
```

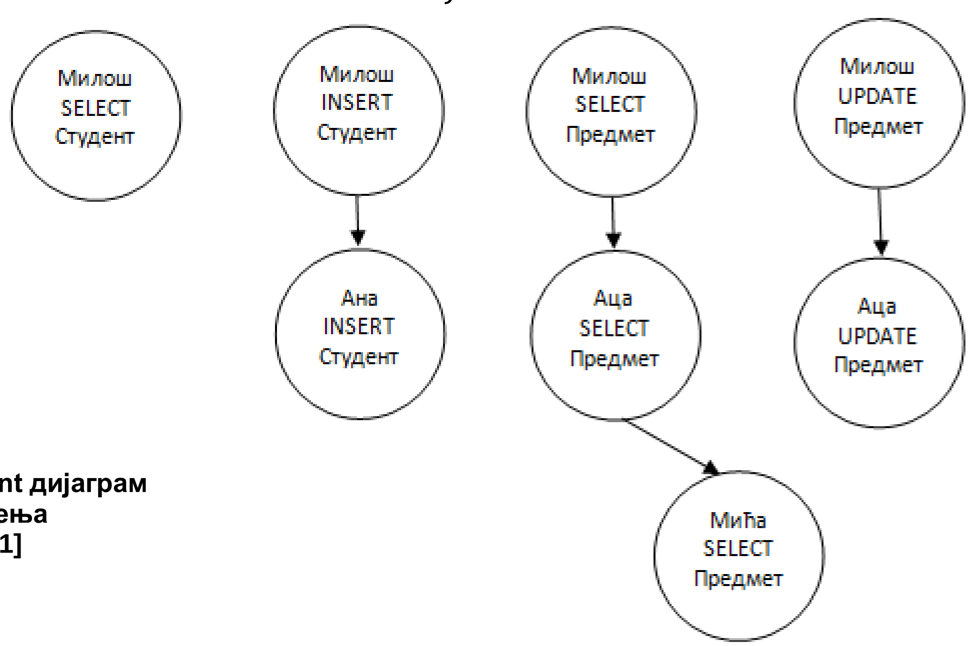
Давање и пренос привилегија приказаћемо преко тзв. ″Grant″ дијаграма (графа) на Слици 5.1. Чвор овога графа представља једну привилегију корисника, а усмерена грана приказује пренос привилегија на другог корисника.



Слика 5.1. Grant дијаграм за приказани пример [1]

На Слици 5.2. приказан је Grant дијаграм за исти пример, после повлачења привилегија Зорану и Ани за SELECT Студент и Аци за SELECT Предмет преко наредби:

```
REVOKE GRANT SELECT ON Student
       FROM Ana
       CASCADES;
REVOKE GRANT SELECT ON Predmet
       FROM Aca
       RESTRICT;
```



Слика 5.2. Grant дијаграм после повлачења привилегија [1]

Као што се са Слике 5.2 види, опција CASCADES укида и све привилегије које је Ана пренела, док опција RESTRICT не дозвољава укидање привилегије Аци за SELECT ON Предмет због тога што је он ту привилегију пренео Мићи.

На крају треба напоменути да и поред веома софистицираних мера заштите сам СУБП не може да у потпуности заштити податке. На пример, пошто се неки СУБП заснивају на фајл систему оперативног система, могуће је подацима у бази приступити и преко овога система заобилазећи механизме заштите података. Због тога је неопходно да се механизми СУБП-а остваре у кооперацији са свим оперативним системима над којима раде.

6. Корисници СУБП-а

Корисници СУБП-а су:

- **Корисници готових апликација** – људи који нису информатички стручњаци и позивају готове програме, при чему ти програми улазе у интеракцију са СУБП-ом.
- **Апликативни програмери** су информатички професионалци који комуницирају са СУБП-ом преко DML позива, који су уграђени (embedded) у језик домаћин (host language), нпр. у UML, C++, Visual Basic. DML прекомпајлер конвертује DML исказе у позиве процедура језика домаћина. Након оваквог препроцесирања, преводилац host језика може да генерише преведени (object) код.
- **Интерактивни корисници** формулишу своје захтеве у упитном језику, нпр. SQL-у. Процесор упита преводи исказе упитног језика у инструкције нижег нивоа које су разумљиве за менаџера базе података (Database Manager).
- **Администратор базе података** је особа која и управља подацима који чине базу, али и програмима који приступају тим подацима.

7. Системи база података нове генерације

Развој система база података у постреляционом периоду кретао се у два правца: надградња објектно оријентисаних програмских језика управљачем трајних (перзистентних) објеката (перзистентни ОО системи), и изградња објектно оријентисаних база података са пуном подршком ANSI SQL језику.

ООСУБП прве врсте обично обезбеђују разне функције стандардног СУБП – једноставне упитне језике, приступне методе као што је директни приступ или груписање, управљање трансакцијама и контролу конкурентности и опоравка.

Међутим, они су некомпатибилни са РСУБП и не поседују читав низ својстава утемељених и развијених у контексту РСУБП, као што су механизам погледа, ауторизација, потпуно непроцедурални упитни језици, управљање каталозима, итд. Због тога се други правац сматра правцем развоја система база података нове генерације.

Интегрисани релационо/објектни системи су проширења релационих система концептима језгра ОО модела, као и неким специфичним ОО концептима.

Оваквим интеграцијама приступају и произвођачи перзистентних ОО система, и произвођачи релационих производа. Први планирају проширења својих система упитним језиком компатибилним са ANSI SQL, док други проширују ANSI SQL2 концептима објектног модела (такво проширење је и најављени ANSI SQL3 стандард).
И поред овакве начелне сагласности у погледу развоја нове генерације система база података, постоје и значајне разлике међу заинтересованим истраживачким и произвођачким странама. Тако се и догодило да су се у периоду од 1990-1995. године у релевантној литератури појавила три манифеста система база података нове генерације, које су саставили врхунски истраживачи и експерти у области ОО и релационе технологије. Први од тих манифеста заступа концепте објектног модела са управљачем перзистентних објеката, сматрајући га следбеником релационог модела у еволуцији модела података, у истом смислу одбацивања и негирања концепата у коме је релациони модел следбеник мрежног и хијерархијског.

Други манифест је дело Комитета за унапређење функција СУБП, састављеног од најеминентнијих истраживача у области релационе технологије, са америчких универзитета и из индустрије. Он полази од чињенице да РСУБП не само да не подржава апликације са комплексним подацима, већ да не подржава адекватно ни пословне апликације којима је првенствено намењен; адекватна подршка пословним апликацијама подразумева и подршку комплексним документима, као и мултимедијалним подацима – слици, рукопису, звуку, фотографији. Основне поставке овог манифеста су следеће.

- Поред традиционалног управљања, нова генерација СУБП треба да подржи богатију структуру објеката и правила. Пожељна својства су: систем апстрактних типова података, разноврсни конструктори типова података (низ, слог, скуп, унија), рекурзивна композиција конструктора, наслеђивање, учаурење, декларативна ограничења и процедурални тригери.
- Нова генерација СУБП мора да наследи сва својства релационе генерације СУБП, пре свега непроцедурални приступ подацима, независност података и механизам погледа.
- Нова генерација СУБП мора бити отворена према другим подсистемима, тј. треба да омогући лак приступ из других система као што су системи раширених табела, програмски језици, алати за подршку одлучивању, графички пакети; она такође треба да омогући рад у клијент/сервер архитектури, као и у дистрибуираном окружењу. Као сумеђу система нове генерације треба развијати перзистентне језике (који омогућавају рад са базом) за разноврсне програмске језике. За интерактивне упите треба користити SQL, тј. његове надградње, јер је упркос својим бројним недостацима тај језик освојио тржиште.

Трећи у овом низу манифеста полази од релационог модела у његовом примарном облику, а не од релационих система или релационих језика, и поготову не од SQL-а, с обзиром на његова значајна одступања од самог релационог модела. Релациони модел се не проширује, већ се предлажу системи који, осим добрих својстава релационог модела, поседују и добра ортогонална својства (друга добра својства која релациони модел не познаје); та ортогонална својства су углавном добра својства објектно оријентисаног модела. Такође, наводе се својства постојећих модела која се оцењују као лоша, и која ови системи треба да искључе.

За разлику од низа имплементација интегрисаних релационо/објектних система, који интеграцију базирају на изједначавању релационог концепта релације и објектног концепта класе, приступ који се предлаже овим манифестом полази од изједначавања концепта класе са релационим концептом домена. Тиме се превазилази низ тешкоћа као што су изражавање *ad hoc* упита над објектном базом, реализација учаурења структуре и понашања објекта, карактеризација типа, пренето ажурирање објеката, одржавање правила интегритета, семантика операција релационе алгебре, итд.

Међу добрим својствима релационог модела, која треба подржати у системима база података нове генерације, истичу се следећа:

- домен као именовани скуп вредности произвољне сложености; вредности из домена може се приступити само преко операција дефинисаних над доменом (вредност домена, а не n-торка релације, одговара објекту у објектно оријентисаном моделу, па је вредност домена учаурена);
- скуп n-арних операција за сваку уређену листу од n домена (не обавезно различитих), са назнаком домена резултата операције;
- физичка репрезентација различитих вредности једног домена може бити различита (нпр. два природнојезичка документа могу бити приређена различитим процесорима текста);
- за одређену структуру n-торке (атрибуте и домене), могуће је дефинисати два нова домена – један чије су вредности n-торке са задатом структуром, и други чије су вредности релације са задатом структуром; дакле, ″n-торка″ и ″релација″ су конструктори домена;
- услов интегритета може бити општег облика формуле релационог рачуна, и може се односити на домен, атрибут, релацију или базу података у целини.

Лоша својства релационих система, која се не смеју подржати системом нове генерације (према овом манифесту), односе се, пре свега, на својства SQL језика којима он одступа од релационог модела (нпр. позивање на редослед атрибута релације, неименовани атрибути, дуплирање n-торки, присуство физичке репрезентације и приступних путева на логичком нивоу, слоговно оријентисан приступ, брисање и ажурирање коришћењем курсора, итд).

Међу новим својствима која системи нове генерације треба да поседују истичу се провера типова у време компилације, прецизни модел наслеђивања (једноструког или вишеструког), угњеждене трансакције. Као лоша својства објектних система препознају се изједначавање релације и класе објеката, постојање идентификатора објеката, концепт јавних и заштићених инстанцних променљивих, итд. Зато системи база података нове генерације не треба да подрже ова својства.

8. Закључак

Системи управљања базом података обично се категоришу према моделу података који подржавају: односни, оријентисани према објекту, мрежни итд. Велики део интерног инжењерства СУБП-а је независан од модела података, те је заокупљен управљањем факторима попут перформанси, подударности, интегритета и обнове након хардверских попуста. У овим областима постоје велике разлике међу производима.
Назив ″база података″ се строго односи на збирку записа, а на софтвер би се требало односити као на систем управљања базом података или СУБП. Када је контекст недвосмислен, многи администратори за базе података и програмери ипак користе термин ″база података″ да покрију оба значења. Многи професионалци ће сматрати да збирка података ствара базу података једино ако има одређена својства: на пример, ако се подацима управља како би осигурали свој интегритет и квалитет, ако омогућује заједнички приступ некој заједници корисника, ако има шему, или ако подржава упитни језик. Ипак договорена дефиниција ових својстава не постоји.

ЛИТЕРАТУРА

1. Лазаревић Б. ….: ***Базе података***, ФОН Београд, Београд 2003.
2. Павловић-Лажетић Г.: ***Основе релационих база података***, Математички факултет, Београд, 2000.
3. Ross Mistry, Stacia Misner: ***Introducing Microsoft SQL Server 2008***, Microsoft Press, Washington, 2010.
4. Michael Lee, Gentry Bieker: ***Mastering SQL Server 2008***, Wiley Publishing, Inc., Indianapolis, Indiana, 2009.

www:

1. http://www.vtsnis.edu.rs/Predmeti/baze_podataka/BAZE%20PREDAVANJA%202%20-%20SUBP.pdf
2. http://www.ef.uns.ac.rs/Download/bazepodataka/29-05-08%20predavanja%20subp.pdf
3. http://www.singipedia.com/content/1100-Uvod-u-baze-podataka
4. http://www.fer.unizg.hr/_download/repository/BazePodataka_SQLPredavanja.pdf