UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

PRIMERJALNA ANALIZA ARHITEKTUR SISTEMOV PODATKOVNE ANALITIKE

Ljubljana, september 2016 DARKO JAGARINEC

IZJAVA O AVTORSTVU

Podpisani Darko Jagarinec, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Primerjalna analiza arhitektur sistemov podatkovne analitike, pripravljenega v sodelovanju s svetovalcem prof. dr. Jurijem Jakličem,

IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;

7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne 12. septembra 2016 Podpis študenta:

KAZALO

UVOD **1**
1 PODATKOVNA ANALITIKA IN PODATKOVNO SKLADIŠČE **6**
1.1 Transakcijski sistem 6
1.2 Analitično procesiranje 8
1.3 Vrste arhitektur sistemov podatkovne analitike in njihov namen ter uporaba 9
1.3.1 Centralizirano podatkovno skladišče 13
1.3.2 Distribuirano podatkovno skladišče 14
1.3.3 Federativno podatkovno skladišče 15
1.3.4 Pomnilniška arhitektura sistema podatkovne analitike 17
1.4 Planiranje in potek razvoja podatkovne analitike 28
2 PREDSTAVITEV PROBLEMATIKE **31**
2.1 Faze izdelave sistema podatkovne analitike 31
2.1.1 Definicija poslovnih zahtev 32
2.1.2 Analiza vseh virov 34
2.1.3 Standardizacija in čiščenje podatkov 36
2.1.4 Normalizacija tabel 37
2.1.5 Proces ETL 37
2.1.6 Metapodatki 39
2.2 Napake in problemi pri izvedbah 40
2.3 Izbira vrste arhitekture sistema podatkovne analitike 50
3 PREGLED IZBRANIH IZVEDENK ARHITEKTUR **52**
3.1 Tradicionalna arhitektura sistema podatkovne analitike 53
3.1.1 Omejitve tradicionalnih podatkovnih baz 54
3.1.2 Primer arhitekture z enim strežnikom 54
3.1.3 Skalabilnost 59
3.2 Arhitektura sistema podatkovne analitike z uporabo pomnilniške podatkovne baze 60
3.2.1 Omejitve pomnilniških podatkovnih baz 60
3.2.2 Primeri sistemov za pomnilniške podatkovne baze 61
3.2.3 Skalabilnost 66
3.3 Arhitektura sistema podatkovne analitike z distribuirano podatkovno bazo NoSQL 66
3.3.1 Omejitve NoSQL podatkovnih baz 66
3.3.2 Primer NoSQL podatkovnih baz 68
3.3.3 Skalabilnost 69
4 PRIMERJALNA ANALIZA IZVEDENK ARHITEKTUR **72**
4.1 Povezave in soodvisnosti določenih arhitektur 72
4.2 Način primerjave med posameznimi arhitekturami 72
4.3 Kriteriji za izbiro izvedenke arhitekture in izbira kriterijev 73
4.4 Primerjalna analiza ter njen pomen za odločanje 74
SKLEP **78**
LITERATURA IN VIRI **80**

KAZALO SLIK

Slika 1: Shema transakcijskega sistema 6
Slika 2: Entiteta oziroma entitetni tip 7
Slika 3: Atribut 7
Slika 4: Povezava med entitetama 8
Slika 5: Arhitektura podatkovnega skladišča za velike organizacije 10
Slika 6: Arhitektura podatkovnih področij 11
Slika 7: Hub-and-spoke (centralno) področno podatkovno skladišče 11
Slika 8: Arhitektura podatkovnega skladišča z operativno hrambo podatkov za velike organizacije 12
Slika 9: Porazdeljena arhitektura podatkovnega skladišča 12
Slika 10: Arhitektura podatkovnega skladiščenja po Inmonu 14
Slika 11: Pristop podatkovnega skladišča R. Kimballa 15
Slika 12: Federativno oziroma hibridno podatkovno skladišče 16
Slika 13: Visokonivojska arhitektura sistema pomnilniške podatkovne baze 18
Slika 14: Hierarhija pomnilnika 20
Slika 15: Slovar in atributni vektor pri vpisu podatkov 25
Slika 16: SDLC – razvojni cikel programske opreme 29
Slika 17: Vir podatkov – tekstovna datoteka 35
Slika 18: Več virov kot vhodna struktura 35
Slika 19: Enostaven proces ETL 38
Slika 20: Kompleksen proces ETL 39
Slika 21: Primer uspešne transakcije in operacij 56
Slika 22: Primer neuspešne transakcije in operacij 57

KAZALO TABEL

Tabela 1: Rezultat poslovne zahteve za nivo managerjev 33
Tabela 2: Rezultat poslovne zahteve za nivo oddelka analitičnih obdelav 33
Tabela 3: Vir podatkov – tabela 35
Tabela 4: Metapodatki tabele na viru 40
Tabela 5: Predlagana izvedenka arhitekture sistema 75

UVOD

Podjetje za svoje informacijsko in operativno delovanje potrebuje informacijski sistem (v nadaljevanju IS). Transakcijski sistem (angl. *On Line Transaction Program*, v nadaljevanju OLTP) je sistem, ki z aplikacijo (programom) obravnava vhodne podatke in jih glede na poslovni proces tudi shranjuje (Ma, Chou & Yen, 2000, str. 127). Poleg transakcijskega sistema ima podjetje lahko tudi: dokumentni sistem, sistem za elektronsko pošto, sistem za obveščanje in ostale sisteme.

Transakcijski sistem za svoje delovanje potrebuje mesto, kjer se podatki skozi časovno obdobje kopičijo in odlagajo na za to primerno mesto – temu pravimo hramba transakcijskih podatkov. Časovno obdobje hrambe transakcijskih podatkov je odvisno od politike podjetja, lahko je omejeno na kratek čas, recimo mesec do pol leta, lahko pa tudi več let.

Transakcijski sistem vsakega podjetja je arhitekturno zasnovan tako, da je kar najbolj prilagojen (učinkovit) za uporabo, da v najboljši možni meri pokriva vsak poslovni proces in tudi hitro deluje – brez nepotrebnega čakanja za zaključevanje določene akcije procesa transakcijskega sistema. Akcija transakcijskega sistema je lahko zajem podatkov o kupcu, izpisek računa, popravek podatkov o kupcu, storno računa, popravek računa in podobno.

Transakcijski sistemi za svoje delovanje potrebujejo le določen del podatkov, kot so šifranti (kupci, artikli, kratka zgodovina), zato taki sistemi niso primerni za analizo poslovanja iz naslednjih dejstev (Ma et al., 2000):

- ker hranijo prekratko zgodovino podatkov,
- morajo v osnovi delovati hitro,
- so optimizirani in namenjeni večjemu številu uporabnikov,
- delovati morajo po principu 24/7, torej neprestano (spletna trgovina).

Že zgoraj našteti razlogi zadoščajo, da pri obravnavanju večje količine podatkov skozi daljše časovno obdobje s transakcijskim sistemom ne moremo operirati, zato potrebujemo drugačen pristop do reševanja tovrstnih problemov in izzivov.

Za pravilen pristop je potrebno razmisliti o uvedbi podatkovnega skladišča v podjetje, kjer naj bi bilo že v definiciji poskrbljeno za (Marco, 1998; Watson, Goodhue & Wixom, 2002; Gorla, 2003):

- hrambo večje količine podatkov,
- neovirano delovanje transakcijskega sistema,
- neodvisno delovanje od transakcijskega sistema,

- optimizirano delovanje za manjše število uporabnikov in
- izdelavo množice različnih poročil skozi zgodovino delovanja transakcijskega sistema.

Podatkovno skladišče (Marco, 1998) s svojo značilno arhitekturo sistema podatkovne analitike in procesi omogoča podjetju celovit pogled na zgodovino podatkov. Pomembno je dejstvo, da so podatki iz transakcijskega sistema prečiščeni in pretvorjeni v obliko, ki je primerna za njihovo uporabo v podatkovnem skladišču.

Podatkovno skladišče za podjetje pomeni dodano vrednost pri poznavanju, preučevanju in hranjenju podatkov, saj se razlikuje od klasične hrambe podatkov ostalih sistemov. Struktura hranjenja podatkov v podatkovnem skladišču je namreč zelo odvisna od tega, kakšno vrsto podatkovnega skladišča izberemo.

Če postavimo vprašanje drugače – kako bi pravzaprav lahko podjetja v tako hitro spreminjajočih se časih obstajala:

- brez pregleda nad zgodovino poslovanja,
- brez načrtovanja prodaje, nabave, zaloge,
- brez raziskovanja z izdelki slabo pokritega področja,
- brez iskanja tržne niše,
- nezaznavanja padca prodaje in vzrokov za padec prodaje,
- brez segmentiranja kupcev ter
- brez odgovorov na še veliko ostalih analitičnih vprašanj.

Podjetja z zgoraj naštetimi pomanjkljivostmi kmalu ne bodo več konkurenčna (Marco, 1998; Watson et al., 2002; Gorla, 2003), ne bodo dovolj vedela o primerljivih izdelkih in bodo na dolgi rok opešala ali začela slabo poslovati, ne da bi poznala prave vzroke. V takih situacijah je pogosto najbolj znan izrek: »Saj nam je šlo dobro, potem pa ne vemo, kaj se je zgodilo«. Zgodilo pa se jim je to, da niso bili pripravljeni ter niso razmišljali dolgoročno in strateško. Podjetje je lahko sicer uspešno, vendar, če kmalu ne zazna, da se izdelek kategorije »krava molznica« spreminja v »psa«, bo lahko takrat, ko bo to ugotovilo, žal zanj že prepozno – bo pa dobro za konkurenčna podjetja. Zato je podatkovno skladišče ali kakšna druga arhitektura (Arhitecture, 2016) sistema podatkovne analitike nujno potrebno za operativne odločitve, po drugi strani pa tudi za analizo preteklega poslovanja. To v resnici pokaže, kako dobro je podjetju šlo v preteklem obdobju, iz tega pa je možno določiti tudi napoved poslovanja in izdelave bolj natančnega plana. Najslabše planiranje je planiranje »na pamet«, brez kakovostnih in oprijemljivih informacij.

Podatkovno skladišče samo po sebi ni črna škatla, v katero se vpelje več virov podatkov, na izhodu pa dobi samo želene rezultate. Resnica je vse prej kot to. Trud, volja in čas so trije največji zavezniki, ki ekipo strokovnjakov pripeljejo do uniformno določene

podatkovne analitike v podjetju. Sam uspeh je, resnici na ljubo, velikokrat resnično odvisen od kvalitete analize podatkovnega analitike in izbire prave arhitekture sistema podatkovne analitike, saj se na tem mestu pokažejo prave zahteve in pravi namen, zakaj bi podjetje potrebovalo podatkovno skladišče.

V obdobju od začetka razvoja podatkovne analitike se je razvilo več različnih arhitektur sistemov podatkovne analitike, saj so razvoj teh vodila podjetja z zahtevami, ki so od podjetja do podjetja zelo različne. Tako bo podjetje, ki potrebuje samo osnovne podatke o svoji prodaji, izbralo drugačno arhitekturo od podjetja, ki želi, glede na nakupne navade svojih kupcev, napovedati prihodnjo prodajo po različnih regijah. Na področju podatkovne analitike se srečamo z različnimi arhitekturami podatkovne analitike, od katerih vsaka nudi določene prednosti uporabe, nekatere pa za svoje delovanje potrebujejo tudi veliko različnih znanj (končnih) uporabnikov. Druge pa so že toliko intuitivne, da jih lahko prične uporabljati že manj vešč poznavalec informatike. Reševanje problema, kako izbrati pravo arhitekturo sistema podatkovne analitike, sem zasnoval raziskovalno, saj konkretnih raziskav s tega področja še nisem zasledil, nekaj poskusov definicij pa lahko odkrijemo, če raziskujemo različne forume, članke in nasvete na spletu.

Na tem področju še ne obstaja veliko raziskav (Widom, 1995; Wu & Buchmann, 1997; Di Mauro, Nordvik & Lucia, 2006; Gu, Goetschalckx & McGinnis, 2006; Gene, 2015), zato so podjetja pri odločitvi večinoma prepuščena svoji iznajdljivosti. Dobra opora so predvsem priporočila vodilnih strokovnjakov s tega področja in veliko različnih nasvetov, ki že obstajajo na spletu. Raziskave so večinoma v obliki predloga najboljših postopkov uvedbe podatkovnega skladišča v podjetje. Žal se pretežno ne osredotočajo na izbiro arhitekture podatkovnega skladišča, ampak predvsem upoštevajo celotni življenjski cikel – od uvedbe podatkovnega skladišča do njegove večletne uporabe.

Hwang in Capel (2002) sta iz naslova izkušenj velikih podjetij ugotovila, da bi ta morala ob uvedbi podatkovnega skladišča v začetku analize izbire vrste podatkovnega skladišča razjasniti določene akcije, termine in posledice, katerim je potrebno nameniti večji del analize.

Avtorja raziskave Hwang in Capel (2002) sta od tisoč poslanih vprašalnikov dobila samo 26 pravilno izpolnjenih in primernih za analizo. Iz izkušenj teh 26 velikih podjetij iz različnih branž sta ugotovila, da je potrebno preveriti najmanj naslednje **razvojne dejavnike**:

- čas razvoja,
- stroške razvoja,
- starost podatkovnega skladišča,
- velikost podatkov in
- število uporabnikov.

Poleg tega je pomembno, da se za **obvladovanje (vzdrževanje)** podatkovnega skladišča določi najmanj:

- število vzdrževalcev,
- čas osveževanja,
- obravnavanje t.i. »umazanih podatkov«,
- postopke arhiviranja in
- delovanje operativne podatkovne shrambe (angl. *Operational Data Store*, v nadaljevanju ODS).

V vseh točkah, tako razvojnih dejavnikov kot v obvladovanju podatkovnega skladišča, sta avtorja primerjala med seboj dve osnovni (primarni) vrsti podatkovnih skladišč oziroma arhitekturi sistema podatkovne analitike.

Problem pravilne izbire arhitekture podatkovne analitike zaznavamo tudi v literaturi (Sen & Sinha, 2005). V delu bom raziskoval kriterije, na katere mora biti podjetje pozorno, kadar se odloča za uvedbo katere izmed arhitektur podatkovne analitike. Lahko rečem, da sta podpora vodstva in močan sponzor dva osnovna predpogoja, da se lahko začne izvajati analiza uvedbe podatkovnega skladišča ter s tem pripadajoče arhitekture. Problemi, ki spremljajo to področje, so prisotni in prežijo na čisto vsaki točki vpeljave podatkovnega skladišča. Zelo lahkomiselno je tako mišljenje, da se v različnih fazah uvedbe podatkovnega skladišča lahko »skrijejo« napake, ki bi bile narejene v predhodnih fazah. Na žalost namreč velja nepisano pravilo, da pojavitev napake lahko eskalira v zaustavitev projekta, v najboljšem primeru pa v ponoven začetek – analizo, kar pa velikokrat ni zaželeno s strani vodstva in sponzorja. Velikokrat se mešata tudi osnovna termina: vodstvo podjetja in sponzor projekta. To ni nujno en in isti subjekt, saj je vodstvo lahko tudi sponzor, lahko pa je sponzor tudi oddelek marketinga ali prodaje, torej je plačnik podatkovnega skladišča proračun nekega drugega oddelka in ne nujno glavnega vodstva. Velja pa dejstvo, da mora vodstvo vseeno odobriti tak projekt, pa čeprav je sponzor »podrejen« člen v podjetju; konec koncev pa se sredstva črpajo iz istega proračuna.

Namen naloge je prispevati k lažjemu odločanju pri izbiri ustrezne izvedenke arhitekture sistema podatkovne analitike ter ugotoviti, na osnovi katerih kriterijev bo sprejeta odločitev o najprimernejši izvedbi.

Cilj naloge je pregled oziroma primerjalna analiza različnih pristopov oziroma arhitektur sistemov podatkovne analitike, ki naj bi nas vodili do pravilne ali vsaj najbolj ustrezne vrste in izvedbe podatkovnega skladišča v okviru sistemov podatkovne analitike. Cilj dela je tudi prikazati in prispevati k lažjemu odločanju o izbiri arhitekture glede na problematiko npr. pridobivanja podatkov ter razpoložljiv čas, v katerem moramo imeti sistem podatkovne analitike na voljo za končne uporabnike. Pričakujem, da bom tekom

raziskovanja spoznal tudi druge kriterije, ki vplivajo na odločitev izbire arhitekture sistema podatkovne analitike.

Metode dela, ki jih bom pri izdelavi magistrskega dela uporabil, temeljijo predvsem na preučevanju teorije in uspešne prakse izbire arhitekture sistema podatkovne analitike. Obenem bom preučil gradiva in najboljše prakse, katere bi lahko pripomogle k odločitvi, da se za določene primere klasičen ter tradicionalen pristop k izvedbi podatkovnega skladišča oziroma sistema podatkovne analitike zavrže in izbere sodobnejši sistem, ki gre v korak s časom ter potrebami.

Pri izbiri metodološkega dela se bom naslonil na tujo strokovno literaturo, tuje in domače vire, posamezne prispevke vplivnih avtorjev in članke z najnovejšimi teoretičnimi ter praktičnimi spoznanji s področja podatkovnih skladišč in ostalih arhitektur sistemov podatkovne analitike na spletu ter poskušal vključiti tudi svoja praktična znanja, ki sem jih uspel skozi leta pridobiti na različnih projektih v okolju IT.

Magistrsko nalogo bom razdelil na štiri poglavja. V prvem poglavju se bom osredotočil na pomen podatkovnih skladišč in arhitektur. Pri opisu problema, namena in cilja se bom osredotočil predvsem na praktične probleme, ki nastajajo pri izdelavi podatkovnega skladišča, ter probleme, ki samo izdelavo spremljajo od začetka do konca, pa tudi v kasnejšem življenjskem ciklu podatkovnega skladišča (Kimball & Ross, 2002, str. 15). V naslednjem poglavju bom opisal, katere so arhitekture podatkovnih skladišč, zakaj poznamo več vrst podatkovnih skladišč in kakšna je pravzaprav narava uporabe posameznega podatkovnega skladišča. Predstavil bom tudi posamezne faze izdelave podatkovnega skladišča.

V opisu problemov in napak bom analiziral pasti, na katere moramo biti pozorni in se jim izogniti v največji možni meri. Napako lahko naredimo, ne da bi vedeli, da lahko kasneje vpliva na konec izvedbe projekta, še bolj pa na njegovo uporabo. Zato je pomembno, da pravilno postopamo v odločitvi, katero vrsto podatkovnega skladišča oziroma katero arhitekturo podatkovne analitike izberemo. Pri analizi arhitektur bom najprej skušal ugotoviti splošno znana dejstva, skrb pa bom namenil tudi tistim, na katere se vse prevečkrat pozablja in se jih navadno ugotovi šele takrat, ko pride do težav, navadno prepozno. Vse arhitekture bom preučil in izbral kriterije, ki, glede na svojo pomembnost, določajo, katero arhitekturo podatkovne analitike je najbolj smotrno izbrati. Analiziral bom literaturo in že predstavljene raziskave s področja podatkovnih skladišč, sodobnih pristopov hranjenja podatkov v pomnilniku, kot tudi tiste zadnje tehnologije, ki vnašajo velike spremembe v informatiko na področju analitike podatkov, vključil pa bom še nekaj svojih pridobljenih izkušenj.

Pri izbiri arhitektur bom predstavil svoj pogled na problematiko in rešitve v praktičnem delu ter upošteval svoje dvajsetletne izkušnje v informatiki. Še vedno obstajajo arhitekture, ki so kompleksnejše, kot si jih lahko podjetje izdela.

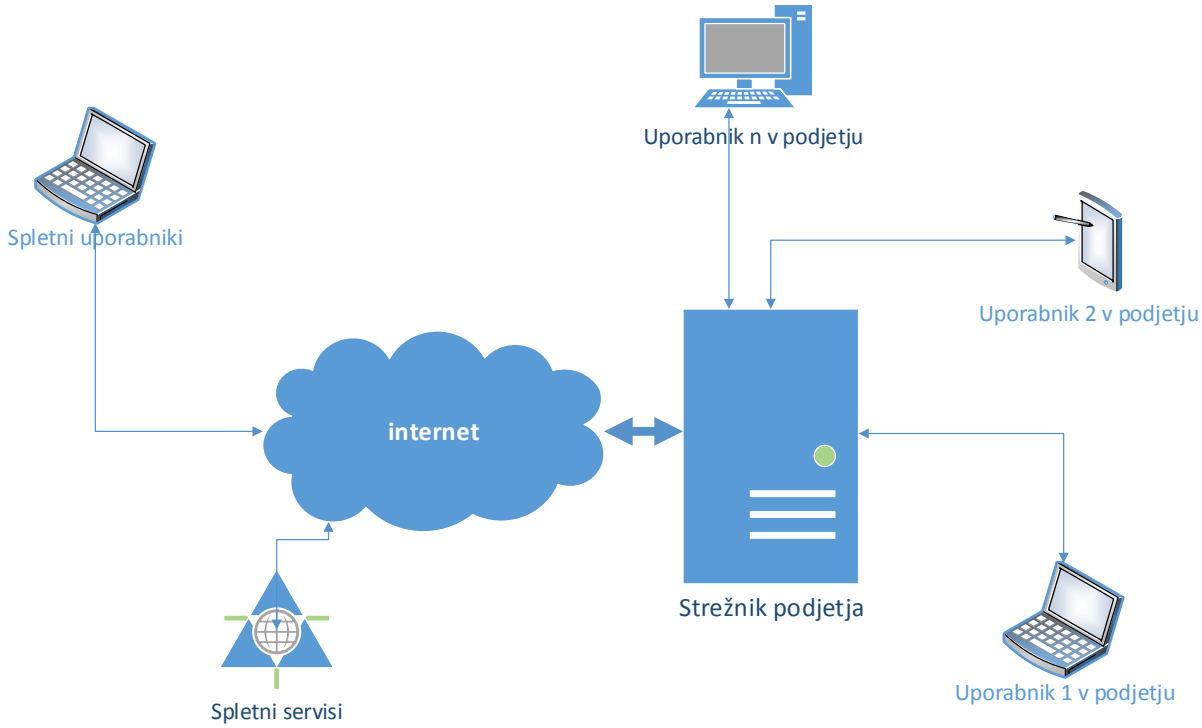
1 PODATKOVNA ANALITIKA IN PODATKOVNO SKLADIŠČE

1.1 Transakcijski sistem

Transakcijski sistemi (OLTP) so implementirani tako, da delujejo zelo hitro in z majhnim obsegom podatkov. Tabele so praviloma pretvorjene v normalizirano obliko, kar je v informatiki označeno kot 3NF (3. normalna forma). V zvezi s podatkovnim modelom je pomembno predvsem, da se podatkovni model sistema OLTP razlikuje od podatkovnega modela podatkovnega ali področnega skladišča.

Na Sliki 1 je prikazan primer arhitekture transakcijskega sistema v podjetjih, pri čemer velja, da ima prikazani transakcijski sistem najmanj dvonivojsko arhitekturo strežnik – odjemalec (angl. *client-server*). Pri tem so uporabniki sistema lahko različni uporabniki z različnimi odjemalci v podjetju, spletni uporabniki, ki niso vnaprej znani ter uporabniki, ki dostopajo do podatkov strežnika sistema preko vnaprej definiranih spletnih servisov. Spletni uporabniki in spletni servisi dostopajo do strežnika podjetja preko interneta.

Slika 1: Shema transakcijskega sistema



Vir: M. Ardales, Types Of Information Systems, 2009, str. 9.

V transakcijskemu sistemu (sistem OLTP) vsebuje podatkovni model štiri osnovne gradnike (Grad & Jaklič, 1996):

- entiteto,
- atribut,
- povezavo in
- ključ.

Entiteta (entitetni tip) je objekt, ki obstaja v svetu modeliranja in se razlikuje od ostalih, primer na Sliki 2, kjer je prikazana entiteta »Kupec«. Vsaki entiteti lahko določimo lastnosti, ki naj bi jih imela. Entiteta je zato osnovni gradnik podatkovnega modela in je označena s pravokotnikom. Imena entitet se v podatkovnem modelu ne smejo ponavljati, saj so unikatna.

Slika 2: Entiteta oziroma entitetni tip

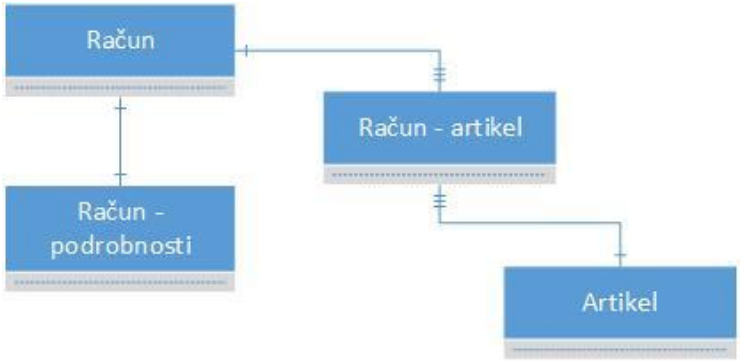
Vsaka entiteta ima svoje lastnosti in lastnosti, ki jih lahko določimo. Slednje imenujemo atributi. Vsak atribut opišemo z njegovim tipom in dolžino, prednastavljeno vrednostjo in obveznostjo vsebine (prazno, neprazno). Atribute zapišemo v pravokotnik pod ime entitete, na Sliki 3 pa sta primera atributov »Ime« in »Priimek«.

Slika 3: Atribut

Ker entitete ne morejo obstajati samostojno, so namreč sestavni del podatkovnega modela, jih povezujemo med seboj. Povezava je vedno definirana med dvema entitetama. Za povezavo je obvezna še definicija njune povezanosti.

Naj spomnim, da v relacijskem podatkovnem modelu fizično obstajata samo povezavi 1:1 in 1:N, povezave N:M (te obstajajo na konceptualnem nivoju) pa moramo razbiti z vpeljavo dodatne entitete kardinalnosti 1:N, kot je prikazana na Sliki 4 med entitetami »Račun«, »Račun – artikel« in »Artikel«.

Slika 4: Povezava med entitetama



Vsak atribut entitete ima določen pomen. Vedno pa ostaja nek atribut, ki enolično določa vse ostale atribute znotraj entitete. Takemu ključu pravimo primarni ključ. Atributa »Ime« in »Priimek« po naravi ne predstavljata unikatno določenih primerkov entitete, medtem ko je atribut »EMŠO« lahko unikaten in neponovljiv – ta atribut je velikokrat najboljši kandidat za primarni ključ.

V podatkovnem ali področnem skladišču je podatkovni model sicer sestavljen iz enakih gradnikov kot transakcijski sistem, vendar entitete niso več v normalizirani obliki, kar pomeni, da niso več povezane med seboj tako, kot morajo biti za hitro delovanje transakcijskega sistema. V tem primeru je podatkovni model denormaliziran, kar pomeni, da se določene entitete združujejo med seboj z namenom, da bi dosegli hitrejše odzivne čase, kadar gre za kompleksnejše poizvedbe v podatkovnem skladišču.

V podatkovnem modelu podatkovnega skladišča sta lahko dodatna dva gradnika, ki sta pomembna za izdelavo in učinkovito delovanje podatkovnega skladišča. To sta dejstvo (angl. *fact*) – formalna struktura, ki vedno predstavlja konkreten števen podatek, in dimenzija (angl. *dimension*) – formalna struktura, ki ima podobno vlogo kot šifrant v transakcijskemu sistemu.

1.2 Analitično procesiranje

OLAP (OLAP, b.l.; Jagarinec, 2005) tehnologija omogoča izvedbe analitičnih obdelav podatkov v sistemih podatkovne analitike in pomeni sprotno procesiranje podatkov (angl. *On-Line Analytical Processing*, v nadaljevanju OLAP). V osnovi opravljajo vlogo orodja, ki preko za v ta namen predpripravljenih podatkovnih struktur (kock) pregledujejo množico podatkov. Vsi podatki so ločeni od transakcijskega sistema (OLTP), značilne pa so še naslednje lastnosti (OLAP, b.l.; Jagarinec, 2005):

- poizvedovanje po podatkih v sistemih OLAP ne obremenjujejo transakcijskega sistema (OLTP),

- OLAP ne potrebuje dostopa do vseh podatkov v podatkovni bazi podatkovne analitike, ampak samo do dela podatkov, ki je že agregiran in pripravljen za obdelavo,
- OLAP lahko izvaja kompleksnejše analize, ki ne motijo operativnega delovanja sistema podjetja,
- OLAP uporablja že konsolidirane in urejene podatke.

OLAP ne smemo enačiti s podatkovnim skladiščem, saj je slednje skupek vseh konsolidiranih podatkov iz različnih virov (transakcijskih sistemov). Pretvorba konsolidiranih podatkov ter matematične operacije med temi seti podatkov (agregacija, seštevanje, grupiranje) pa so predpriprava za orodja OLAP, saj se le-ta povežejo na večdimenzionalni model podatkov in izvajajo različna poročila, pa tudi podatkovno rudarjenje.

1.3 Vrste arhitektur sistemov podatkovne analitike in njihov namen ter uporaba

Uvodoma smo že predstavili, zakaj podjetje sploh potrebuje sistem za podatkovno analitiko. Vrsto podatkovnega skladišča največkrat določa lastna arhitektura sistema. Glede na lastno arhitekturo ločimo tri osnovne vrste podatkovnih skladišč (naštete so štiri, ker je tretja alineja kombinacija prvih dveh):

- centralizirano podatkovno skladišče (Inmon, Welch & Glassey, 1997),
- distribuirano podatkovno skladišče (Kimball & Ross, 2002),
- kombinacija obeh – federativno ali hibridno podatkovno skladišče,
- pomnilniške (angl. *in-memory*) podatkovne baze.

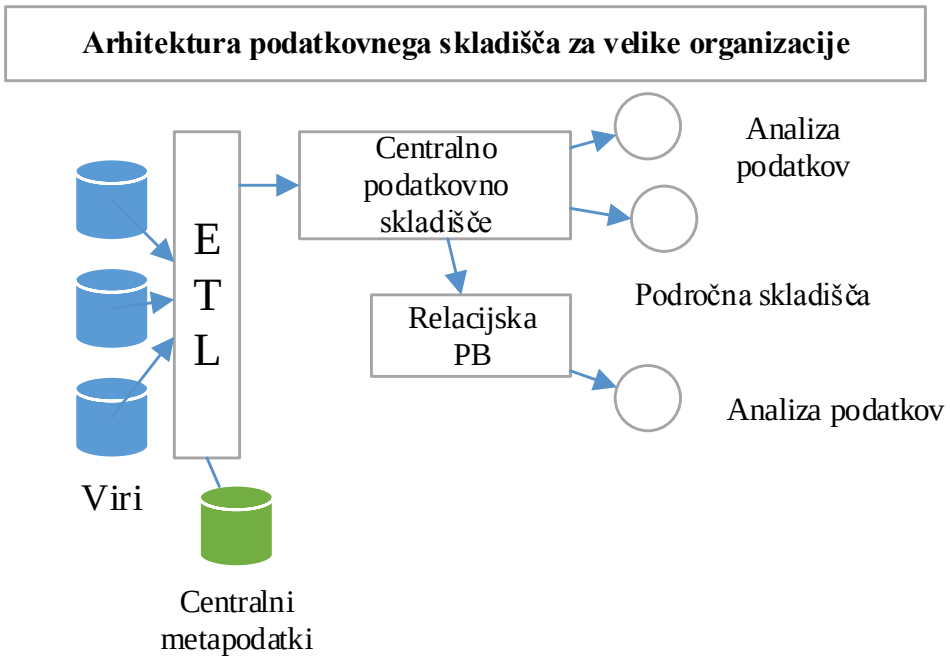
Prvo vrsto v svoji definiciji zagovarja W. H. Inmon, drugo pa R. Kimball, obe pa sta primarni vrsti. Tretja je kombinacija obeh arhitektur in ji pravimo tudi federativna (hibridna); četrta pa je plod razvoja in svetovnih trendov na področju arhitektur obvladovanja večje količine podatkov s hitro pridobljenimi rezultati v pomnilniku računalnika.

W. H. Inmon in R. Kimball sta pionirja podatkovnih skladišč, zato se vsi več ali manj sklicujemo na njuni definiciji oziroma njuni osnovni arhitekturi. Podatkovna skladišča, ki obstajajo, in tista, ki sem jih sam zasledil, so kombinacija tako centraliziranega kot distribuiranega podatkovnega skladišča. Možno je, da se povsem držimo enega ali drugega koncepta, pri sami izvedbi pa se pokaže, da je potrebno zaradi samega poslovnega problema uporabljati kombinacijo obeh. To ne pomeni, da delamo narobe, ampak da izmed obeh pristopov uporabimo tistih del, ki je najbolj primeren za določen del implementacije podatkovnega skladišča – glede na vrsto poslovanja. V literaturi pa se je v zadnjih letih, kljub stabilnosti teh dveh različnih vrst podatkovnih skladišč, začelo pojavljati več različnih arhitektur sistema podatkovne analitike.

Prav tako obsežen korak je pomenila odločitev, katero vrsto podatkovnega skladišča oziroma katero arhitekturo sistema podatkovne analitike bomo izbrali. Po izkušnjah sodeč lahko vse do koraka implementacije podatkovnega skladišča še razmišljamo, katero vrsto podatkovnega skladišča bomo izbrali. Proces prenesi-pretvori-naloži – pogosteje poimenovan ETL (angl. *Extract Transform Load*, v nadaljevanju ETL), je lahko za vsako vrsto malce drugačen, saj že izbira vrste praviloma pomeni samo sestavo procesa ETL, dinamiko polnjenja, granularnost podatkov v podatkovnem skladišču in možnosti uporabe. Glede na osnovno definicijo vrst podatkovnega skladišča: centralizirano podatkovno skladišče, distribuirano podatkovno skladišče in kombinacija obeh, lahko vrste razdelimo še na bolj podrobne tipe arhitektur, ki jih poznamo pod imeni (Sen & Sinha, 2005):

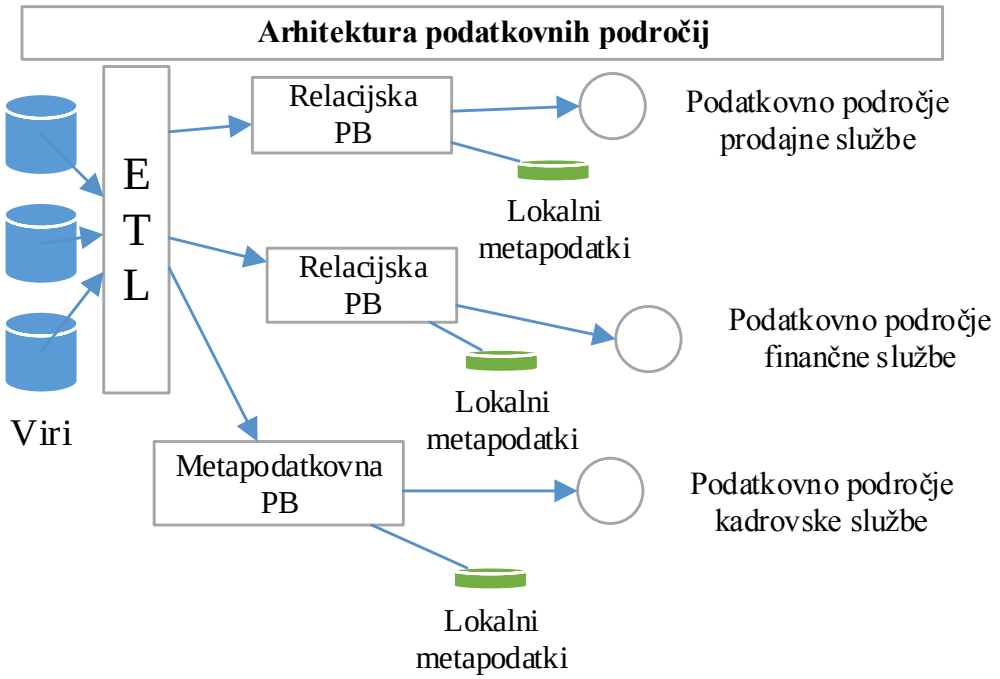
- (centralno) podatkovno skladišče (angl. *Enterprise Data Warehouse Arhitecture*),
- arhitektura področnega skladišča (angl. *(Federated) Data Mart Arhitecture*),
- arhitektura centraliziranega podatkovnega skladišča (angl. *Hun-and-spoke Data Mart Arhitecture*),
- (podjetno) podatkovno skladišče z operativno podatkovno shrambo (angl. *Enterprise Warehouse with Operational Store*),
- porazdeljena arhitektura podatkovnega skladišča (angl. *Distributed Data Warehouse Arhitecture*).

Slika 5: Arhitektura podatkovnega skladišča za velike organizacije



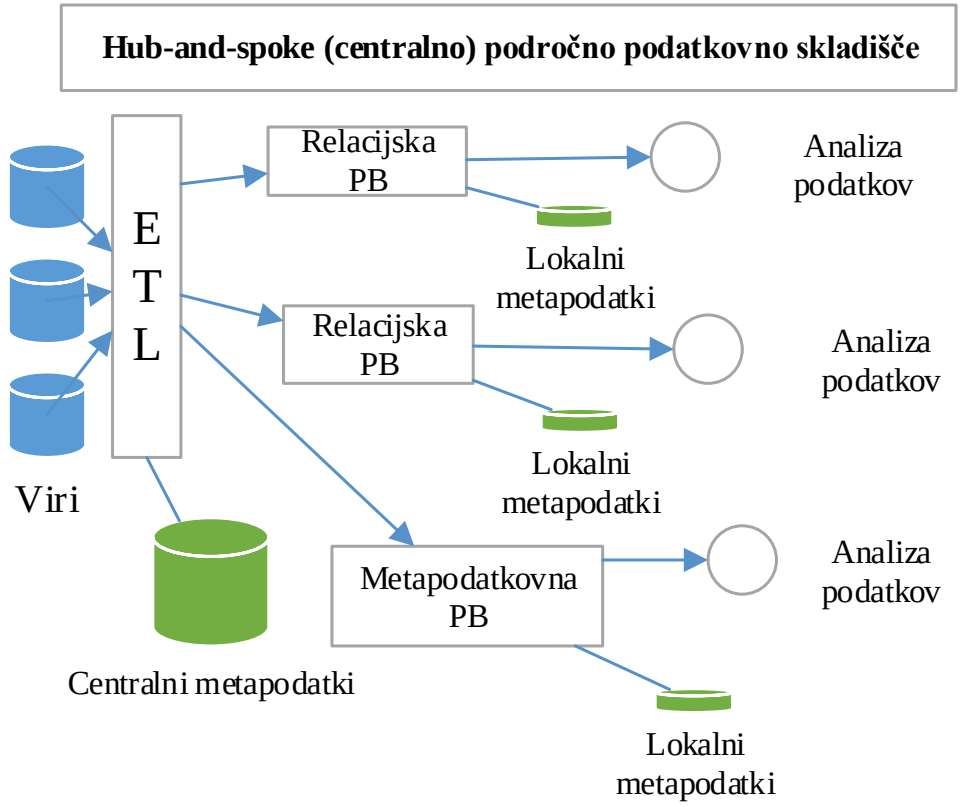
Vir: A. Sen & A. P. Sinha, A Comparison of Data Warehousing Methodologies, 2005, str. 80.

Slika 6: Arhitektura podatkovnih področij



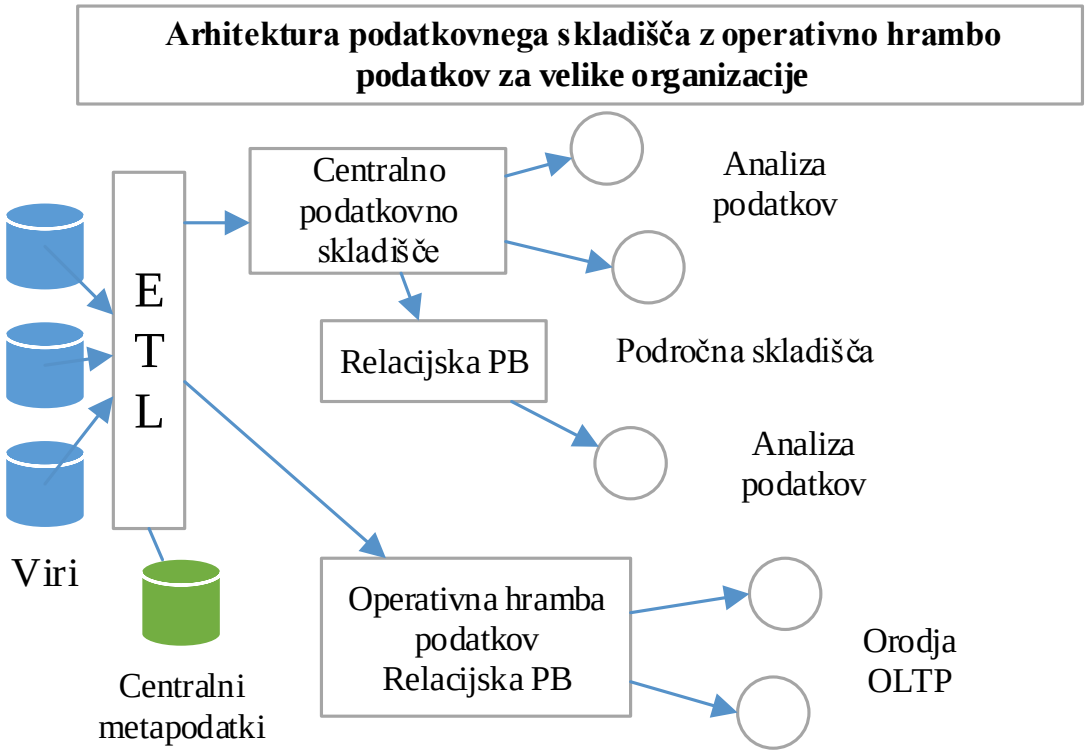
Vir: A. Sen & A. P. Sinha, A Comparison of Data Warehousing Methodologies, 2005, str. 80.

Slika 7: Hub-and-spoke (centralno) področno podatkovno skladišče



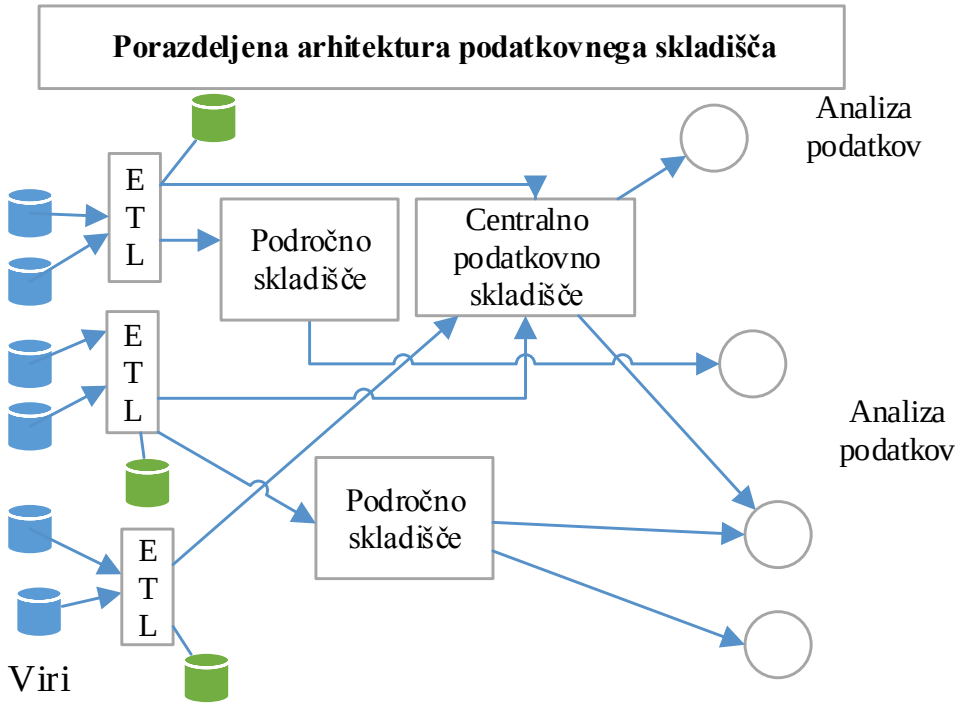
Vir: A. Sen & A. P. Sinha, A Comparison of Data Warehousing Methodologies, 2005, str. 80.

Slika 8: Arhitektura podatkovnega skladišča z operativno hrambo podatkov za velike organizacije



Vir: A. Sen & A. P. Sinha, A Comparison of Data Warehousing Methodologies, 2005, str. 80.

Slika 9: Porazdeljena arhitektura podatkovnega skladišča



Vir: A. Sen & A. P. Sinha, A Comparison of Data Warehousing Methodologies, 2005, str. 80.

Za definiranje vseh petih arhitektur, prikazanih na Slikah 5, 6, 7, 8 in 9, sta avtorja uporabila več ali manj vse možne kombinacije, ki se pojavljajo v praksi. Vse arhitekture so teoretično predstavljene in se uporabljajo za različne namene ter različne stopnje zrelosti podjetij, ki se odločijo za eno izmed arhitektur. V literaturi (Buyer's Guide, b.l.; Sen & Sinha, 2005; In which situations is NoSQL better than relational databases such as SQL, 2016) so predstavljene izvedbe posameznih arhitektur podatkovnih skladišč na različnih platformah (Platform, 2016), ki jih poznamo. Opredelimo jih z naslednjimi opisi:

- tradicionalna relacijska podatkovna baza (velikokrat rečeno sistem upravljanja podatkovne baze),
- specializiran analitičen sistem za upravljanje podatkovne baze,
- NoSQL podatkovna baza,
- pomnilniška podatkovna baza,
- namenski sistemi in
- podatkovna skladišča v oblaku.

Okolja ali platforme se torej razlikujejo glede na potrebe podjetij, vendar se vse platforme namestijo kot centralno podatkovno skladišče, področno podatkovno skladišče ali pa kot kombinacija obeh, kar smo že zasledili pri federativnem podatkovnem skladišču v tem poglavju. Centralno podatkovno skladišče se po definiciji uporablja v celotnem podjetju, kar pomeni, da je eno in edino za celotno podjetje. Področno skladišče (angl. *Data Marts*, v nadaljevanju DM) je manjše podatkovno skladišče od centralnega in se osredotoča na individualne ali specifične potrebe določenega oddelka v podjetju. Kot smo že večkrat poudarili, lahko v podjetju naredimo več področnih podatkovnih skladišč in jih povežemo (integriramo) s centralnim podatkovnim skladiščem (preko skupnih povezovalnih ključev – univerzalnih identifikatorjev).

1.3.1 Centralizirano podatkovno skladišče

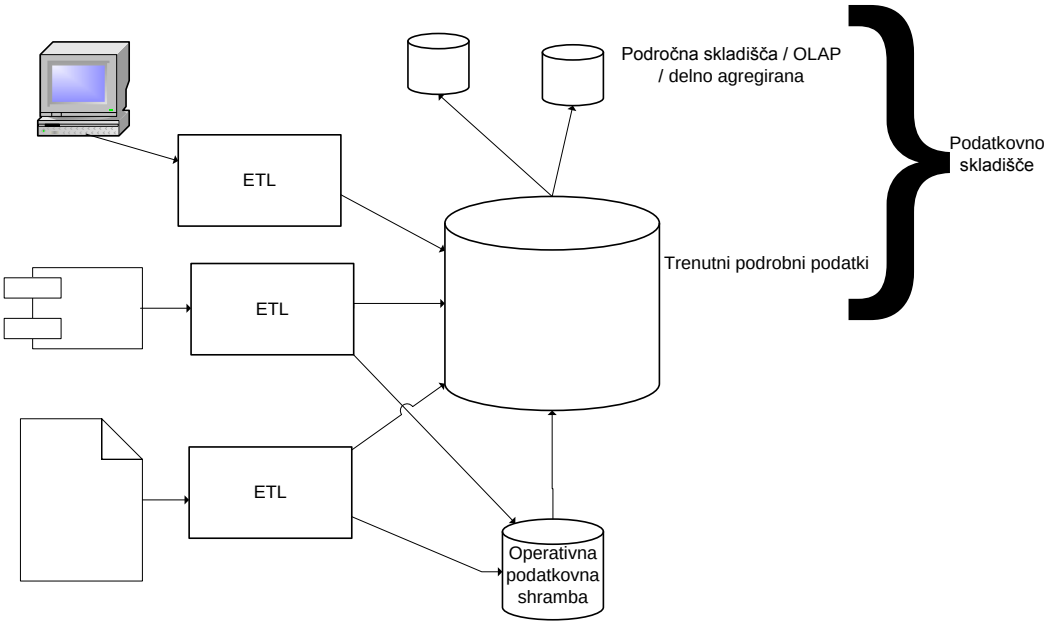
Slika 10 predstavlja centralizirano podatkovno skladišče, katerega zagovornik je W. H. Inmon (Inmon et al., 1997). Ta pristop temelji na dejstvu, da v osnovi izdelamo podatkovno skladišče, nato pa iz njega izpeljemo manjša (odvisna) področna skladišča. Tako izpeljana področna skladišča so odvisna struktura, saj so podatki izpeljani oziroma naloženi izključno iz centralnega podatkovnega skladišča. Gradnik ETL ima v osnovi vlogo prenesti podatke iz virov, jih pretvoriti in naložiti v končno obliko. Osnovne značilnosti, ki so predstavljene na Sliki 10 ter ločijo podatkovno skladišče in področno skladišče (Inmon et al., 1997), so:

- podatkovno skladišče vsebuje veliko količino podrobnih podatkov iz daljšega obdobja v enostavnih strukturah, področno skladišče pa vsebuje le agregirane in obdelane podatke kratkega obdobja, navadno v kompleksnih strukturah,

- struktura podatkovnega skladišča je namenjena za različne uporabe, struktura področnega skladišča pa za ozko usmerjen namen,
- področna skladišča so občutno manjša,
- podatkovno skladišče vsebuje tudi že agregirane in s tem obdelane podatke poslovanja, kar lahko v področnem skladišču še nadaljnje agregiramo.

Operativna podatkovna hramba je predstavljena kot hibridna struktura, ki izpolnjuje operativne in analitične zahteve. Ima majhen odzivni čas, obenem pa je prostor integriranih podatkov iz različnih virov (Kimball, 1996). Zaradi svoje vsebine je predmetno usmerjena in integrirana. Ne vsebuje zgodovinskih, ampak le aktualne oziroma trenutne podatke.

Slika 10: Arhitektura podatkovnega skladiščenja po Inmonu



Vir: Inmon et al., Managing the Data Warehouse, 1997, str. 79, slika 5.1.

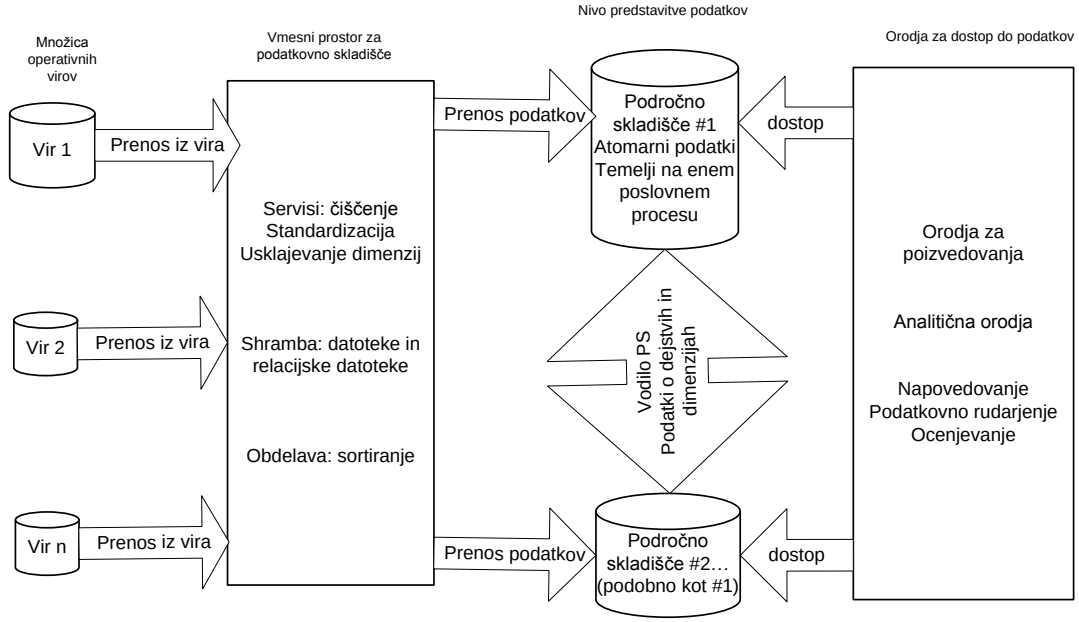
1.3.2 Distribuirano podatkovno skladišče

Na Sliki 11 je predstavljeno distribuirano podatkovno skladišče, katerega zagovornik je Kimball (Kimball & Ross, 2002). Ta pristop temelji na dejstvu, da se že v začetku in takoj po obdelavi podatkov osredotočimo na več področnih skladišč, ki med seboj niso nujno povezana. Neodvisna področna skladišča se tako imenujejo, ker niso odvisna med seboj niti ne morejo biti odvisna od nobene strukture, saj so v taki vrsti podatkovnega skladišča področna skladišča namenjena za točno določene namene. Takšna vrsta arhitekture omogoča hitro in razmeroma enostavno gradnjo prvega področnega skladišča, kar ima zelo dober vpliv na uvedbo podatkovnega skladišča ter zato lahko hitro dobimo prve rezultate.

Tak pristop je zelo navdušujoč, saj pridobimo delujočo podatkovno skladišče, obenem pa podporo zagovornikov s strani vodstva in končnih uporabnikov.

Ima pa ta pristop tudi problem, ki sploh ni zanemarljiv, ker je namreč vsako področno skladišče med seboj neodvisno, to pomeni, da ima med seboj tudi neodvisne osnovne primarne identifikatorje (Kimball, 1996). Področno skladišče za potrebe oddelka trženja ima lahko drugačen pogled na osnovni in atomarni zapis, kot ga ima področje nabave. Zato pri združevanju, in do tega slej ko prej tudi pride, nastajajo velike težave pri iskanju univerzalnega identifikatorja in s tem tudi težave pri uparjanju podatkov, ki naj bi logično sodili skupaj.

Slika 11: Pristop podatkovnega skladišča R. Kimballa

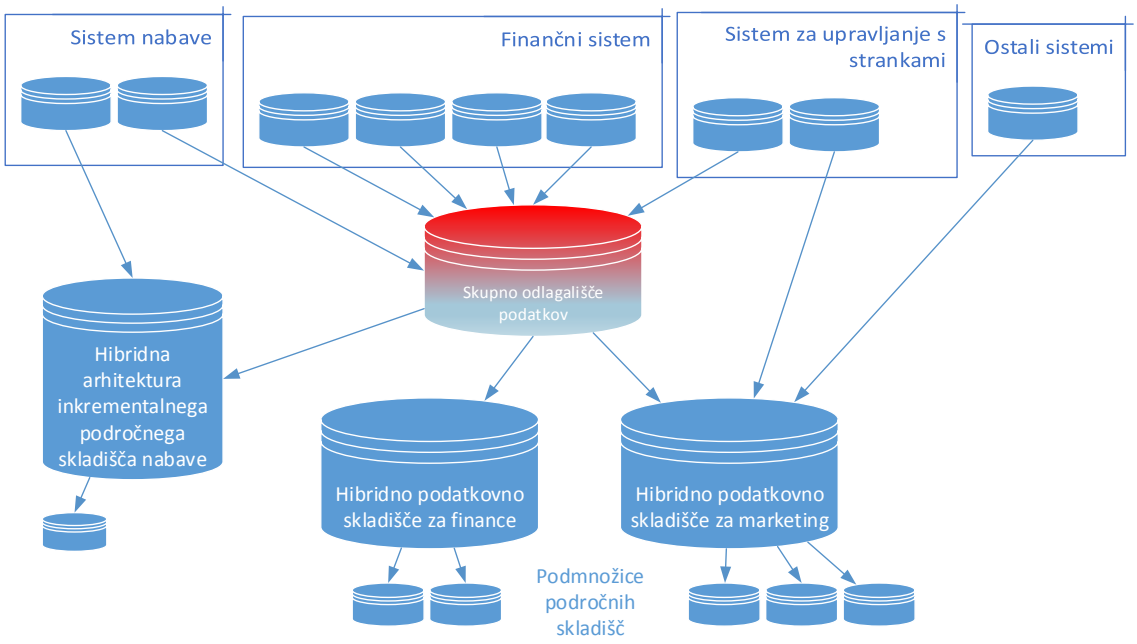


Vir: R. Kimball & M. Ross, Managing the Data Warehouse, 2002, str.7, slika 1.1.

1.3.3 Federativno podatkovno skladišče

Na Sliki 12 je kombinacija centralnih in področnih podatkovnih skladišč, ki se je oblikovala v federativno, velikokrat rečeno tudi hibridno podatkovno skladišče. Arhitektura temelji na skupnem poslovnem modelu in področjih priprave informacij v skupni rabi (Hackney 2000a, 2000b, 2000c; White, 2000). Ta arhitektura zagotavlja nizke stroške in zelo hitro povrnitev vloženih sredstev z uporabo neodvisnih področnih skladišč, saj kasnejša podatkovna integracija ni potrebna.

Slika 12: Federativno oziroma hibridno podatkovno skladišče



Vir: Future of Data Warehouse, Data Mining and Data Visualisation, 2016, slika 1.

Tak pristop podpira iterativni razvoj podatkovnega skladišča in vsebuje neodvisna področna skladišča. Pomembno je, da je narejen skupni podatkovni model poslovnih podatkov, saj njegova izdelava zagotavlja konsistentnost uporabe imen podatkov (metapodatkov).

Ob vsaki gradnji novega področnega skladišča se skupni poslovni model ažurira (osveži). Če je okolje, kjer so vodilo poslovne zahteve uporabnikov iz poslovnega sveta, najprej razvijemo skupni poslovni model, nato pa ga uporabimo za razvoj podatkovnih modelov podrejenih področnih skladišč.

Vendar ima tudi ta pristop slabo lastnost, in sicer je vsakič, ko se gradi novo področno skladišče, potrebno vpeljati nov set aplikacij za zajem in transformacijo, ki praviloma nista integrirana z aplikacijami za gradnjo ostalih področnih skladišč. To lahko vodi v težavne situacije pri uporabi, saj uporaba neodvisnih področnih skladišč poveča tudi število programskih orodij za zajem in transformacijo. Vendar se tudi tako težavo lahko odpravi, in sicer z razbitjem procesiranja v več korakov – razvije se množica rutin, ki zajemajo podatke in z njimi polnijo (operativno) področje priprave podatkov. Nato se podatke s teh področij prenese v neodvisna področna skladišča z orodji ETL v obliko, v kateri je podatkovno skladišče.

1.3.4 Pomnilniška arhitektura sistema podatkovne analitike

Glede arhitekture računalniških sistemov so diski (tukaj so mišljeni trdi diski) vedno veljali kot prenosni mediji, kot mediji za hranjenje podatkov, mesto za odlaganje podatkov, kar ne pomeni, da temu danes ni tako. Trdi diski kot mediji so dandanes še vedno nujno potrebni in nepogrešljivi za delovanje strežnikov, namiznih računalnikov ter prenosnikov.

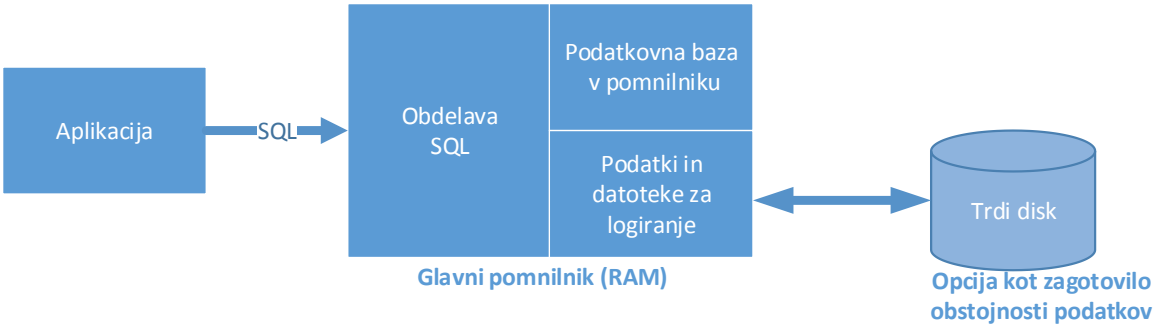
Napredek v zadnjih nekaj letih je prinesel zamenjavo pogonskih magnetnih trdih diskov, kot jih poznamo že več desetletij, s pomnilniškimi diski (angl. *solid state disk*, v nadaljevanju SSD), ki za svojo uporabo ne koristijo elektromagnetnega delovanja za pisanje in branje podatkov (SSD, b.l.; Shih-Wen, 2010; Hard disk drive, 2016), ampak za svoje delovanje uporabljajo pomnilniške elemente, katerih razlaga presega okvire tega dela.

V tem poglavju poskušamo predstaviti možnosti, da bi namesto klasičnih trdih diskov za velike količine podatkov arhitekture podatkovnega skladišča uporabljali kar pomnilniške diske. Žal temu ni tako, ker hrambe oziroma arhitekture podatkovnih skladišč uporabljajo velikostni razred podatkov nekaj TB (angl. *Tera Byte*), pomnilniški diski pa so še vedno ranga 1 TB, poleg tega pa so slednji še vedno precej dražji od klasičnih trdih diskov in imajo, žal, tudi omejeno število branj in pisanj (SSD, b.l.). To pa je v arhitekturi podatkovnih skladišč nesprejemljivo iz razloga, ker se podatki velikokrat berejo in pišejo (v to so všteti: prvi vpis v pomnilniški element, popravljanje ter brisanje podatka).

Glede na povzetke literature ugotavljamo, da izredno hitre pomnilniške diske (SSD, b.l.) ni mogoče uporabljati oziroma vsaj ni smotrno uporabljati za podatkovna skladišča (ne glede na arhitekturo). Pravi preboj v hitrosti in zmogljivosti se je zgodil zadnja leta s prihodom tehnologije »in-memory« (Knabke & Olbrich, b.l.; Howard, 2012; H. Zhang, Chen, Ooi, Tan & M. Zhang, 2015), kar bi lahko enostavno prevedli v »v pomnilniku«. Kadar je v svetu informatike omenjan pomnilnik, je implicitno mišljen pomnilnik RAM (angl. *Random Access Memory*, v nadaljevanju RAM) ali, rečeno, tudi delovni pomnilnik (angl. *main memory*). Ta pomnilnik uporablja računalnik v svoji osnovi za delovanje operacijskega sistema in je povprečno velikosti 4 GB – 8 GB za prenosnike in osebne računalnike ter 8 GB -128 GB (tudi več) za strežnike.

Slika 13 prikazuje zelo splošno in konceptualno zasnovo arhitekture pomnilniške podatkovne baze, ki jo pogosto imenujemo IMDB (angl. *In-Memory DataBase*, v nadaljevanju IMDB), velikokrat tudi MMDB (angl. *Main-Memory DataBase*, v nadaljevanju MMDB). Gupta, V. Verma in M.S. Verma (2013) so predstavili koncept, ki preprosto deluje tako, da aplikacija preko poizvedovalnega jezika in procesorja bere podatke iz pomnilnika, ta pa mora za svoje delovanje zagotavljati, da se spremembe zapisujejo ter se v primeru prekinitve električnega toka ne izgubijo že potrjeni podatki (potrjene transakcije).

Slika 13: Visokonivojska arhitektura sistema pomnilniške podatkovne baze



Vir: Gupta M. K. et al., In-Memory Database Systems - A Paradigm Shift, 2013, stran 334.

Večja količina pomnilnika danes v velikih okoljih ni več želja, ampak nuja (Gene, 2015; Zhang et al., 2015), saj je večina komercialnih ponudnikov podatkovnih baz že predstavila delovanje in procesiranje podatkovne baze popolnoma podprto v pomnilniku (Mansharamani, 2013). Ne glede na podporo pa vemo, da je upravljanje podatkov v pomnilniku še vedno v povojih in se bo še nekaj let krepko razvijalo ter spreminjalo.

Zhang et al. (2015) so pregledovali delovanja pomnilniškega sistema podatkovne strukture, upoštevaje lastnosti ACID (ACID, 2016), glede na naslednje aspekte:

- Indeksi: Iskanje podatkov v arhitekturi podatkovne analitike, ki uporablja trde diske (ti so še vedno najbolj razširjeni), je popolnoma različno od principov iskanja podatkov v pomnilniku. Pri uporabi tehnologije trdih diskov moramo zmanjševati vhodno/izhodne operacije, pri iskanju podatkov v pomnilniku pa je pomembno, da se osredotočamo še na izrabo pomnilnika in predpomnilnika – z namenom, da se izognemo prekomernemu »skakanju« po pomnilniku. Obstaja pa že več algoritmov, ki presegajo okvir tega magistrskega dela.
- Podatkovni nivoji: Za podatke v pomnilniku je zelo pomembno, kako jih vanj shranjujemo. V tem sklopu je najbolj optimalna postavitev s stolpci (angl. *columns*), saj je za tako obliko značilna boljša analitika podatkov, večja njihova kompresija, kar pa na drugi strani ne moremo trditi za okolja OLTP. Kot že v prejšnjem odstavku zapisano, se bosta na tem področju morali pojaviti še kakšna optimizacija in pomnilniška fragmentacija podatkov.
- Vzporednost (paralelnost): Iz naslova enostavnosti tega aspekta povzemamo, da je zelo pomembno, kje in kako se paralelno izvajanje dogaja. Glede na arhitekturo je priporočljivo in najhitreje čim bližje procesorju računalnika. Cilj delovanja je postavljen na način, da se v enem procesorskem taktu oziroma enem ciklu procesorja izvede obdelovanje (procesiranje) podatka.
- Kontrola sočasnosti / upravljanja s transakcijami: Pri delu s podatki na trdem disku in s podatki, ki so v celoti v pomnilniku, se pojavijo izzivi s kontrolo sočasnih dostopov.

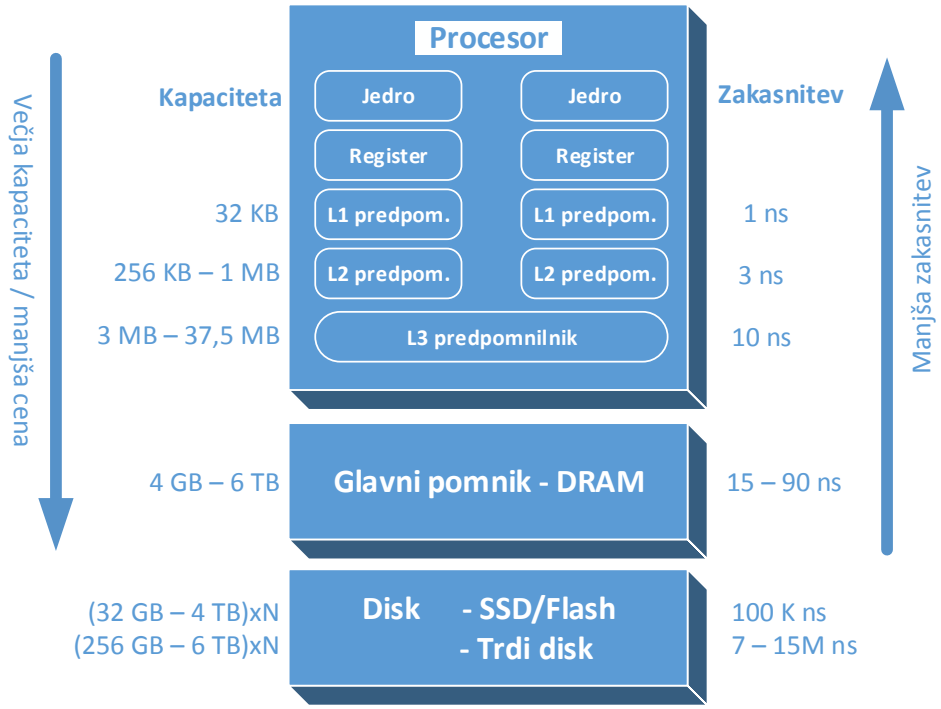
Dokler so dostopi samo bralne narave, podatki se samo berejo, težav ni. Nastopijo pa, ko želimo podatke popravljati. Takrat uvedemo mehanizem konsistentnosti, kar pomeni, da ne pokvarimo podatkov oziroma ne pustimo, da nam ostali procesi spremenijo tiste podatke, katere ravno mi spreminjamo ta trenutek. Posledično to pomeni, da moramo imeti mehanizem, da transakcijo, ki podatke, katere ima ravno naš proces v obdelavi, na nek način zaustavimo in tej transakciji sporočimo, da popravljanje določenega seta podatkov v tem trenutku ni možno.

- Poizvedovanja: Tudi na področju izvajanja poizvedb (angl. *Structured Query Language*, v nadaljevanju SQL) se klasična podatkovna baza, ki uporablja trdi disk za trajen medij, zelo razlikuje od podatkovne baze v pomnilniku. V tej smeri bo v prihodnosti še veliko raziskav in principov hitrejšega iskanja podatkov.
- Odkrivanje in odprava napak: Glede na dejstvo, da je podatkovna baza v celoti v pomnilniku, je skrb za trajnost podatkov na mestu. V nekaterih okoljih so se odločili, glede na poslovanje, da izguba podatkov iz pomnilnika (npr. ob izpadu električne energije, okvara strojne opreme) ne pomeni poslovnega tveganja. Nasprotno pa v marsikaterih okoljih izguba podatkov podatkovne baze v pomnilniku pomeni poslovno škodo in poslovno tveganje, zato, zahvaljujoč SSD, že lahko izdelujejo rezervne kopije podatkov v pomnilnikih za tovrstne medije za primer, da 128 GB podatkovna baza v pomnilniku za svojo rezervno kopijo podatkov lahko uporablja 128 GB SSD (logiranje transakcij, kopija registrov itd).
- Obseg podatkov: Velikost pomnilnika je v informatiki še vedno precej manjša od velikosti trdih diskov. Glede na obseg podatkovnega skladišča in izbiro določene arhitekture podatkovne analitike je količina pomnilnika za celotno podatkovno bazo lahko premajhna. Rešitve nakazujejo, da se bodo začele uporabljati kombinacije obeh – tako klasične hrambe podatkov na trdih diskih kot podatkov v pomnilniku. Glede na razvoj in potrebe pa se bo v prihodnosti pričelo standardizirati, kateri del v arhitekturi podatkovne analitike se bo »selil« v pomnilnik računalnika, kateri podporni deli arhitekture pa bodo ostali na trdih diskih. Še dodatno zadevo lahko olajšajo mediji SSD, ki se relativno enostavno vključujejo v delovanje podatkov v pomnilniku ali kot podaljšek pomnilnika (ko prične zmanjkovati razpoložljivega prostora). V razvoju so tudi algoritmi, ki bodo pripomogli k učinkovitejši izrabi arhitekture. Ta bo razmejila, kaj sodi v pomnilnik, kaj pa v drugo obliko (trdi disk, SSD).

Pri pomnilniški (»in-memory«) podatkovni bazi je že iz imena razvidno, da sloni na delovanju v pomnilniku, zato bom namenil več pozornosti samemu delovanju, saj je to potrebno zaradi nadaljnje izbire arhitektur.

V osnovi obstaja več različnih pomnilniških nivojev, katere imenujemo hierarhija pomnilnika.

Slika 14: Hierarhija pomnilnika



Vir: Zhang et al., In-Memory Big Data Management and Processing: A Survey, 2015, str. 1923, slika 3.

Slika 14 nazorno prikazuje, kako so v procesorju (angl. *Central Processing Unit*, v nadaljevanju CPU) registri, ki so fizično na procesorju, torej je dostop procesorja do teh registrov praktično 0 ns. Na desni strani slike je predstavljena zakasnitev (drugače rečeno tudi latenca), ki nam s številko prikazuje, koliko časa je potrebno od zahteve po podatku do pridobitve podatka. Na levi strani pa je označena kapaciteta, tako od nekaj KB (angl. *Kilo Bytes)* do TB. Na levi strani od zgoraj navzdol pada tudi cena.

Glede na Sliko 14 pa je razvidno, da hitrost odziva pridobivanja podatka pada, ko se logično oddaljujemo od pomnilniških gradnikov na procesorju proti pomnilniku in diskom. Naštejmo jih po vrsti, kot so tudi fizično postavljeni in strukturirani:

- registri,
- L1 predpomnilnik,
- L2 predpomnilnik,
- L3 predpomnilnik,
- glavni pomnilnik (RAM) ter
- diski: trdi disk, SSD, pomnilniški disk (angl. *flash disk*).

V osnovnem delovanju gre vsaka obdelava podatka skozi procesor, kar pomeni, da gre skozi vse nivoje:

disk →glavni pomnilnik
→ L3 predpomnilnik
→ L2 predpomnilnik
→ L1 predpomnilnik
→ register, kjer se zgodi obdelava, in v obratnem vrstnem redu nazaj proti disku
register → predpomnilnik
→ L1 predpomnilnik
→ L2 predpomnilnik
→ L3 predpomnilnik
→ glavni pomnilnik.

Tam se potem izvajajo še preostale operacije, ki so potrebne za izvedbo določene akcije, klica funkcije ipd. Sistem, ki se ukvarja s tem, kako vse te podatke pošiljati v obdelavo v register procesorja, mora biti učinkovit in uigran. Hkrati pa skrbi tudi za pravo razmerje, kje lokacijsko hraniti in kako obdelovati začasne podatke (podatki. ki še niso potrebni ta trenutek, bodo pa naslednji trenutek pri združevanju rezultatov). Na tem mestu se ta umetnost optimizacije ukvarja skoraj ves čas samo s tem, da ugotavlja, kjer se bo podatek nahajal, da bo naslednji podatek nekje zelo blizu trenutnega, ali pa da bo potem v bližnji prihodnosti spet uporaben. Določiti pravo mejo med tema dvema dejstvoma je ključ do izkoriščenosti pomnilnikov na vseh naštetih nivojih.

Registri so najhitrejša računalniška komponenta. Glede na to, da so neposredno v procesorju, je zakasnitev praktično enaka 0 ns. V registre se torej zapisujejo podatki, ki se potem uporabijo za izvajanje aritmetičnih operacij, katere mora procesor opravljati za vse priključene zunanje enote. Dolžina besede (npr. ukaza), ki se zapiše v register, je praviloma tudi dolžina besede v procesorju. Po izvajanju operacij se iz registra rezultat prepiše v tako imenovane izhodne registre, kar se potem nadaljuje po poti do glavnega pomnilnika. Obstajajo pa seveda tudi večji registri – velikosti tudi do štiri besede, kar pomeni, da se za en ukaz iz registra podatki štirikrat prenesejo v izvajanje.

Predpomnilnik (angl. *cache*) se poglavitno uporablja med registri in glavnim pomnilnikom zato, da se pretok podatkov iz glavnega pomnilnika čim hitreje izvaja v registre in obratno. Predpomnilnik je sestavljen iz elektronskih gradnikov za SRAM (angl. *Static Read Access Memory*, v nadaljevanju SRAM), ki je mnogo hitrejši od glavnega pomnilnika DRAM (angl. *Dynamic Read Access Memory,* v nadaljevanju DRAM). Stopnje predpomnilnika L1, L2 in L3 so samo stopnje, ki se sicer med seboj razlikujejo glede na zakasnitev dostopa ali kapaciteto, v glavnem pa opravljajo skoraj vse stopnje isto funkcionalnost – most med registri in glavnim pomnilnikom.

Glavni pomnilnik in disk - glavni pomnilnik je dejansko lahko edini med zunanjimi enotami direktno naslovljen s strani CPU, saj mu slednji pošilja rezultate ali pa sprejema

ukaze programa v glavnem pomnilniku. Glavni pomnilnik je navadno sestavljen iz nestabilnega RAM-a, kar pomeni, da se ob prekinitvi električne energije izgubi tudi vsebina RAM-a. Zadnji tehnološko dovršen gradnik je DRAM, ki je postal cenovno dostopnejši (cenejši) in zato primeren za pomnilniške podatkovne baze (angl. *in-memory databases)*. Avtorja Robbins (2008) in Zilka (2011) sta celo izjavila, da glavni pomnilniki postajajo diski oziroma dobivajo vlogo, kot da bi bili diski. Žal v realnosti ranljivost DRAM-a, da ob prekinitvi električne energije izgubi vso svojo vsebino, še vedno narekuje tehnologijo, ki zahteva, da imamo vselej diske za rezervno kopijo podatkov. Zilka (2011) v svojem intervjuju večkrat navaja, da je sicer pomnilniška podatkovna baza verjetno res podatkovna baza prihodnosti, vendar se zadnja tehnologija spopada s kar nekaj izzivi, ki jih lahko strnemo v:

- Potrebna pretvorba podatkov, kot jo poznamo dandanes, lahko postane omejujoča. Odvečno delo (angl. *overhead*) bo vedno predstavljajo tisto delo, zaradi katerega lahko pomnilniške podatkovne baze, kljub hitrosti, postanejo neučinkovite.
- Izziv je seveda tudi vmesnik, preko katerega bodo sodelovale računalniške komponente. Ta mora biti enostaven in hiter.
- Zakasnitev – predvsem pri konverziji podatkov.

Zilka (2011) povzema, da je v primeru, ko sta problem dostop do podatkov in volumen podatkov, edina rešitev, da uvedemo čim manj strežnikov s podatki v pomnilniško podatkovno bazo. S tem se izognemo vsem težavam, ki sledijo zaradi deljenja podatkov, zakasnitve dostopa podatkov ipd.

V povezavi z diski ne smemo pozabiti še na dejstvo, da so ravno diski glede zakasnitve med najpočasnejšimi gradniki v računalniški infrastrukturi. Glede na Sliko 14 so popolnoma na dnu, sicer so najcenejši, vendar obenem tudi najpočasnejši. Z namenom, da bi se glavni pomnilnik (DRAM) pri komunikaciji z diskom vseeno izognil čakanju na podatke, tudi disk uporablja določeno pomnilniško strukturo, ki ji rečemo medpomnilnik (angl. *buffer*). Ta skrbi, da se podatki dovolj hitro vpisujejo iz glavnega pomnilnika na disk in se tudi hitro preberejo v glavnem pomnilniku (z diska). Ta medpomnilnik je torej dostopni mehanizem, preko katerega se izvajajo vse te operacije.

Gupta et al. (2013) še posebej poudarjajo, da je potrebno ločevati med miselnostjo, da je pomnilniška podatkovna baza le kopija vseh podatkov iz relacijske podatkovne baze. Take predpostavke so napačne, saj se, kot že prikazano, pomnilniške strukture razlikujejo od podatkovnih struktur na klasičnih trdih diskih. Še posebej pa se čedalje bolj izpostavlja iskanje po podatkih, ko se iz tradicionalnega iskanja po principu B-drevo v pomnilniku izvaja T-drevo iskanja podatkov.

Prihod podatkovnih baz »v pomnilnik« ter s tem tudi analitika »v pomnilniku« sta čedalje bolj prisotna in razširjena, zato se je že razvilo več modelov analitike »v pomnilniku«, s

tem pa tudi več modelov stiskanja podatkov v pomnilniških podatkovnih bazah. Osnovni princip delovanja je pri vseh podatkovnih analitikah »v pomnilniku« enak, razlikovati pa se začnejo pri obravnavanju velikega števila podatkov v pomnilniku in njihovi obdelavi. Kljub velikostim delovnega pomnilnika (RAM) ranga več TB, je potrebno vse podatke v pomnilniku obravnavati racionalno in učinkovito. Pri obravnavi podatkov v pomnilniku se je zato razvilo več algoritmov in pristopov, kako te podatke v pomnilniku, katerih hramba je dražja od podatkov na disku, učinkovito shranjevati v procesu delovanja ter učinkovito uporabiti pri analizah.

Krueger, Wust, Linkhorst in Plattner (2012) so podali vsaj dva razloga, zakaj uporabljati stiskanje podatkov pri uporabi podatkovnih baz »v pomnilniku«:

- zmanjšanje velikosti celotne podatkovne baze v pomnilniku ter
- povečanje performanc podatkovne baze z zmanjševanjem prenosa podatkov med delovnim pomnilnikom in procesorjem.

Drugi razlog je dejansko aktualen iz naslova zmogljivosti in arhitektur – tako procesorjev kot delovnih pomnilnikov. V računalniški arhitekturi se velik pomen pripisuje hitrosti delovanja procesorja, malo manj oziroma premalo pozornosti pa se osredotoča na hitrost delovanja delovnega pomnilnika. Logično to pomeni, da so trenutno procesorji mnogo hitrejši od delovnih pomnilnikov, saj se proizvajalci ukvarjajo predvsem s pospeševanjem delovanja procesorjev, čeprav je nedavno postala pomembna tudi njihova hitrost delovanja. Kot poljudno primerjavo lahko podamo naslednjo osnovno razliko: hitrost delovanja procesorja je pri taktu 3 GHz in več, delovni pomnilnik pa deluje s frekvenco 300 MHz in več – iz tega sledi, da so takti delovanja procesorjev teoretično vsaj desetkrat hitrejši od taktov delovanja pomnilnikov. Gledano s perspektive procesorjev ti, glede na arhitekturo, prejemajo in pošiljajo podatke iz ter v delovni pomnilnik, pri tem pa morajo velikokrat procesorji preprosto čakati, kar lahko pomeni precejšnjo omejitev ali počasnejše delovanje od želenega. Trenutno hitrosti delovnega pomnilnika iz naslova povedanega predstavljajo omejitev zaradi svojih zakasnitev (angl. *latency*).

Obstoječi algoritmi branja podatkov iz pomnilnika se zaradi navedenih omejitev osredotočajo na čim boljšo izkoriščenost in zgoščenost podatkov v delovnih pomnilnikih iz razloga, da se ob branju podatkov iz delovnega pomnilna v obdelavo v procesor prebere karseda veliko podatkov, po drugi strani pa se mora to branje izvesti čim manjkrat. Iz tega sledi, da gredo vsi algoritmi v smeri čim večje lokalizacije podatkov iz delovnega pomnilnika v procesor. Po Kruegerju et al. (2012) je torej pomembno, da se sproti zagotavlja čim več podatkov pri branju iz delovnega pomnilnika v procesor ter da so ti podatki na voljo procesorju ob naslednji zahtevi po branju podatka – temu pravijo »lokalnost podatka« (ang. *data locality*). V primeru večjedrnega procesorja so ti algoritmi še bolj zapleteni, saj se za dostop do pomnilnika borijo vsa jedra hkrati, dostopa pa naenkrat lahko samo eno jedro.

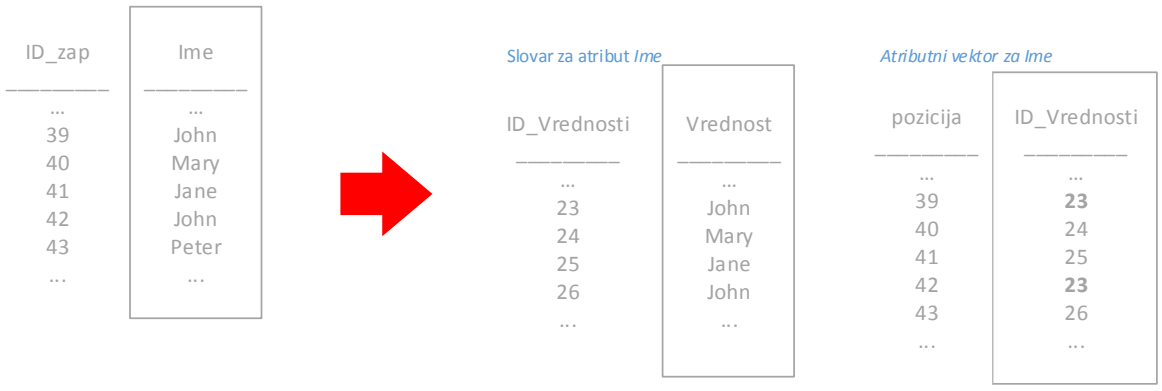
Algoritme stiskanja podatkov v pomnilniku (Krueger et al., 2012) pri IMDB lahko razdelimo v naslednje:

- Princip motivacije se osredotoča na delovne cikle procesorja, kadar čaka na podatke iz glavnega pomnilnika, ko dejansko procesor troši svoje cikle delovanja brez dodane vrednosti. V teh delovnih ciklih čakanja se zato izvajajo tako stiskanje (angl. *compress*) kot odpakiranje (angl. *decompress*) podatkov, vse dokler ima procesor take »prazne« cikle delovanja (kot ne počne praktično ničesar). V kolikor pa stiskanje in odpakiranje podatkov postane tako procesorsko intenzivno, da zaradi tega procesor ne uspe pravočasno opravljati ostalih dodeljenih operacij, postane stiskanje podatkov neučinkovito. Glede na dejstvo, da se nekatere operacije, to so izvajanja poizvedb (SQL), že lahko uporabljajo nad stisnjenimi podatki, lahko tako prihranimo čas odpakiranja podatka.
- Kodiranje RLE (angl. *Run-Length Encoding*, v nadaljevanju RLE): Ta algoritem uporablja funkcionalnost, da n-terice v obliki (1, 1, 3, 3, 3, 4, 4, 4) pretvori v parice (vrednost, število ponovitev), kar v dotičnem primeru pomeni v obliko ((1, 2), (3, 3), (4, 3)). Tak način shranjevanja ponovitev vrednosti pomeni, da se skrajša zapis podatkov pod pogoji, da so podatki urejeni zaporedoma ter ni veliko različnih vrednosti. Posplošeno je velikost takega stolpca dejansko število različnih vrednost. To kodiranje se v veliki meri zanaša na urejenost podatkov, ki so nared za stiskanje (kompresijo).
- Kodiranje bit – vektor (angl. *bit-vector encoding*): To kodiranje shranjuje bitno sliko (angl. *bitmap*) za vsako različno vrednost. Vsaka bitna slika mora zakodirati števila različnih vrednosti v bitih. Torej vrednost ena (1) v bitni sliki pomeni, da ima podatek na tistem mestu določeno vrednost. Velikost oziroma učinkovitost stiskanja podatka je zato odvisna od števila podatkov in števila različnih vrednosti.
- Kodiranje po metodi odstranjevanja vrednosti nič (0): Ta metoda se naslanja na odstranjevanje vodilnih ničel v vrednosti podatka. Glavni pokazatelj, kako dobro je tovrstno stiskanje podatkov, je povprečno število odstranjenih vodilnih ničel.
- Kodiranje slovarja (angl. *attribute dictionary vector encoding*): Ta vrsta kodiranja je najbolj razširjena v stolpčnih podatkovnih bazah (angl. *column-store*). Sloni na izdelavi tabele ali t.i. slovarja (angl. *dictionary*). Vanj se zapišejo unikatni identifikatorji za vse različne ponovljive vrednosti podatka – v vektor podatka pa se zapišejo unikatno zgrajeni identifikatorji. V slovar se vpisujejo vse kombinacije podatkov, v vektorju vrednosti pa hranimo samo identifikatorje. Tako prihranimo pri prostoru, saj so unikatno določeni identifikatorji manjši od vrednosti posameznih atributov. Vrednosti, kaj določen identifikator pomeni, pa so shranjene v slovarju. Praviloma tako prihranimo veliko prostora v pomnilniku, prav tako pa lahko obravnavamo vrednosti kot fiksne, saj se shranjujejo le identifikatorji (to so številčne vrednosti), kar izrazito pripomore k hitrosti delovanja. Ta način bi bil dejansko lahko zelo uporaben, vendar se, glede na prakso, izkaže, da velikost tega slovarja (različnih kombinacij vrednosti atributov) zelo pomembno vpliva na hitrost delovanja.

Kot povzetek Krueger et al. (2012) navajajo, da je, glede na primerjavo naštetih algoritmov, pomembno predvsem to, koliko zgrešenih klicev po podatkih iz pomnilnika doleti procesor – zadetek podatka v predpomnilniku ne štejemo, ker je to namen oziroma cilj hitrega delovanja. Tisto, kar nas mora dejansko skrbeti, so zgrešeni klici po podatkih, zaradi katerih je potrebno v medpomnilnik (ki je med procesorjem in glavnim pomnilnikom) ponovno naložiti določen set podatkov iz glavnega pomnilnika. V tem članku so avtorji ponazorili problem, ki je tipičen za večino podjetij, ter prišli do zaključka, da je pri 5% različnih vrednosti stolpca najbolj primerna metoda kodiranja RLE (v kolikor gre za sortirane vrednosti po velikosti), najslabše pa kodiranje bit – vektor. Zanimivo pa je tudi dejstvo, da se razmišljanje obrne v dobrobit algoritma kodiranja slovarja. Pri delovanju stiskanja podatkov po metodi RLE je potrebno sproti sortirati podatke, sicer tak algoritem ni smiseln. To posledično pomeni, da je potrebno z vsakim novih zapisom podatek uvrstiti med že obstoječe podatke, kar se rešuje preko tako imenovanega nadomestnega ključa podatka (angl. *surrogate key*). Kljub temu tako delovanje v nekaj (več) milijonskih tabelah ni tako učinkovito, kot je algoritem kodiranja slovarja. V transakcijskih sistemih (pa tudi analitičnih sistemih) se bolje obnese slednji algoritem kodiranja, ker novi podatki ne povzročajo rekonstrukcije obstoječih zapisov kot pri algoritmu RLE, ampak se lahko neposredno v slovar dodajajo samo odmiki – bodisi da gre za veliko ali malo različnih vrednosti zapisov.

Deshpande (2015) je metode stiskanja podatkov v pomnilniku pri IMDB ocenjeval s podobnega zornega kota kot Krueger et al. (2012), vendar je svoje raziskovanje razširil v smeri dodatnih tehnik stiskanja podatkov znotraj metode slovarja (angl. *attribute and dictionary vector*).

Slika 15: Slovar in atributni vektor pri vpisu podatkov



Vir: A. H. Deshpande, Efficient Compression Techniques for an In Memory Database System, 2015, stran 8978, slika 2.

Pri metodi stiskanja podatkov v pomnilniku bomo predstavili najbolj razširjeno in pogosto metodo – metodo slovarja in atributnega vektorja, kot je predstavljena na Sliki 15. Pri predstavitvi te metode ni pomembno samo to, kako se shranjujejo podatki, ampak tudi standardno delovanje poizvedovanja po podatkih. Princip delovanja z razdelitvijo podatka na slovar in atributni vektor spremeni tudi osnovno delovanje operacij v pomnilniški podatkovnih bazi (IMDB), tj. vnos novega podatka (angl. *insert*), brisanje podatka (angl. *delete*) in popravek podatka (angl. *update*). Pri IMDB je pred vnosom novega podatka potrebno najprej preveriti, ali podatek v slovarju že obstaja. Če obstaja, se v atributni vektor zapiše le še identifikator zapisa. V nasprotnem primeru je potrebno v slovar najprej vnesti novo vrednost, ga urediti (sortirati), nato pa popravljati atributni vektor. Popravek podatka sloni na podobnem principu, le da se popravi samo atributni vektor – implicitno je vključeno tudi urejanje slovarja, v kolikor te vrednosti v njem še ni (npr. v primeru popravka vrednosti atributa v drugo vrednost, ki še ne obstaja). Zanimivo je, da brisanje atributa v svetu IMDB ne pomeni dejanskega brisanja, ampak samo označitev t.i. zastavice (angl. *flag*), kateri podatek se je brisal – fizičnega brisanja ni, obstaja samo logično brisanje. Zastavica je tipično časovni žig (kdaj je podatek postal »brisan«) ali pa logična informacija – veljaven/neveljaven.

Stiskanje podatka v slovarju in stiskanje podatka v atributnem vektorju lahko obravnavamo ločeno ter za vsakega posebej izberemo način stiskanja:

- Stiskanje atributnega vektorja: Ker so vrednosti atributnega vektorja številčne, bi lahko te vrednosti stisnili na način, da bi vodili odmike od prve zapisane vrednosti že obstoječih zapisov v atributnem vektorju. To pomeni, da za vsak ponavljajoč podatek vodimo le prvotno vrednost, za vsake nadaljnje pa samo še odmik od prve vrednosti. S tem lahko dosežemo veliko stopnjo stiskanja podatkov v atributnem vektorju, poglobljeno pa lahko predstavimo še razširjene metode znotraj stiskanja podatka atributnega vektorja:
  - kodiranje predpone: primerno, kjer se večkrat zaporedno pojavljajo iste vrednosti v atributu, idealno pa deluje v primerih, kjer so podatki urejeni oziroma sortirani (v primeru priimkov bi to bila urejenost po prvi črki priimka, primer pa so tudi imena mest, držav, ipd),
  - kodiranje RLE: to metodo smo že predstavili v prejšnjem odstavku, tokrat lahko delovanje predstavimo kot podaljšek metode s kodiranjem predpone na vseh mestih, kjer se pojavljajo vrednosti v stolpcih. Metoda torej deluje z nadomeščanjem vsake večkrat ponovljive (sekvence) vrednosti z eno vrednostjo in eno ponovljivostjo ter z odmikom, kje se vse nadalje taka sekvenca ponavlja. Ker so vsi odmiki ponovljive vrednosti znani vnaprej, pri poizvedbah ni potrebno odpakirati vrednosti, ampak lahko poizvedba deluje neposredno nad stisnjenimi podatki,
  - kodiranje gruč (angl. *cluster encoding*): to kodiranje uporablja razdeljenost atributnega vektorja v gruče po n vrednosti v vsaki. Deluje po principu, da se vsaka

ponovljivost podatka v grupi zapiše kot bit 1 ali 0, če se ne ponovi. Tak pristop ni tako učinkovit, saj je potrebno precej preračunavanja, hkrati pa ne omogoča neposrednega dostopa do podatka pri poizvedbah, kar zelo poslabša odzivnost oziroma performance,

- vitko kodiranje (angl. *sparse encoding*): tovrstno kodiranje temelji na odstranjevanju konstantno ponavljajočih se vrednosti v atributnem vektorju. Uporablja se bitni vektor, ki sproti zapisuje, na katerem mestu je bila odstranjena ponavljajoča vrednost. Zelo je uporabno v primerih, kjer so v atributu privzete (angl. *default*) ali prazne vrednosti (angl. *null*), kar lahko nadomestimo s številko nič oziroma naslednjo prosto številko v vektorju. Učinkovitost se pokaže npr. pri nekaj milijonski tabeli s praznimi vrednostmi, ko v atributni vektor zapišemo vrednost podatka samo enkrat za vse pozicije, kjer se le-ta pojavi. Dostop do tako stisnjenih podatkov v atributnem vektorju je posreden, ker je potrebno sproti preračunavati originalno lokacijo podatka s pomočjo bitnega vektorja. To pa lahko zelo škodljivo vpliva na performance poizvedovanj,
- posredno kodiranje (angl. *indirect encoding*): princip pri posrednem kodiranju je podoben principu kodiranja gruč. Pri izdelavi posamezne gruče določene velikosti se preverja, koliko različnih vrednosti je znotraj posamezne gruče. V primeru malo različnih, se lahko izdela drugi nivo slovarja, kar pomeni, da naredimo podobno kot pri osnovnem pristopu. Tako gručo ponovno pregledamo in naredimo podatkovno strukturo, v katero shranjujemo povezave iz gruče na nove slovarje. Tovrstno kodiranje je najbolj učinkovito v primerih, ko ima večinoma ponavljajoče vrednosti (torej malo različnih vrednosti v podatkih).

- Stiskanje slovarja: Tipično je slovar sestavljen iz identifikatorja (ID) in pripadajoče vrednosti (ponavadi je to niz). Tudi ta niz je seveda možno kodirati po metodi, imenovani delta (angl. *delta*) kodiranje. Primerna je za kodiranje v slovarju, kadar imamo podatke o imenih mest celega sveta. V primeru, ko je slovar sortiran in podatek ne vsebuje več niza ampak številčne vrednosti, je možen neposreden dostop poizvedovanj do podatkov in s tem so na voljo hitri odzivi na poizvedbe.

Kot primer navajamo Oraclov primer stiskanja podatkov v pomnilniški podatkovni bazi (Oracle Database In-Memory – ORACLE WHITE PAPER, 2016). Oracle uporablja za stiskanje podatkov več načinov, opisali bomo samo privzetega, ki je najbolj uporaben in smiseln. Privzeta uporaba pri Oraclu je stiskanje podatkov z namenom čim hitrejših odzivnih časov povpraševanj po podatkih (angl. *MEMCOMPRESS FOR QUERY HIGH*), uporablja pa stiskalne tehnike, kot so: kodiranje slovarja, kodiranje RLE in kodiranje bit – vektor. Oracle zagotavlja stiskanje podatkov med 2 do 20 krat, odvisno od podatkovnega tipa, ki ga stiskamo.

1.4 Planiranje in potek razvoja podatkovne analitike

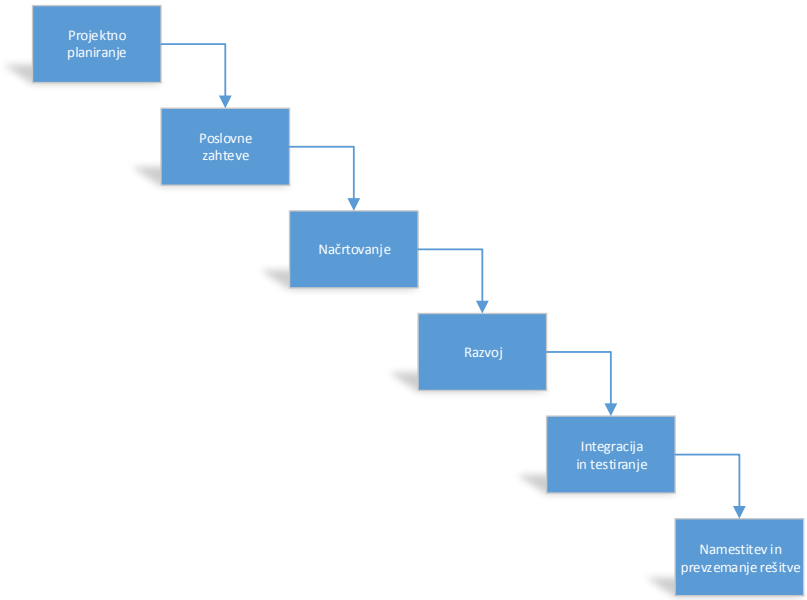
Poznamo dve glavni tradicionalni strategiji izdelave podatkovnega skladišča. Prvo je definiral Inmon (2002) in je izdelana po obratnem delovanju principa SDLC (angl. *System Development Life Cycle – watefall approach*, v nadaljevanju SDLC). Inmon (2002) predlaga, da namesto da bi začeli z definiranjem poslovnih zahtev, se raje osredotočimo in orientiramo na podatke na virih. Torej: podatki, ki jih imamo na virih, so tisti, ki »vodijo« planiranje in potek izgradnje. Potek je predstavljen kot zbiranje podatkov, integracija in testiranje integracije. Vsi programi, ki bodo uporabljali podatke v podatkovnem skladišču, so napisani glede na podatke, ki jih imamo. Podatke, ki smo jih prenesli, po obdelavi s programi še analiziramo. Šele nato se formulirajo poslovne zahteve. Tak pristop je po naravi dela iterativen. Če na določeno poslovno zahtevo ne moremo pripraviti odgovora, moramo razširiti prenos podatkov še z atributi, ki jih potrebujemo.

Po drugi strani pa Kimballov dimenzijski življenjski cikel (Kimball, Reeves, Thronthwaite & Ross, 1998) predlaga tak pristop, da se najprej osredotočimo na poslovne zahteve, ki so bile zajete v intervjujih in vprašalnikih, in tako izdelamo področna skladišča. Življenjski cikel se prične s projektnim planom, v katerega najprej zajamemo poslovne zahteve, nato definiramo model dimenzij, arhitekturo, fizično implementacijo, izdelavo, testiranje in nenazadnje distribucijo podatkov.

Kadar je potrebno izdelati podatkovno skladišče zelo velikega podjetja, ki je distribuirano čez več geografskih področij, je v praksi nepraktično ugotavljati in zbirati vse poslovne zahteve. V tem primeru lahko od uporabnikov pridobimo zahteve z uvedbo pristopa na osnovi prototipa, kjer bodo uporabniki dobili občutek, katere stvari bi dejansko lahko pridobili iz podatkovnega skladišča. Če je potrebno zgraditi dokaj individualna področna skladišča, se lotimo planiranja in razvoja po pristopu dimenzijskega življenjskega cikla. Tako se lažje osredotočimo na poslovne procese v transakcijskih sistemih, kot če bi izdelovali (podjetno) podatkovno skladišče, kjer bi morali zajete vse poslovne procese.

Imamo torej dva pristopa k planu in kasneje implementaciji – Inmonov obratni od pristopa SDLC (The Software Development Life Cycle (SDLC) For Small To Medium Database Applications, 2016) in Kimballov, ki zahteva konkretne poslovne zahteve (tj. dimenzijski življenjski cikel). Izbira pristopa je pomemben kriterij pri odločitvi za podatkovno skladišče, saj nam v grobem že sam pristop nakazuje, kako bomo planirali izgradnjo podatkovnega skladišča. Slika 16 prikazuje razvojni cikel programske opreme po pristopu življenjskega cikla programske opreme SDLC.

Slika 16: SDLC – razvojni cikel programske opreme



Vir: The Software Development Life Cycle (SDLC) For Small To Medium Database Applications, 2016, stran 4.

Pristop SDLC je zasnovan v šestih korakih (The Software Development Life Cycle (SDLC) For Small To Medium Database Applications, 2016):

- **projektno planiranje (angl. *project planning*)**: Določi se celoten projektni plan z roki, obsegom, izvajalci, stroški in potrditvenim testom,
- **opredelitev poslovnih zahtev (angl. *requirements definition*)**: Te so bile že določene z vprašalniki in intervjuji, zajete pa so v elektronski obliki in se bodo uporabljale kot referenca za testiranje podatkovnega skladišča in za potrditveni test,
- **načrtovanje (angl. *design*)**: Določimo design podatkovnega skladišča, področnega skladišča ali katerega izmed obeh pristopov po Inmonu ali Kimballu. Analizirati je potrebno vse vire in predstaviti podatkovni model podatkovnega ali področnega skladišča. Najkasneje na tem mestu je potrebno izbrati orodja, s katerimi bodo uporabniki pregledovali podatke v podatkovnem skladišču, in ugotoviti, katere zahteve so označene kot poslovne zahteve,
- **razvoj (angl. *development*)**: Razvoj vmesnikov za priključitev na različne vire, razvoj (več) procesov ETL, dopolnjevanje meta podatkovne baze s podatki o procesih ETL, razvoj vmesnikov za priključitev orodij za pregled podatkov v podatkovnem skladišču,
- **integracija in testiranje (angl. *integration and test*):** Pomemben korak, ko se mora podatkovno skladišče integrirati v okolje oziroma postati uporabno za uporabnike. Če imamo podatkovno skladišče, uporabniki pa ne znajo priti do njega, četudi je recimo orodje dostopno preko spletnega vmesnika, je podatkovno skladišče neuporabno in povečuje nezadovoljstvo uporabnikov,

- **namestitev in prevzemanje rešitve (angl. *installation and acceptance*):** Razvoj podatkovnega skladišča poteka v razvojnem okolju. Vsako izvajanje testa se mora dokumentirati in popisati postopke testiranja. Testi morajo biti ponovljivi in dati iste rezultate, če bi, recimo, gledali vedno isto obdobje. Potrditev rešitve pa pomeni, da se vsi uporabniki, tudi manjša skupina ključnih uporabnikov (angl. *Power Users*), strinjajo, da so rezultati v primerjavi s podatki na virih točni in relevantni. Potrditveni test je najbolj stresna situacija, saj se s tem potrjuje uspeh ali neuspeh izgradnje podatkovnega skladišča.

Kateregakoli izmed obeh pristopov vzamemo, velja pri njunem upoštevanju ohranjati zdravo mero razuma. Moja ugotovitev je, da, glede na literaturo, ki sem jo prebral, in na moje dvajsetletne izkušnje, se v praksi upošteva tisti pristop, ki je nekako vmes med obema. Če bi z barvno lestvico označili Inmonom pristop s črno barvo, Kimballov pa z belo, se v praksi vedno definira pristop, ki odraža eno od sivin. Upoštevati striktno samo enega ali drugega je teoretična predpostavka, da lažje določimo, kateremu izmed pionirskih pristopov se najbolj približamo. Imeti referenco, da upoštevamo Kimballov pristop, nima v poslovnem svetu tako pomembne veljave, kot jo ima za nas same, ko načrtujemo izgradnjo podatkovnega skladišča. Na osnovi literature lahko povzamemo, da vrsta podatkovnega skladišča v resnici pomeni, katero arhitekturo naj bi izbrali. To je dejansko res, ni pa popolnoma skladno z novejšo tehnologijo in novejšimi pristopi ter vrstami arhitektur sistemov podatkovne analitike.

Robinson (2004b) je že predstavil nekaj napotkov, s katerimi bi lahko projekt podatkovnega skladišča oziroma projekt podatkovne analitike pripeljali do učinkovite izvedbe. Poudaril je predvsem, da se moramo projekta lotiti s polno časovno zasedenostjo (lastno ali preko določitve vodje projekta), se redno posvetovati z drugimi vodji projektov in z intervjuji preverjati zadovoljstvo poteka projekta. Posebno je pomembno tudi zavedanje, da moramo neprestano obvladovati tri področja: arhitekturo, tehnologijo in poslovni model. Odločitev o izbiri in vpeljavi arhitekture mora biti posledično povezana z izbiro tehnologije, saj slednja določa orodja ter uporabo podatkovne analitike. Oboje pa mora biti vedno upoštevano za poslovni model oziroma poslovanje, za katerega se podatkovna analitika vpeljuje.

Načrt poslovne inteligence (Deliwala, 2010; Sen, Ramamurthy & Sinha, 2012; Knowlton, 2015) predstavlja praktična navodila, kako uspešno izvesti načrt, kaj za podjetje pomeni poslovna inteligenca in kako jo že v snovanju vključimo v poslovno strategijo. Predstavlja pa tudi težnjo, da je potrebno pojem poslovne inteligence vključevati v obstoječe in vse nove poslovne sisteme, katere ima podjetje namen še (do)kupiti. Načrt je povzeto sestavljen iz naslednjih korakov:

- merjenje obstoječega stanja (nivo okolja poslovne inteligence),
- postavitev podatkovnega slovarja,

- določanje ciljev in
- izdelava dolgoročnega izvedbenega načrta za dosego ciljev na zadanih področjih.

Za merjenje obstoječega stanja lahko uporabljamo preproste matrike, kjer definiramo, katera poslovna področja bodo imela določene funkcionalnosti. Pomembno je zavedanje, da vsaka sprememba obstoječega sistema, vsaka iniciativa ali konkretna uvedba novosti terja svoj plan in pripravljenost skupine, ki skrbi za poslovno inteligenco, da se uskladi s skupino, katera v podjetju skrbi za poslovni del. Obe skupini morata prav tako določiti cilje, ki morajo biti usklajeni s strategijo podjetja. Sprejeti morata tudi odločitev, kako na osnovi teh ciljev izdelati izvedbeni načrt; kateri vsi projekti na temo poslovne inteligence se bodo izvajali, da se dosežejo zadani cilji po različnih področjih. Podjetja se verjetno še velikokrat odločajo za uvedbo sistema podatkovne analitike brez dolgoročnih načrtov, saj potrebujejo hitre rezultate in lahko zapadejo v situacijo, ko niso pripravljena obvladovati velike količine zajetih podatkov na virih, ali so zahteve po novih analizah izven meja dobav obstoječe podatkovne analitike, kar pa meče slabo luč na celotno postavljeno infrastrukturo sistema za poslovno odločanje.

Na začetku tega poglavja sta omenjeni dve glavni tradicionalni strategiji izdelave podatkovnega skladišča, pred tem poglavjem pa še strategija, ki uporablja načrt poslovne inteligence. Zadnja leta v ospredje prihajajo potrebe po kompleksnih analizah na osnovi ogromne količine pridobljenih podatkov. Kako časovno predpostaviti, koliko stare (zakasnitev en dan ali nekaj dni) podatke imamo v podatkovnem skladišču na voljo za izvajanje analiz, je seveda odvisno od tega, na katere poslovne zahteve omenjeno podatkovno skladišče ali arhitektura sistema podatkovne analitike odgovarja s svojo vsebino.

Če dodamo vsaj še dejstvo, da porast dela s spletom in porast uporabe spleta za vse vrste poslovnih funkcij v današnjem času pravzaprav postavlja tiste prave realne zahteve podjetju, kakšno arhitekturo bi bilo najbolj izbrati, postane seveda jasno, da z obstoječo strojno in programsko opremo iz preteklih let ne bo več možno pokrivati vseh potreb čedalje bolj zahtevnih kupcev.

2 PREDSTAVITEV PROBLEMATIKE

2.1 Faze izdelave sistema podatkovne analitike

Izdelavo podatkovnega skladišča razdelimo v več glavnih korakov, ki se lahko še podrobneje delijo v pod-korake. Prvi, najbolj pomemben, korak se prične z definicijo poslovnih zahtev. Za pravilen začetek pa obstaja množica usmeritev, ki poleg Kimballa in Inmona predstavljajo svoje koncepte (Golfarelli & Rizzi, 1998).

2.1.1 Definicija poslovnih zahtev

Definiranje vseh poslovnih zahtev je tisti korak, od katerega so odvisni vsi nadaljnji koraki pri gradnji podatkovnega skladišča. V primeru, ko ne moremo določiti poslovne zahteve, pravzaprav ne moremo pričeti z gradnjo podatkovnega skladišča oziroma področnega skladišča. V tem primeru se moramo najprej vprašati o možnosti izvedljivosti takšnega projekta v podjetju. Po mojih izkušnjah je tudi v praksi to vedno prvi korak, zaradi katerega se je začelo govoriti o možnosti gradnje podatkovnega skladišča. Pri definiciji poslovnih zahtev za podatkovno skladišče je obvezno izpolniti naslednje korake oziroma jih vsaj upoštevati, kadar hočemo formalizirati poslovne zahteve (Haertzen, 2008a):

- preveriti cilje podjetja in usmeritve,
- določiti skupino uporabnikov,
- izdelati vprašalnike za celotno podjetje,
- izvesti delavnice oziroma predstavitve ter
- določiti obseg in velikost.

Te korake je potrebno formalno zapisati, saj se lahko le tako kadarkoli sklicujemo na vhodne predpostavke, ki so zajete v definiciji poslovnih zahtev. V praksi se je uveljavil pristop, da se pri testiranju podatkovnega skladišča najprej preveri, ali izhodni podatki uspejo odgovoriti na vprašanja, ki so bila določena kot poslovne zahteve. Z vprašalniki drastično povečamo produktivnost, določimo konkretne rešitve, hitre rezultate, dobro sodelovanje med »naročniki« in izdelovalci podatkovnega skladišča ter konec koncev tudi manjše stroške življenjskega cikla izdelave podatkovnega skladišča.

Ko določimo poslovne zahteve, kar pomeni, da opredelimo, katere podatke bomo pregledovali v podatkovnem skladišču, s tem hkrati določimo ali dopolnimo (Haertzen, 2008a):

- vizijo in strategijo podjetja,
- poslovne načrte,
- letna poročila,
- planiranje po področjih,
- pregled poslovanja,
- projektni plan podatkovnega skladišča ter
- specifikacijo za izhodne podatke podatkovnega skladišča.

Pri definiciji poslovnih zahtev moramo prav tako določiti skupino uporabnikov (Clarry, 2006), saj skupina managerjev analizira podatke na najvišjem nivoju, po drugi strani pa analitiki pregledujejo iste podatke na najnižjem nivoju. Tudi tipi vprašanj oziroma poslovne zahteve se razlikujejo glede na to, katera skupina jih pregleduje. Če jih

pregledujejo managerji ali direktorji, jih najbolj zanimajo agregirani podatki za celo regijo ali državo, analitike pa samo za določeno trgovino. Seveda je možno zelo enostavno prehajati med prikazom podatkov za nivo pregleda za managerja in za analitika, ampak vseeno moramo upoštevati najbolj podroben pregled, ki ga ni več smiselno bolj podrobno opisati. Primer je v tabeli 1, kjer je prikazan rezultat poslovne zahteve z vprašanjem: »Koliko je prodanih izdelkov, skupaj z njihovo skupno ceno, po državah«. Iz tega je razvidno, da določenega nivoja uporabnikov ne zanimajo podrobnosti, zato jih v rezultatih, ki so jih zahtevali, niti ne pričakujejo.

Tabela 1: Rezultat poslovne zahteve za nivo managerjev

Država	**Število izdelkov**	**Skupna vsota v evrih**
Slovenija	1.000	50.000
Avstrija	2.300	115.000
Nemčija	9.000	450.000
		=615.000

Na drugi strani pa ima prodajni analitik svojo poslovno zahtevo s podobnim vprašanjem: »Koliko je prodanih izdelkov, skupaj z njihovo skupno ceno, po državah, ceni na enoto izdelka in trgovini«. Rezultat je prikazan v tabeli 2.

Tabela 2: Rezultat poslovne zahteve za nivo oddelka analitičnih obdelav

Država	**Trgovina**	**Cena na enoto v evrih**	**Število izdelkov**	**Skupna vsota v evrih**
Slovenija	Trgo	49	800	39.200
	ABC	54	200	10.800
Avstrija	Schwann	50	2.000	100.000
	Turbo	50	300	15.000
Nemčija	Alzar	50	5.000	250.000
	Bergo	49	2.000	98.000
	CED	51	2.000	102.000
				=615.000

Oba rezultata v zgornjih tabelah sta v resnici odgovorila na eno in isto vprašanje, konkretno tudi na vprašanje analitične skupine, ki jo je zanimala prodaja po državah, trgovinah in po ceni na enoto izdelka. Bistvo zajema vseh poslovnih potreb je ravno v tem postopku – zajeti najbolj podrobna vprašanja, ki nas bodo še zanimala. Pri definiranju vseh možnih kombinacij podatkov se je sicer mogoče zanašati na pregled vseh možnih podatkov iz različnih virov, vendar to praviloma pomeni preveliko količino podatkov oziroma preveliko množico nepreglednih podatkov. V primeru, da je virov veliko, to posledično pomeni preveliko množico možnih poizvedb. V praksi se je zato izkazalo, da je najbolj učinkovito izdelati vprašalnike ali intervjuje za zaposlene s standardnimi vprašanji v obliki (Haertzen, 2008a):

- kaj pravzaprav pričakujete od pridobljenih podatkov,

- kako merite pridobljene rezultate,
- kaj je kritičen dejavnik uspeha pri vašem delu,
- kako lahko identificirate priložnosti in probleme,
- kateri podatki so za vas najbolj pomembni – izdelki, stranka, čas prodaje,
- katere vire podatkov imate ter
- s kakšnim namenom boste uporabljali podatkovno skladišče.

Skozi vsa ta vprašanja in definicijo zahtev tako hkrati obvladujemo še področje planiranja, vendar v smislu, kako bi lahko planirali glede na to, da bomo imeli podatkovno skladišče in kaj iz njega lahko pridobimo. Če nas zanima pregled poslovanja za dve leti za nazaj, hkrati pa želimo napovedati, kaj naj bi se ob določenih predpostavkah zgodilo v prihodnosti, je na tem mestu idealna priložnost, da se tudi podjetje konsolidira glede svoje vizije in še bolje – strategije podjetja. Pri definiciji poslovnih zahtev se izkoristi še priložnost, da se čim bolj avtomatizirajo letna poročila. Kvalitetno in strokovno postavljanje poslovnih zahtev zagotavlja tudi kvalitetno oblikovano letno poročilo.

Ko se preko delavnic in intervjujev zaposlenim približa dodana vrednost uvedbe podatkovnega skladišča, se vse zaposlene tudi navduši, da se potrudijo in izdelajo oziroma definirajo vsa možna vprašanja, ki jih zanimajo in bi zanje lahko podatke pridobili iz podatkovnega skladišča. Prepričan sem, da je velik poudarek vsakega projekta uvedbe podatkovnega skladišča na definiciji pravih vprašanj, saj tako lahko iz podatkovnega skladišča podamo prave in kvalitetne odgovore. Definiranju osnovnih zahtev sledi korak do analize virov in formatiranja objektov oziroma tabel.

2.1.2 Analiza vseh virov

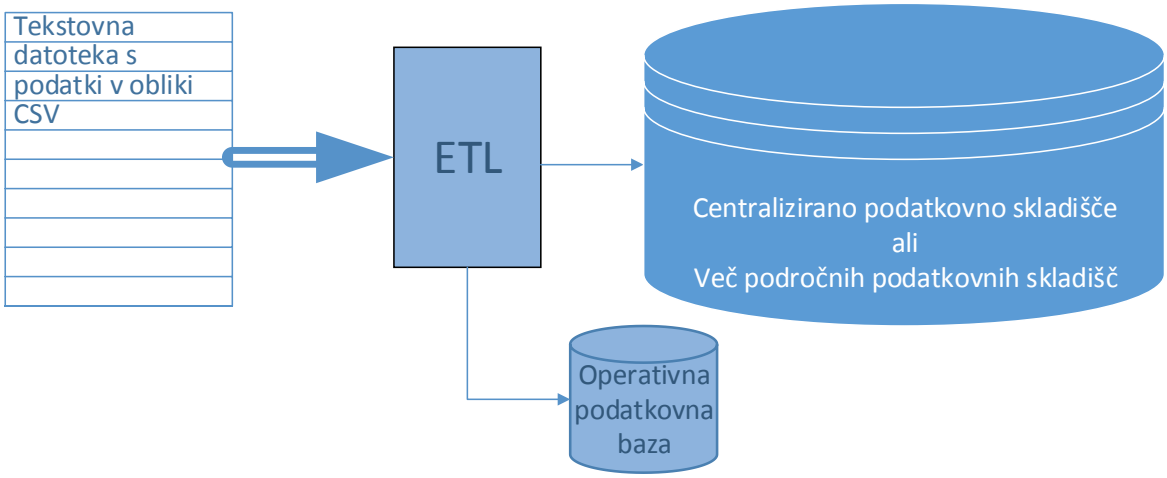
Podatkovno skladišče, še posebej centralizirano, vsebuje podatke iz več virov. Vir je določen kot podatkovni objekt, iz katerega bomo prečrpavali podatke preko določenih procesov v podatkovno skladišče. Tabela 3 predstavlja vir podatkov kot tabelo s stolpci (atributi), Slika 17 pa predstavlja vir podatkov v obliki tekstovne datoteke.

Na Slikah 17 in 18 je prikazano, kako lahko v začetnem procesu prenosa podatkov obstaja en ali več virov (tabela ali tekstovna datoteka). Ni nujno, da podatkovno skladišče kot ciljna struktura pomeni tudi več virov na začetnem delu procesa prenosa, saj je področno skladišče lahko tako kompleksno, da zajame več virov kot vhodno strukturo. Vsak vir je torej lahko v osnovi tekstovna datoteka, tabela ene izmed podatkovnih baz ali pa skupek tabel (pogled nad podatki). V nadaljevanju predpostavljam, da imajo vsi viri za prenos v podatkovno skladišče samo osnovni konstrukt podatka – tabelo.

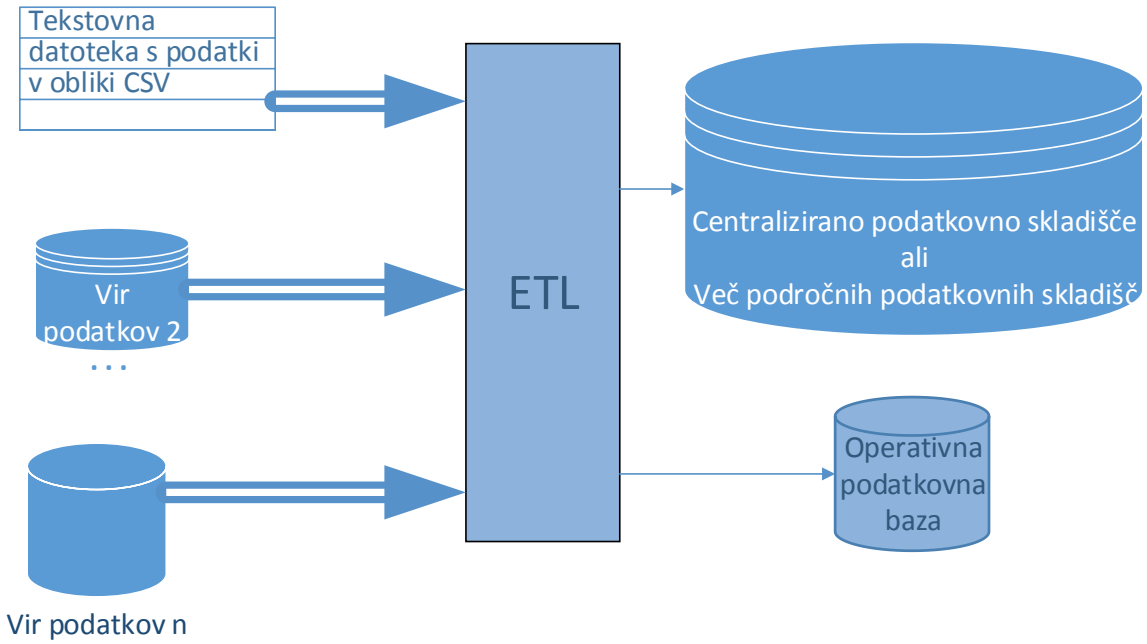
Tabela 3: Vir podatkov – tabela

Ime	Priimek	Spol	Datum_rojstva	Kraj_rojstva
Darko	Jagarinec	M	05.11.1976	Ljubljana
Ula	Jagarinec	Ž	15.08.2007	Ljubljana
Bernarda	Jagarinec	Ž	15.06.1976	Ljubljana

Slika 17: Vir podatkov – tekstovna datoteka



Slika 18: Več virov kot vhodna struktura



Vsak vir, ki bo kadarkoli uporabljen, je potrebno analizirati. To pomeni, da najprej preverimo, kaj pomenijo tabele v viru, npr. Račun, Kupec, Postavka, Kosovnica, Posta itd. Podatke moramo tudi vsebinsko pregledati, da dobimo pravo sliko, kaj lahko od vira pričakujemo. Za vsako polje v tabeli moramo preveriti podatkovni tip ter odgovoriti na

vprašanje: ali ga bomo lahko pretvarjali pri prenosu v podatkovno skladišče in katere napake lahko določeni podatki že vsebujejo. Vsako tabelo v viru je potrebno tudi formatirati in določiti, kakšna bo predvidoma oblika v izvozni datoteki. Na koncu tega koraka dobimo pravo sliko, kako je formuliran posamezni vir.

2.1.3 Standardizacija in čiščenje podatkov

Za prenos tabel iz virov je potrebno določiti povprečen čas prenašanja iz vira v podatkovno skladišče. Vsako polje v viru mora biti definirano z interno določenimi standardi v obliki dolžine znakovnih polj, formata številčnih polj, oblike datuma in kodne tabele znakov. Vsaka tabela v podatkovnem skladišču mora biti unikatno povezana z identifikatorjem tabele v viru, kar pomeni, da moramo imeti spisek vseh tabel, ki se iz vira prenašajo v ponor – podatkovno skladišče. Na tem mestu je, po izkušnjah sodeč, porabljenega največ časa, saj se v praksi pokaže, da sta ravno analiziranje virov in čiščenje podatkov najbolj kritični poti pri doseganju kakovosti podatkov v podatkovnem ali področnem skladišču.

Pokaže se namreč, da v praksi šele v tem koraku dejansko pregledamo podatke, ki jih imamo v virih in na tem mestu pravzaprav razumemo, kako aplikacija dejansko zapisuje podatke skozi svoje delovanje. Ta korak vsebuje tudi največji element presenečenja.

Okvirno poznamo več načinov čiščenja podatkov (Haertzen, 2008b):

- dopolnitev manjkajočega dela podatka,
- vpeljava domene za vrednosti podatka,
- formatiranje v enoten zapis datumskega podatka,
- odstranitev duplikatov podatka,
- obravnavanje »kvazi« podatkov,
- obravnavanje netočnih podatkov in
- iskanje zgrešenih kontekstov.

Dopolnitev manjkajočega podatka lahko obravnavamo šele pri samem prenosu podatkov iz virov v podatkovno skladišče. Če manjka ime, priimek, kraj rojstva, moramo določiti standarde, kakšno bo privzeto ime, priimek, kraj rojstva, ko le-ti na viru manjkajo. Za to odločitev moramo poznati tako vir kot tudi poslovne zahteve, ki so jih definirali uporabniki podatkovnega skladišča.

Vpeljava domene za vrednosti podatka je zelo klasičen pristop, kadar imamo dve možnosti za vrednost podatka. Dober primer je spol, ker lahko nastopi v obliki 'M' in 'Ž', lahko v obliki 'm' in 'ž' in seveda množici kombinacij. Če uspemo napisati pravilo, kako naj se vedno preslikajo vse kombinacije vrednosti v samo dve vrednosti M in Ž, moramo določiti posebno formulo, ki obravnava vhodne podatke in vrne vedno samo dve vrednosti M in Ž.

Formatiranje v enoten zapis datumskega podatka je iskanje oblike datuma, ki bo enotna za vsa datumska polja v arhitekturi. V kolikor se odločimo za format »DD.MM.LLLL«, kjer DD pomeni dan, MM mesec, LLLL pa leto, potem morajo biti vsi datumi v taki obliki. To je s stališča obvladovanja podatkov poglavitnega pomena, da ne pride do zamenjav meseca in dneva.

Odstranitev duplikatov podatka je praviloma matematična operacija, ki se pojavi pri čiščenju podatkov iz virov. Sama identifikacija podatkov je vedno unikatno določena, prav tako tudi vsi kriteriji, ki nakazujejo, da je kateri od podatkovnih zapisov duplikat, kar pomeni, da se je po kriterijih določen zapis ponovil več kot enkrat. V primeru, da je duplikatov več, govorimo o ponovitvi n-teric. V procesu transformacije podatkov vse take duplikate identificiramo in jih s poslovno logiko, ki jo določimo ob pojavitvi primerov, tudi obravnavamo. V kolikor je naša poslovna logika, da take zapise pobrišemo iz nabora podatkov, potem to tudi storimo in zapis ustrezno označimo kot duplikat (ali n-terica, če sta več kot 2).

2.1.4 Normalizacija tabel

Tabele na viru se v mnogih primerih splača normalizirati, kar pomeni, da jih spravimo v obliko, ki bo najbolj primerna za najučinkovitejši prenos, hkrati pa zagotovimo, da se ne izgubi vsebina ali pomen zapisanih podatkih.

Večji del tega koraka odpade na izdelavo programske kode za takšno normalizacijo. Na tem mestu se lahko odločimo tudi za izdelavo abstraktnih skupin, ki naj bi bile, glede na svoj pomen, med seboj tesneje povezane kot ostale. Primer bi lahko utemeljili z najbolj klasičnim primerom trgovine. Stranka in njen račun abstraktno spadata skupaj, račun in postavke računa pa abstraktno skupaj – določevanje tovrstnih abstraktnih skupin ni lahko delo in je potrebno že kar nekaj prakse za utečeno delo. Zelo pa nam pri tem pomaga dobro poznavanje virov.

2.1.5 Proces ETL

Ta proces bi lahko poimenovali kar osnovni proces pri arhitekturi sistema podatkovne analitike, saj vsebuje bistvene in najbolj vitalne dele analitike – zaradi svoje pomembnosti, vsebine ali kritičnosti (ne)delovanja (Earls, 2003; Sen & Sinha, 2005).

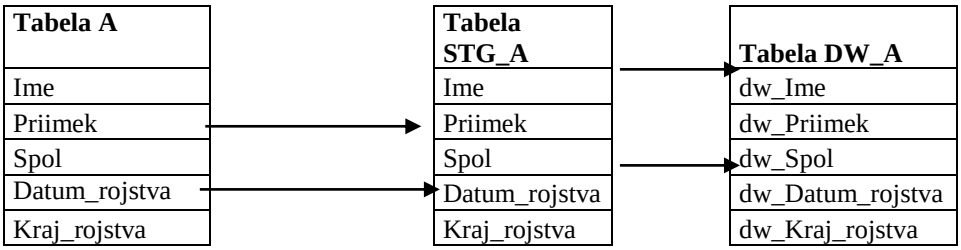
Vsi trije koraki: analiza virov, standardizacija in čiščenje podatkov ter normalizacija tabel so torej imenovani kot proces ETL. Glede na moje izkušnje je jedro učinkovitega podatkovnega skladišča vedno v kakovostnem procesu ETL. Doseči kvaliteten, učinkovit, ponovljiv ter obvladljiv proces ETL je pa vse prej kot lahko. Moja ocena temelji na tem, da je delo s procesi ETL približno dve tretjini vsega dela, ki ga moramo opraviti v implementaciji podatkovnega oziroma področnega skladišča, ob tem pa ne upoštevamo predstavitvenega nivoja oziroma orodij za delo s podatkovnim oziroma področnim

skladiščem. Moj namen je predvsem poudariti ta korak, ki je dovolj zgodnja kretnica, ali projekt implementacije podatkovnega skladišča poteka v pravi smeri ali pa bo potrebno ponovno evaluirati (oceniti) izvedljivost projekta.

Na Sliki 18 je prikazano več virov – vse do procesa ETL, ki je v osnovi lahko zapleten proces, lahko pa je enostavno mapiranje atributov iz tabele v viru v tabelo na ponoru. Dejstvo je, da sam proces ETL obvladuje tisto najbolj pomembno točko podatkovnega skladišča, ki je najbolj kritična ter bi ji morali nameniti največji del, tako časa kot truda in energije.

Slika 19 prikazuje enostaven proces ETL, kjer je prikazan prenos podatkov iz tabele A v viru v tabelo STG_A v operativnem delu podatkovnega skladišča. Tabela STG_A praviloma še ni objekt podatkovnega skladišča, ampak jo trenutno odložimo v nek vmesen prostor, saj bomo z njo še izvajali različne operacije (angl. *staging*).

Slika 19: Enostaven proces ETL



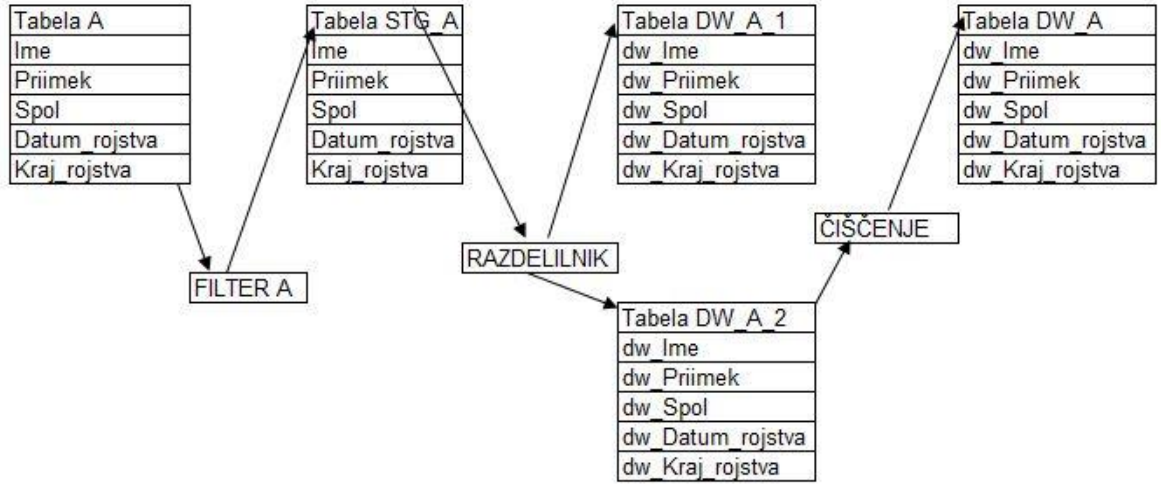
V praksi tako enostavnih procesov ETL ne bomo zasledili, razen pri enostavnih šifrantih, pa še to je zelo malo verjetno. Potrebno je namreč slediti spreminjanju vrednosti šifrantov, saj je ključnega pomena, da poznamo dogajanje skozi zgodovino. Poznati spreminjanje vrednosti šifranta skozi zgodovino pa pomeni, da lahko za tak šifrant uvedemo nov termin – dimenzija podatkovnega skladišča. Več o tem kasneje v predstavitvi vrst podatkovnih skladišč.

Slika 20 prikazuje malce bolj kompleksen proces ETL, kjer je vključenih več tabel v viru, obenem pa se izvede več operacij nad tabelami v operativnem delu podatkovnega skladišča. Pri tem je pomembno, da zelo dobro poznamo in argumentiramo, zakaj smo določeno stvar narediti tako, kot smo jo, saj je vzdrževanje slabo opisanega procesa ETL zelo težko opravilo. Delo samo po sebi ni težko, je pa težko razumeti razmišljanje predhodnika, še posebej, ker je orodje za izdelavo procesa ETL v grafični obliki, kar pomeni, da več ali manj samo rišemo na procesno ploščo in povezujemo objekte z operacijami. Glede na moje izkušnje priporočam, da proces ETL vedno narišemo ročno na list papirja, nato pa razmislimo, kaj proces ETL sploh pomeni, kako se obnaša v robnih primerih in kako hitro deluje. Brez testiranja procesov ETL bi bila implementacija vsakega

podatkovnega skladišča obsojena na povečanje procenta statistike neuspelih implementacij. V praksi se pogosto srečujem s situacijo, ko se proces ETL izdela v relativno kratkem času (odvisno od obsega projekta in števila ur za izdelavo procesa ETL). Večji del časa moramo nameniti testiranju procesa ETL, kar pa je vse prej kot lahko in zanimivo delo.

Testiranje ETL je pomembno predvsem zato, ker je lahko nepravilno in neučinkovito delovanje procesa ETL povod za zaustavitev projekta implementacije podatkovnega skladišča. Zelo preprosto rečeno: če se podatki narobe prenašajo iz virov v podatkovno skladišče, je ves trud, čas in denar zaman, vsekakor pa ne zastonj.

Slika 20: Kompleksen proces ETL



2.1.6 Metapodatki

Opis virov – atributov tabel v virih, povezavah, ključih na tabelah in med tabelami ter opis objektov v operativnem in kasneje v podatkovnem skladišču ne morejo ostati nezapisani. Stvar namreč pri desetih tabelah in enem viru mogoče še obvladujemo, ko pa imamo deset virov in 500 tabel, veliko procesov čiščenja podatkov, formatiranje podatkov, pa je nujno potrebno to spraviti v elektronsko obliko, ki jo lahko obvladujemo in kontroliramo.

V ta namen izdelamo metapodatkovno bazo oziroma repozitorij (angl. *metadata repository*), podatkom v takem repozitoriju pa pravimo kar metapodatki, kar bom privzel v nadaljevanju. V praksi se vedno pokaže, da v primeru, ko nimamo metapodatkov (pravimo jim tudi podatki o podatkih), nimamo pravzaprav ničesar. Metapodatki so tisti podatki, ki nam opišejo, kaj podatkovno skladišče sploh vsebuje, hkrati pa so to podatki, s katerimi arhitektura sistema podatkovne analitike dobiva svojo moč.

Jasen primer je v tabeli 4, kjer je prikazan opis vsakega podatka v viru. Brez takih opisov po procesu prenosa podatkov ne bi bilo več razvidno, kateri podatek v ponoru pomeni

določen podatek v viru. Za tabelo STRANKA je najprej opis same vsebine tabele STRANKA, nato pa podroben opis vseh atributov te tabele. Če bi imeli stolpca Ime in Priimek samo v tabeli STRANKA, ne bi prišlo do zmede. Do pravega kaosa pa pride, če bi imeli še podobne tabele z imeni, recimo tabela DOBAVITELJ, kjer bi tudi imeli ime in priimek dobavitelja. Že ta primer bi lahko postal dvoumen, če bi hoteli pri čiščenjih podatkov obravnavati obe imeni in oba priimka, neko določeno akcijo (pretvorba malih v velike črke) pa bi izvajali v napačni tabeli.

Tabela 4: Metapodatki tabele na viru

Identifikator	**Format**	**Opis**
Tabela STRANKA		Tabela vseh strank
Ime	CHAR(35)	Ime stranke
Priimek	CHAR(35)	Priimek stranke
Spol	CHAR(1)	Spol stranke
Datum_rojstva	DATE	Datum rojstva stranke
Kraj_rojstva	CHAR(70)	Kraj rojstva stranke (opis iz identifikacijskega dokumenta)
Obcina_stpr	NUMBER	Šifra občine stalnega prebivališča stranke
Oddaljenost_km	NUMBER	Približna oddaljenost stranke od trgovine, kjer je pristopila v klub
Vpisno_mesto	NUMBER	Šifra trgovine, v kateri je stranka pristopila v klub
...	...	...

ETL je v celoti fizično predstavljen z metapodatki. Dejansko ETL poveže med seboj metapodatke tabel vira, operativne podatkovne shrambe in podatkovnega skladišča. Kodni zapis procesa je torej sestavljen le iz opisov in definicij, ki smo jih skrbno zapisali v metapodatkih. To je pri iskanju napak v procesu ETL nepogrešljiva stvar.

2.2 Napake in problemi pri izvedbah

Več avtorjev v svojih monografijah in člankih poudarja tipične napake, ki se zgodijo v življenjskem ciklu podatkovnega skladišča, splošno tudi v katerikoli arhitekturi podatkovne analitike. Življenjski cikel pomeni, da imamo pregled nad celotnim življenjskim ciklom neke aplikacije – od začetka do konca, torej od poslovnih potreb do ponovne spremembe teh. Napake pa niso vedno nujne samo v življenjskem ciklu, lahko se zgodijo že pri izbiri neprimerne arhitekture podatkovne analitike (Gutiérrez & Marotta, 2000; Gowan, Mathieu & Hey, 2006). Nekateri avtorji (Lin, Hsu & Sheen, 2007) so že v preteklosti omenjali, da pravega načina in kriterija, s katerima bomo zagotovo izbrali najbolj ustrezno podatkovno analitiko oziroma arhitekturo, ne bo moč vedno deterministično izbrati, zato je iz članka sklepati, da vedno obstaja mehka logika pri odločitvi za orodja in arhitekture. Drugače povedano: pravilna in najbolj ustrezna arhitektura sistema podatkovne analitike še ne prinese zagotovljenega uspeha pri implementaciji.

Sen in Sinha (2005) sta poudarila tipične napake, pri katerih se implementacije podatkovnih skladišč ustavijo ali iščejo novo pot, kako se izkopati iz težavnih situacij, ko nadaljnje delo ne pomeni več izziva ali celo postane preveč kompleksno. Poleg kritičnih procesov, kot so: integracija končnih rešitev, uglaševanje in optimizacija vseh procesov v procesu polnjenja podatkovnega ali področnega skladišča, je izrednega pomena tudi vzdrževanje podatkovnega skladišča. Če nismo sposobni dobro voditi in obvladovati vzdrževanja, so na vidiku težave pri prvi spremembi – ali vira ali poslovne zahteve. Zagotovo drži, da je polnjenje podatkovne baze izpostavljeno stalnim spremembam in da brez sprememb ter prilagajanja spremembam podatkovno skladišče na dolgi rok ne more učinkovito delovati.

Robinson (2004a) je tako definiral sedem napak pri implementaciji podatkovnega skladišča:

- izogibati se je potrebno programski kodi, katere sprememb ni moč hitro uvesti,
- ne priporoča se uporaba okornih programskih vmesnikov API (angl. *Application Program Interface*, v nadaljevanju API),
- ne načrtuje se ničesar, kar kasneje v razvojnih aktivnosti ni mogoče razširiti,
- ne vriva se ničesar nepotrebnega v dialog med končnim uporabnikom in podatki,
- ne uporablja se bližnjic pri čiščenju podatkov in analizi virov,
- potreben je temeljit razmislek o granularnosti in particioniranju podatkov ter
- ne priporoča se uporaba orodij OLAP brez analize poslovnih zahtev.

1. Izogibati se je potrebno programski kodi, katere sprememb ni moč hitro uvesti.

Aplikacije, ki jih bomo vpeljevali v življenje organizacije, bodo namenjene za analitične namene oziroma obdelave in ne za transakcijske namene – za slednje se uporabljajo transakcijski sistemi. V praksi se žal pojavlja percepcija informatikov in poslovnih uporabnikov (ki večinoma nimajo vrhunskega znanja s področja informatike), da informatiki smatrajo, da so vsi ostali uporabniki vsaj na enakem nivoju poznavanja informatike, kot so sami. Na tem mestu se torej srečata iskanje rešitve oziroma produkt projekta in poznavanje podatkov. Več iteracij je potrebno, da poslovni svet, ki ni vešč informacijskih prijemov in podatkov, razume, kaj želijo podatkovni analitiki narediti s podatki. Tukaj trčita poslovni svet in svet informatike. Zato je pomembno, kako se informatiki lotevajo tako programiranja kot izdelave raznih poročil. Poslovni svet jih mora razumeti in, kar je najbolj pomembno, znati uporabljati.

2. Ne priporoča se uporaba okornih programskih vmesnikov API.

Področje, ki je zelo spolzko v svetu informatike in programiranja, je tudi način definiranja integracijskih povezav med različnimi sistemi. Tem integracijskim vmesnikom, ki povezujejo različne informacijske in ostale poslovne sisteme med seboj, pravimo

programski vmesnik aplikacije API. Izrednega pomena za te programske vmesnike aplikacije je njihova sestava, torej struktura klicanja in vračanja rezultatov. Tak vmesnik mora v današnjih časih biti tako hiter, da zmore obvladovati ogromno število zahtev oziroma programskih klicev, saj se od določenega vmesnika pričakuje skorajda takojšen rezultat. Če gre za primer, ko mora tak API vrniti več kot npr. tisoč zapisov (če je take narave), se mora taka priprava rezultata izvesti zelo hitro, skorajda hipno, saj se podatki verjetno še dalje obdelujejo v procesiranju podatkov.

3. Ne načrtuje se ničesar, kar ne da kasneje v razvojnih aktivnosti mi mogoče razširiti.

Podatkovne analitike in njihovo razmišljanje ne moremo enačiti z razmišljanjem in koncepti delovanja transakcijskih sistemov. Bistvo podatkovnih analitikov je še vedno, kako v množici podatkov najti zakonitosti, ki jih na oko ne opazi nihče; kako najti povezave med različnimi podatki in kako najti neke potencialne težave – morda v prihodnosti poslovanja podjetja. Iz vsega sklepamo, da je podatkovna analitika zelo pomembna za obstoj podjetja ter opazovanje poslovanja, trendov prodaje ipd. Samo arhitekturo sistema podatkovne analitike, kakor tudi pravi tip podatkovnega skladišča ali druge hrambe podatkov, začrtamo v osnovi začetka projekta. V nobenem primeru pa z izbiro arhitekture ne moremo mimo dejstva, da bo potrebno v življenjskem ciklu arhitekture dodajati novonastale atribute iz transakcijskih sistemov (virov) ter nove povezave med samimi poizvedbami (angl. *joins*).

4. Ne vriva se ničesar nepotrebnega v dialog med končnim uporabnikom in podatki.

Predpostavljati moramo, da je uporabnikovo znanje naravnano in pričakovano v smeri, da samostojno vrta v podatke v globino do nivoja, ki ga za posamezno analizo potrebuje. Pri raziskovanju podatkov ne smemo uporabnika obremenjevati z njegovimi odločitvami pri vrtanju v podatke na način, da mu odpremo dialog komunikacije z vprašanji ter od njega zahtevamo, da se odloči, katero povezavo med podatki bo uporabljal (če jih je na voljo več). V izogib temu vedno vnaprej pripravimo set podatkov, v katerem lahko uporabnik, glede na hipne odločitve, kako priti do želenih podatkov, transparentno brska po njih in se ne obremenjuje s tem, ali bo izbral pravilno povezavo med ključi različnih tabel. Seveda so to želene lastnosti uporabnikov, ki brskajo po podatkih v katerikoli arhitekturi sistema podatkovne analitike. V kolikor se vseeno pojavijo take situacije, je bolje pripraviti več različnih okolij ter jih pripraviti namensko – vsako področje za določeno vrsto analitike.

5. Ne uporablja se bližnjic pri čiščenju podatkov in analizi virov.

Ena izmed najbolj časovno potratnih nalog je analiza virov za procesiranje ETL in čiščenje podatkov, preden jih prenesemo v sistem podatkovnega skladišča. Popolnoma normalno je pričakovati, da bo vodja projekta za to fazo namenil več kot polovico časa oziroma resursov. Če smo malce ostri – če si vodja projekta pri tej nalogi izbere bližnjico, kar

pomeni, da preprosto podcenjuje efekt slabo analiziranih virov, lahko to odločitev kasneje v izvedbi projekta bridko obžaluje. Zlato pravilo pravi, da ne smemo biti površni in moramo podrobno analizirati podatke, ne glede na to, kako dolgočasno je to delo.

6. Potreben je temeljit razmislek o granularnosti in particioniranju podatkov.

Dva večja izziva pri hranjenju podatkov fizično v podatkovni bazi arhitekture sistema podatkovne analitike sta nedvomno: pravilna umeščenost podatkov na pravem nivoju granularnosti (drobitve podatka) in particioniranja. Granularnost pomeni teoretično nivo oziroma skupni imenovalec podatkov, za katere se bomo odločili, da jih polnimo v arhitekturo sistema podatkovne analitike. To pomeni sprejem odločitve, da bomo vse podatke spremljali s 15 minutnim intervalom ali pa na nivoju trgovine, države in podobno.

Kar zadeva samo particioniranje pa je pomemben predvsem razmislek, ali bomo lahko ogromne količine podatkov logično ločevali med seboj – glede na nastanek. Dober primer je prodaja nekega artikla po mesecih, kjer bomo za podatkovno shranjevanje uporabili particioniranje glede na mesec. To zelo grobo na splošno pomeni, da bi lahko zelo hitro pridobili podatke za določen mesec in se tako izognili brskanju po podatkih čez celo leto. To opravilo je za podatkovne analitike prava »nočna mora«, saj je slišati marsikateri bridek komentar, ko se opravlja to med opravili najbolj težavno delo. Dober razmislek v tej točki projekta kasneje prinaša navdušujoče rezultate brskanja po podatkih v kompleksnih analizah. Tega koraka ne smemo nikakor izpustiti – za nobeno ceno.

7. Ne priporoča se uporaba orodij OLAP brez analize poslovnih zahtev.

V informatiki se žal zgodi tudi situacija, ko (končni) uporabniki dejansko ne vedo, kaj pridobiti iz podatkovnega skladišča, dokler ga ne vidijo v živo ter se ne prepričajo, kako je dejansko podatkovno skladišče videti. Takrat je lahko že prepozno, uporabniki ne bodo uporabljali podatkovnega skladišča in usoda slednjega je zapečatena. Veliko raznovrstnih poskusov brskanja po podatkih, tudi napačnih, bodo morali izvesti uporabniki, ki bodo poskušali pravo vrednost podatkovnega skladišča. Le tako bodo sproti lahko sodelovali in sooblikovali potrebe po podatkih ter razmišljati o končnih analizah, za katere bodo uporabljali te podatke. Poleg uporabnikov so tukaj na preizkušnji tudi zaposleni v oddelku informatike (angl. *Information Technology*, v nadaljevanju IT), saj se bodo morali z uporabniki podatkovnega skladišča toliko zbližati in jih poslušati, da bodo dejansko resnično začutili, kaj uporabniki želijo delati s podatki in kako jih želijo vključiti v svoje analize. Sodelovanje na tem mestu je ključnega pomena – oddelek informatike mora razumeti tako podatke, kot tudi poslovne zahteve uporabnikov. Streznitev osebja informatike sledi tudi v dejstvu, da ne vedo vsega, kar zadeva podatke v podatkovnih bazah vira, saj informatiki dejansko ne razmišljajo o logičnih izpeljavah s povezavami med več viri podatkov. Praviloma morajo informatiki na tem mestu spraševati uporabnike, kaj želijo pridobiti iz podatkov, kako si jih želijo prikazati in imeti distribuirane, tudi kako

pogosto in še mnogo več. Torej sodelovanje uporabnikov in informatikov je ključnega pomena pri razumevanju podatkov ter pričakovanih analiz.

Raziskava (Gorla, 2003) je preprosto prikazala, kakšna je razlika pri uporabi OLAP glede na znanje IT uporabnikov. OLAP se glede na to raziskavo osredotoča predvsem na ROLAP (angl. *Relational OLAP*) in MOLAP (angl. *Multidimensional OLAP*). V ROLAP so podatki agregirani in shranjeni v klasični relacijski bazi, v MOLAP (Wu & Buchmann, 1997) pa so podatki očiščeni, agregirani v več dimenzijah ter preneseni v kocko (angl. *data cube*). Zato so glaven razlike med tema dvema uporabama strnjene v naslednja priporočila:

- pri izbiri za manj izkušene uporabnike izberite MOLAP, za bolj izkušene pa ROLAP,
- uporabniki, ki bodo samo zaganjali poročila, naj uporabljajo MOLAP; tisti, ki pa bodo detajlneje vrtali v podatke, pa ROLAP,
- če so preneseni podatki več ali manj malo spremenjeni, se bolj splača uporabiti MOLAP; v primeru velikih sprememb pa je ROLAP bolj fleksibilen za dopolnjene poizvedbe (SQL),
- v začetkih projektov uvedbe sistema podatkovne analitike je priporočljivo začeti z MOLAP, kasneje – ob bolj kompleksni uporabi, pa je ROLAP prava izbira.

Glede na osebne izkušnje sem se srečal še z dodatnimi omejitvami projektov podatkovnih skladišč, ki bi jih lahko kategoriziral na naslednje pomanjkljivosti oziroma bodoče izzive:

- neučinkovita uporaba: Uporabniki sistema podatkovne analitike skladno s svojim znanjem uporabljajo podatkovno skladišče podjetja za brskanje po podatkih, ki jih potrebujejo – bodisi za svoje lastne analize bodisi za podporo novim projektom, ki potrebujejo za osnovo analizo preteklih podatkov. Ne smemo seveda pozabiti tudi na dejstvo, da se uspešnost npr. prodajnih akcij prav tako spremlja na osnovi že pridobljenih in zapisanih podatkih za nazaj, torej izvajamo analizo uspešnosti celotne reklamne akcije.

  Pri tem se lahko zgodi, da uporabniki ne poznajo moči podatkovnega skladišča, katerega imajo lahko vzorno izdelanega s kopico podatkov za analize, ali pa ne znajo rokovati s takim sistemom. Tekom projekta uvedbe ali izdelave ne smemo nikoli pozabiti na končne uporabnike (angl. *end user, power users*), ki so dejansko odločevalci, ali bo sistem podatkovne analitike zaživel ali ne. Pri svoji praksi sem vedno vključeval vse vrste uporabnikov v projekt podatkovnega skladišča, v vsakem primeru tudi pri pripravi in izbiri arhitekture sistema podatkovne analitike, saj mora biti podjetje dovolj dobro informacijsko osveščeno ter podkovano. Za slednje nam lahko bolj malo pomaga dobra odločitev in idealna izbira arhitekture sistema podatkovne analitike, ko kasneje v podjetju ni mogoče speljati take oblike shranjevanja podatkovne analitike. Naj ilustriram to s primerom, ko se npr. v arhitekturi odločimo za vpeljavo particioniranja (angl. *partitioning*) podatkov na nivoju meseca, za katerega spremljamo prodajo vseh artiklov v Sloveniji. Kaj malo nam ta, sicer zelo močna funkcionalnost,

lahko pomaga, če pozabimo na dejstvo, da so take opcije in funkcionalnosti podatkovne baze še posebej drage, saj so plačljive.

- nihče ga ne uporablja: Poleg dejstva in nesrečne situacije, da se sistem podatkovne analitike izkaže za neučinkovitega, lahko slednje vodi v popolno izolacijo uporabe sistema v podjetju. Uporaba je v podjetju lahko sicer predpisana, vendar uporabniki lahko za svoje naloge uporabijo tudi druge načine, kako priti do podatkov, npr. vrtilne tabele v Excelu, ki jim jih je sodelavec v oddelku informatike kolegialno pripravil. Tako na prvi pogled vodstvo dobiva prave podatke in prave usmeritve, ne zaveda pa se, da podatkov željni analitiki pridobivajo te po neformalnih poteh. Seveda se past skriva v tem, da se pravilnost podatkov potem zaupa nepreverjenim in nepotrjenim poizvedbam iz podatkovnih baz. Na osnovi lastnih izkušenj sem to doživel že večkrat, ko sem bil v vlogi pripravljavca podatkov, ki so jih analitiki nujno in takoj potrebovali za hitre analize, ocene stanj in podobno. Kar se je izrodilo nazadnje, je bilo to, da so ti podatki potem postali bolj kredibilni od tistih, ki so bili pripravljeni v sistemu podatkovne analitike, kateri je bil za podjetje precej draga investicija.
- počasno delovanje: Počasno delovanje izvedbe in pridobivanja podatkov lahko nastane iz več razlogov:
  - neustrezna ali zastarela strojna oprema (angl. *hardware*),
  - neučinkovito določeni indeksi v podatkovnih strukturah ter
  - neučinkovito napisane poizvedbe (angl. *queries*).

  Za prvo alinejo ugotavljam, da se na osnovi cenovno dostopne strojne opreme redko dogaja, da bi bili sistemi podatkovne analitike glede zmogljivosti podhranjeni. V daljšem časovnem roku pa je možno, da so podatkovni mediji že dodobra polni, konkretneje – trdi diski so že na več kot 90% zasedenosti svoje kapacitete; izpadi se pojavljajo zaradi iztrošene ostale opreme (napajalnik, grafična kartica), kar seveda ni moč planirati.

  Neučinkovito določeni indeksi v podatkovnih strukturah so praviloma bolj kompleksna težava kot strojna oprema. Potrebno je namreč podrobno poznavanje podatkovnega modela, saj se le tako lahko postavijo učinkoviti indeksi na tabelah, torej taki, ki se uporabljajo v poizvedbah končnih uporabnikov sistema podatkovne analitike. Način in tematika pravilne uporabe pa presegata okvire tega dela, zato na tem mestu samo omenjam, kako pomembno je, da ima sistem podatkovne analitike pravilno dizajnirane indekse na podatkovnih strukturah. Počasnost delovanja torej izvira iz počasnega delovanja poizvedbe, ki zaradi manjkajočega ali nerodno izdelanega indeksa tega ne uporablja – kljub dobri veri, da se uporablja. Rezultat tega je čakanje na rezultate poizvedb, analiz ipd, kar se kaže v zelo znanem nezadovoljstvu uporabnikov s »spet nam vse dela počasi«. To čakanje pa je lahko omejeno na nekaj sekund ali minut, v boljših sistemih podatkovne analitike pa sistem že prej javi presežen čas čakanja na podatke iz podatkovne baze (angl. *timeout*).

Neučinkovito napisane poizvedbe lahko izvirajo iz nepoznavanju podatkovnega modela sistema podatkovne analitike, lahko pa so posledica pomanjkanja poznavanja tehnično definiranih stavkov poizvedovanj (SQL), oboje pa seveda vodi v slabe rezultate. Obstajajo nekatere poizvedbe, ki jih tudi s poznavanjem tako pisanja poizvedb kot podatkovnih struktur ne moremo napisati bolje, hitreje, učinkoviteje. Kot primer navajam poizvedbo med več kot tremi tabelami, ko je v vsaki nekaj sto milijonov zapisov. Prvo vprašanje, ki se poraja v takih izrednih primerih, je vprašanje o ustreznosti postavitve dotičnega podatkovnega modela za tovrstne primere. Slednje pomeni, da je potrebno ob dizajniranju in arhitekturi sistemov podatkovne analitike upoštevati tudi skalabilnost celotnega sistema. Tako lahko sistem podatkovne analitike deluje brez posebnosti, kadar vsebuje podatke za nekaj mesecev, odpove oziroma postane časovno neučinkovit pa takrat, ko vsebuje zgodovinske podatke za nekaj let.

Neučinkovito delovanje sistema podatkovne analitike in nepoznavanje vsebine tvori skupino uporabnikov, ki zaradi premajhnega znanja informatike ali premajhne računalniške pismenosti ne zna pravilno uporabljati sistema podatkovne analitike, ne glede na to, kakšne vrste arhitektura je bila izbrana kot osnova. V kolikor uporabniki ne razumejo koncepta delovanja in možnosti ter omejitev podatkovne analitike, kmalu postanejo podatki oziroma analize neuporabne in posledično, zaradi nepoznavanja uporabe in rabe sistema podatkovne analitike, ta prične izgubljati svojo vrednost ter namen, zaradi katerega je bil uveden. Sam sem srečal že vsaj dve zgodbi, ki sta se končali na ravnokar predstavljen način.

- nepravilna izbira orodja: OLAP oziroma sama izbira orodja OLAP bi v prvi vrsti morala biti v domeni končnih ali vsaj ključnih uporabnikih. Pri izbiri orodja OLAP je potrebno biti izredno pazljiv, saj se določena orodja OLAP zelo osredotočajo na samo arhitekturo podatkov, z drugimi besedami: posamezna orodja OLAP so prirejena za določene podatkovne zbirke. To samo po sebi ni slabo, saj se v tem primeru proizvajalci orodij OLAP lahko osredotočajo na vsebino in dodano vrednost orodja OLAP ob tem, da podatke oziroma podatkovno bazo, zaradi lastništva, obvladujejo do potankosti.

Sem in Atish (2005) glede vzdrževanja arhitekture izpostavljata, da se, v kolikor podjetje nima ekipe in znanja, ki bi nadgrajevala arhitekturo in izpolnjevala nove poslovne potrebe po poročilih, se sistem podatkovne analitike sčasoma preneha uporabljati.

Pri svojem delu sem prevzemal vlogo sistemskega vzdrževalca podatkovnega skladišča, ki je bilo zgrajeno ali po konceptu Inmona ali po konceptu Kimballa. V obeh primerih sem vedno ugotavljal pomanjkanje navodil, napotkov ter ustne predaje nadgrajevanja sistema podatkovne analitike in sem, kljub velikim denarnim vložkom v omenjeni sistem, ostal brez takrat precej vrednih administratorskih navodil. V resnici mi je preostala samo lastna iznajdljivost, kako določena poročila predpripraviti ter raziskati programske kode

(procedure, izvorne kode že obstoječih enostavnih poročil), vedno z željo zadovoljiti potrebe po poročilih.

Z vzdrževanjem imam seveda v mislih dejstvo, da se vzdržuje vsaj dvoje opravil. Eno je redno, drugo je poslovno usmerjeno. Redno vzdrževanje pomeni vzdrževanje sistema, da se vse aktivnosti (redne naloge za prenašanje podatkov iz virov v ponor) izvajajo brez težav; poslovno orientirano pa pomeni, da smo sposobni, glede na arhitekturo, izdelati nova poročila na osnovi obstoječih podatkov. V primerih, ko je potrebno v sistem podatkovne analitike dodati nov vir podatkov, pa moramo imeti še znanje o dodajanju novih virov v proces ETL in jih znati povezati z obstoječimi. Taka opravila so za delovanje z malce nerodnosti lahko precej usodna.

Laurent (2001) pa je izpostavil zelo podobne izzive, s katerimi se srečujejo vzdrževalci, ko delajo z velikimi podatkovnimi bazami. Te lahko strnem v naslednjih devet točk:

1. Potrebno se je prepričati, da so na voljo vsi potrebni podatki o virih podatkov – podatkovni slovar.

Glede na časovne stiske projektov, razvijalcem vedno zmanjkuje časa, da si zgradijo pravi podatkovni slovarji (angl. *data dictionary*). Po praksi sodim, da vedno raziskujejo pomene različnih podatkov, čeprav bi morali imeti tovrstne podatke na voljo ob pričetku projekta. Če upoštevamo svoje lastne izkušnje, smo do sedaj še vedno naleteli na izziv, ko nam lastniki virov niso znali obrazložiti katere od polj (atributa) v tabeli v viru podatkov. Podatki na virih se dopolnjujejo, dodajajo se nova pravila in nove zakonitosti – vse to je znanje o podatkih celotnega podjetja, ki mora biti širše dostopno znotraj podjetja. Priporoča se, da je podatkovni slovar del repozitorija (hrambe) metapodatkov.

2. Potrebno je zagotoviti, da se shranjuje plane ter čase izvajanja in teste obremenitev v podatkovni bazi.

Vsi časi izvajanja, bodisi za ETL bodisi za začetek in konec vseh operacij, se dandanes lahko izvedejo zelo sistemsko in nadzorovano. Vsak kritičen proces mora biti označen z začetkom in koncem izvajanja. Vsi ti podatki so zelo dragoceni, ko iščemo (ob izrednih dogodkih) vzrok, zakaj se določen prenos podatkov ne izvaja več tako hitro, kot se je do nastanka težav. Bistvo tega napotka je tudi dejstvo, da se, poleg že omenjenih metapodatkov, shranjujejo tudi podatki o procesih, ki potekajo v sistemih podatkovne analitike. To lahko razširimo tudi na merjenje ozkih grl pri prenosih, iskanju največkrat iskanega in potratnega stavka za poizvedbo (SQL), število fizičnih branj, ki jih mora podatkovna baza opraviti ter merjenje pričakovanega števila zadetkov z določenim stavkom za poizvedbo.

3. Potrebno je zagotoviti shranjevanje verzij ETL, validacije in obdelovanje napak v procesih v skupni deljeni podatkovni bazi.

Prav tako kot v prejšnji točki omenjen pristop, je dobra praksa, da se v sistemu podatkovne analitike poskuša zaobjeti (ali ujeti) vse možne napake pri obdelovanju ETL v podatkovnih tabelah. Praksa pravi, da se naj ne bi prebijali skozi ogromne količine zapisov v datotekah ali tabelah za log in to v različnih okoljih. Vse napake se praviloma vpisuje na eno in isto določeno mesto, najbolje v meta podatkovni repozitorij. To pomeni, da se vse napake, ki se pojavijo v domeni procesa orodij ETL, logirajo, potem tudi z napakami, ki se zgodijo v procesu po procesu ETL.

4. Slediti in spremljati je potrebno dolgo trajajoče transakcije.

V transakcijskemu sistemu (OLTP) nas naj ne bi preveč skrbele dolgotrajne transakcije, saj so napisane v dobri veri, da so optimalne in izvajajo točno določeno nalogo. Temu pa v svetu podatkovne analitike in procesov ni tako, ker se pri izvajanju prenosov iz virov v ponore (podatkovno skladišče) izgubijo sledi, kje je npr. prišlo do napake. Programsko kodo bi morali napisati v manjših, še obvladljivih kosih, da lažje ugotovimo, v katerem delu procesa prenosa podatkov je domnevno prišlo do napak.

5. Referenčno integriteto je potrebno uporabljati previdno.

Praksa pri prenosu nas postavi pred odločitev, do katerega nivoja bomo v podatkovno analitiko prenašali referenčno integriteto. To pomeni, da se podatki oziroma tuji ključi v odvisnih tabelah vedno preverijo v nadrejenih tabelah, kjer so taisti ključi praviloma primarni ključi. To je pogoj za delo v transakcijskih bazah (OLTP), saj nam je tako zagotovljena integriteta podatkov, pravilna podatkovna odvisnost in tudi dejstvo, da pri transakcijskih sistemih zaradi programskih kod ne pozabimo po nesreči zapisati vseh ključnih podatkov, kar postavlja pod vprašaj celotno transakcijsko delovanje podatkovne baze in pravilnost podatkov (to se dejansko rešuje kasneje v prenosu in čiščenju podatkov v podatkovno analitiko).

6. Potrebno je definirati in predstaviti presežek vloženih resursov v optimizacijo.

Včasih povprečna odzivnost sistema ni dovolj – izogibati se moramo sicer neskončnim optimizacijam kode podatkovne baze. Možno je, da poizvedba deluje samostojno v podatkovni bazi. Ko pa deluje v kombinaciji z ostalimi poizvedbami ali v procesu ETL počasneje, je potrebno identificirati, kje se pojavi napaka. Se pa zgodijo tudi obratni scenariji, ko sprejmemo zadovoljiv čas za izvajanje poizvedbe.

7. Potrebno je spremljati in razumeti optimizator podatkovne baze ter izvajalnih planov.

Splošno znano je, da so branja in pisanja v delovni pomnilnik oziroma pomnilnik (RAM) »cenejša« glede porabe resursov, kot branja in pisanja na trdi disk oziroma disk. Žal pa ni presenečenje, da je pomanjkanje razumevanja delovanja izvajalnih planov (angl. *query plan*) in števila vhodno/izhodnih operacij pravzaprav zelo pogosto. Vsi razvijalci, ki imajo stik z izdelavo poizvedb (SQL) na velikih podatkovnih bazah, bi morali poznati, kako

izdelovati poizvedbe in dejansko poznati podrobnosti izvajalnih planov ter čim bolj optimizirati izvedbe z dosegom najboljše odzivnosti. Pravi pristop se začne z mislijo, da je potrebno vsako poizvedbo najprej razumeti in sestaviti plan, lahko tudi na listu papirja, kako povezati posamezne tabele med seboj. Nato bi morali, v kolikor imamo dovolj dobro orodje za poizvedbe, preskusiti zagnati poizvedbo brez dejanske pridobitve podatkov iz tabel – torej na način preverjanja, kakšen plan bo določena poizvedba v realnem pridobivanju podatkov ubrala v podatkovni bazi v primeru, ko s svojim znanjem napišemo učinkovito poizvedbo, jo zaženemo v podatkovni bazi in pridobimo želene podatke. Tako prihranimo marsikatero pot do skrbnika podatkovnih baz.

8. Pred izbiro je ključnega pomena raziskati funkcionalnosti in omejitve orodij ETL.

Vsaka izbira arhitekture sistema podatkovne analitike logično vključuje tudi izbiro ustreznega in primernega orodja ETL za izbrano vrsto arhitekture. Vsako orodje ETL ni primerno za posamezno arhitekturo, zato se tovrstne analize izvajajo predhodno, še v fazi izbire arhitekture. Po izbiri primernega orodja ETL je potrebno spoznati orodje in njegove omejitve, saj dandanes obstajajo t.i. demonstracijska (tudi demo) okolja, kjer imamo praviloma s strani ponudnika dotičnega orodja ETL načeloma na voljo dovolj časa, da raziščemo tiste funkcionalnosti, ki jih v končni fazi pričakujemo od podatkovne analitike.

9. Za uspeh projekta je ključno definiranje okolja za testiranje, zagotavljanje kakovosti in migracijo podatkov.

V osnovi sta praksa kontrole in upravljanje s spremembami v svetu podatkovnih baz enako pomembna kot pri ostalih sistemih, npr. v okoljih OLTP. Razvijalcem mora biti dostop do produkcijskih podatkov praviloma omejen – vsi ti podatki morajo biti na voljo v repozitoriju vseh virov in ponorov, možno pa je testiranje in vaja razvijalcev na testnem okolju (v kolikor imamo čim bolj podobne podatke produkcijskim).

Velik izziv torej predstavlja ravno zadnje – kako zagotoviti oziroma ponoviti stanje podatkov na testnem sistemu, ki mora iz ponovljivih razlogov biti čim bolj podobno podatkom na produkcijskem okolju, če želimo predhodno pripraviti in testirati vse kompleksne poizvedbe. V primeru, ko to dosežemo, lahko razvijalci preskušajo realne situacije in zagotavljajo kakovost delovanja sistema – tako v testnem okolju kot kasneje v produkcijskem okolju. Ta naloga pa je vse prej kot lahka, saj je zaradi velike količine podatkov v produkcijskem okolju podatkovne analitike težko zagotoviti realno testno okolje, saj gre resnično lahko za velike količine podatkov (nekaj milijard vrstic). Zato se porajata vprašanje in dilema, kje naj razvijalci pravzaprav testirajo svojo, po njihovo sodeč, optimalno programsko kodo.

Ni redko pri projektih podatkovne analitike, da so vsi protagonisti precej daleč od realnega testiranja in razmišljanja o inicialni migraciji podatkov. Nasprotno, trezno razmišljanje razvijalca gre v smeri, da postavlja vprašanja o podatkih že v zelo zgodnji fazi projekta.

Razvijalci in načrtovalci arhitektur sistemov podatkovne analitike morajo razmišljati predvsem o tem, da smo omenili samo nekaj začetnih napak in dilem, povezanih s projektom v vseh fazah. Naštetih korakov torej ne smemo izpuščati, poznati moramo dovolj dobro podatke v virih in preko procesa ETL obdelane podatke v ponorih – tako imamo pregled tudi na samo konverzijo podatkov.

Vsem naštetim napakam se lahko izognemo že pri izbiri arhitekture sistema podatkovne analitike, ko moramo preučiti vse možne razloge, ki lahko potencialno vodijo v neuspešen zaključek projekta uvedbe podatkovne analitike. Ravno problematika izbire prave arhitekture sistema podatkovne analitike je lahko v veliki meri poenostavljena, če na osnovi nekaterih faktorjev določimo najbolj primerno arhitekturo. Ker smo seznanjeni z najbolj pogostimi težavami, ki jih tipično srečujemo na tovrstnih projektih, se lahko na osnovi teh bolje pripravimo na projekt, ki ga podjetje načrtuje. Pri tem moramo v vsakem primeru podjetje dobro poznati, tako zaposlene kot tudi poslovne procese. Imeti moramo informacijo, koliko je podjetje pripravljeno investirati, kakšna je klima v njem pri uvajanju novih projektnih rešitev itd. Kljub temu, da obstaja več različnih izvedb arhitektur sistema podatkovne analitike, moramo na osnovi več kriterijev že v začetku projekta izbrati možnost, katere realizacijo bomo pripeljali do zaključka projekta uvedbe podatkovne analitike. Odločitev za izvedbo arhitekture podatkovne analitike nam olajša nadgradnje podatkovne analitike ter sledenje modernim tehničnim trendom v informatiki.

2.3 Izbira vrste arhitekture sistema podatkovne analitike

Izbira prave vrste arhitekture se dejansko v polni meri osredotoča na možne dejavnike, ki vplivajo na končno določitev arhitekture. Glede na faze izdelave sistema podatkovne analitike in možnih napak, ki so se zgodovinsko že ponavljale in se jim z modernimi prijemi poskušamo v polni meri izogniti, je potrebno na osnovi teh informacij za posamezni primer upoštevati ustrezne dejavnike. Za odgovor na vprašanje, kako pravilno narediti arhitekturo, je prav toliko literature in praks, kot za odgovor na vprašanje, kako ne izbrati določene arhitekture – ko projekt ni bil dokončan ali pa je neuporaben (Mullins, b.l). Pri izbiri arhitekture se Mullins (b.l.) osredotoča predvsem na naslednje možne dejavnike:

- Kako veliko je podjetje, ki izbira sistem podatkovne analitike? Vprašanje se v tem primeru nanaša predvsem na prepoznavnost in velikost izbranega sistema podatkovne analitike, kakšen poslovno informacijski sistem (v nadaljevanju PIS) podjetje ima in kako informacijsko podkovani so uporabniki.
- Kako hitro mora delovati sistem podatkovne analitike? V diskusiji sta predvsem visoka stopnja delovanja (angl. *High Availability*, v nadaljevanju HA) in odzivni časi (angl. *response times*). V kolikor želimo izbrati zelo kratke odzivne čase, ne moremo razmišljati v smeri najete rešitve v oblaku, temveč na strežnikih podjetja. Kratki odzivni časi se nanašajo na čas obdelave podatkov na strežniku in ne na prenos obdelanih podatkov od strežnika do vmesnika.

- Kakšno informacijsko infrastrukturo že ima podjetje? V primeru, da podjetje že z večino svojih strežnikov gostuje v oblaku, moramo upoštevati, da so tudi viri podatkov iz tega naslova v oblaku. V takih primerih bi sistem podatkovne analitike na lastnih strežnikih podjetja povzročal težave, saj bi bilo potrebno prenašati podatke iz virov (ki so v oblaku) na lokacijo strežnika podjetja, kar pomeni ogromne količine prenosov podatkov in obremenjevanje resursov mreže ter nepotrebnega prenašanja podatkov.
- Kakšen je obseg podatkov in odzivnost poizvedb? V kolikor imamo ogromno podatkovno bazo na več virih, moramo pričakovati tudi ogromno količino podatkov na ponorih – torej v sistemu podatkovne analitike. To je zelo ključen dejavnik pri izbiri ustreznega sistema podatkovne analitike za posamezno podjetje.
- Potrebno je razmišljati v smeri, da bo podatkovna analitika v podjetju sestavni del večjega projekta v prihodnosti. Mnogo podjetij je daljše obdobje (več let) podcenjevalo velikost in obseg podatkovnega skladišča oziroma sistema podatkovne analitike. Pri izbiri moramo zato imeti jasno sliko, kaj bomo s pridobljenimi oziroma urejenimi podatki pravzaprav delali in kako jih lahko umestimo v nadaljnje procese obravnavanja podatkov, npr. zajem vseh podatkov s spletnega mesta kupca ipd. Meje podatkovne analitike lahko hitro prerastejo planiran obseg podatkov, če dodamo samo eno entiteto pri zajemu podatkov preko spletne strani. Želimo povedati, da podatki, ki jih planiramo, ne morejo biti samo surovi podatki, kaj kupec kupuje, ampak bo potrebno spremljati, kaj kupec išče, katero akcija na spletni strani je izbral, koliko časa po odprtju spletne strani se je odločil za ponujen izdelek ipd.
- Kako dobro oziroma priznano je orodje/oziroma tehnologija svetovno znanega ponudnika za določeno arhitekturo podatkovne analitike? Izbiro prave arhitekture moramo določati tudi skozi oceno, kako dobro tehnologijo svetovni ponudniki tovrstne opreme ponujajo. Na trgu je veliko ponudnikov, ki se, ali odločijo za splošen koncept ali pa za ozko usmerjen koncept delovanja – zato moramo koncepte delovanja tehnologije primerjati z lastnimi potrebami obvladovanja.

Perkins (2003) poudarja, da so pri izbiri arhitekture podatkovne analitike pomembni naslednji kritični faktorji uspeha:

- Kdo je sponzor projekta in kakšna je njegova oziroma njena vključenost v projekt? V projekt vpeljave podatkovne analitike morajo biti vključeni vsi člani projektne skupine, posebno vlogo pa ima vodstvo podjetja, saj se mora zavedati, da je ta projekt bistvenega pomena za prihodnje odločanje o strategiji podjetja in načinu njenega uresničevanja. V kolikor podjetje ne pozna in ne analizira svojega poslovanja, so vse odločitve, ki ne temeljijo na tej osnovi, preprosta ugibanja.
- Kakšne so poslovne potrebe? Poslovne potrebe bi morale biti zapisane v strategiji podjetja, prav tako tudi merila, po katerih se lahko meri doseganje sprejete strategije. Iz tega naslova so poslovne potrebe osnova za izbiro arhitekture sistema podatkovne analitike.

- Kakšna je obstoječa infrastruktura informatike? Ena izmed pomembnih nalog podjetja je tudi, da ima strategijo za področje informatike, kar pomeni, da se zaveda pomembnosti informatike in ji določa vire, ki so potrebni za doseganje strateško postavljenih ciljev.
- Kakšna bo arhitektura podatkovne analitike in njeno načrtovanje? Katere podatke bomo črpali iz različnih virov pri polnitvi sistema podatkovne analitike in kako bomo zagotavljali pravilnost podatkov, sta dva bistvena razloga, zakaj moramo biti pazljivi pri izbiri arhitekture. Kakovost metapodatkov ter repozitorija virov je namreč ključnega pomena pri razumevanju podatkov na virih, zato je ne smemo zanemariti in jo vključujemo v vseh korakih.
- Kakšna bo tehnologija podatkovne analitike? Takoj po izbrani arhitekturi podatkovne analitike je potrebno izbrati tehnologijo. Pri izbiri najustreznejše tehnologije – glede na izbrano arhitekturo, moramo upoštevati predvsem uporabniške vmesnike, princip prenosa podatkov, strojno opremo, sistemsko programsko opremo, sistemsko opremo in varnostne mehanizme. Slednje je še posebej pomembno, če ima podjetje skupno varnostno shemo za vse aplikacije in sisteme.
- Kakšna je kakovost podatkov in informacij o virih? Za pregledovanje podatkov in sprejemanje odločitev na osnovi analiziranih podatkih morajo uporabniki dobiti kakovostne podatke. Ti morajo biti točni, pomembni, popolni in jedrnati.
- Kakšno je razvojno okolje? Razvojno okolje mora biti v vsakem primeru vzpostavljeno v taki meri, da vsebuje projektno skupino, metodologijo (vodenja projekta) in orodja.

V predstavitvi problematike smo prikazali, da na odločanje vplivajo tako kvalitativni kot kvantitativni kriteriji. Kateri so pravi in najbolj učinkoviti ter merljivi, mora podjetje oziroma organizacija, ki se odloča za arhitekturo in nenazadnje tudi tehnologijo, odločati samo na podlagi v prejšnjem poglavju podanih faktorjev.

Odločanje o izbiri mora potekati na nivoju vodstva, saj so predhodno omenjene napake velikokrat izvirale iz dejstev, da vodstvo ni pravilno urgiralo ali vsaj pravočasno urgiralo pri ozaveščanju v podjetju, da je uvedba podatkovne analitike nujno potrebna in se ji ni mogoče izogniti, v kolikor podjetje ali organizacija želi obstati na trgu ter nuditi svoje izdelke in storitve.

Velik vpliv na odločitev bi moralo imeti prizadevanje po učinkoviti uvedbi določene arhitekture, ki jo, glede na ponudbo programske opreme, podpremo z najbolj ustrezno tehnologijo.

3 PREGLED IZBRANIH IZVEDENK ARHITEKTUR

Poglavje je namenjeno predstavitvi najbolj pogostih izvedenk arhitektur podatkovne analitike. Pri predstavitvi vseh značilnih arhitektur bomo opisali tudi glavne značilnosti, ki jih delajo drugačne od ostalih arhitektur oziroma pojasnili, kje se razlikujejo od ostalih.

Kandidati so bili izbrani na osnovi raziskovalnega dela magistrskega dela in na podlagi najbolj pogosto omenjenih arhitektur v literaturi (Ariyachandra & Watson, b.l.; Buyer's Guide, b.l.; Knabke & Olbrich, b.l.; Loshin, b.l.; Gupta et al., 2013; Nolting, 2013; Bordei, 2014; Henschen, 2014; Allen, 2015; Gene, 2015; Kernochan, 2015; Richards, 2015; Future of Data Warehouse, Data Mining and Data Visualisation, 2016). Omenjene arhitekture so največkrat izvedene in podprte s strani več svetovno znanih ponudnikov platform (IBM, SAP, Oracle, Teradata, Microsoft in drugi).

Zaradi lažje primerjave in raznolikosti smo izbrali tri arhitekture (možne so tudi druge izpeljanke), ki so predstavljene v nadaljevanju naloge. Pri možnih kriterijih, ki jih bomo preučevali, je potrebno upoštevati tudi omejitve, saj vse izvedenke arhitektur niso primerne za vse vrste projektov oziroma zajemov podatkov v virih.

Prav tako se bomo pri kriterijih posebej opredelili še na skalabilnost, ki je največkrat omenjeni kriterij; pogosto šele takrat, ko je sistem podatkovne analitike že v uporabi in postane premalo zmogljiv zaradi premajhnega dimenzioniranja (ocenitve količine) podatkov pri izbiri izvedenke arhitekture.

Tri izbrane izvedenke arhitekture so:

- prva možnost izvira iz časov pred prehodom v drugo tisočletje, ko so bila tradicionalna podatkovna skladišča že tako poslovno zanimiva, da jih je obravnavala več kot polovica direktorjev ali odločevalcev v oddelku informatike (Sen & Sinha, 2005),
- druga predstavlja tradicionalno podatkovno skladišče z delovanjem v pomnilniku (angl. *in-memory database*) in vsemi izzivi, s katerimi se sooča,
- tretja pa predstavlja novejši in sodobnejši pristop k obravnavanju velike količine podatkov, ki smo jo s to nalogo raziskali, tj. NoSQL.

3.1 Tradicionalna arhitektura sistema podatkovne analitike

Tradicionalno arhitekturo sistema podatkovne analitike v IT delimo na centralizirano podatkovno skladišče, ki se osredotoča na definicijo po konceptu Inmona (2002), in arhitekturo sistema s področnimi skladišči, včasih imenovano distribuirano podatkovno skladišče, ki pa se osredotoča na definicijo po konceptu Kimballa (Kimball & Ross, 2002). Če povzamem, se prva arhitektura osredotoča na poznavanje vseh virov in izdelavo velikega podatkovnega skladišča, katero zajema vse podatke v virih, ki jih na osnovi skupnega imenovalca (ključev, umetnih ključev) poveže v veliko denormalizirano podatkovno bazo.

Druga pa predlaga, da se izvede na način, da se čim prej v razvoju in implementaciji naredijo manjša, vsebinsko določna, področna skladišča, ki se dotikajo enega od poslovnih problemov ali področja, odvisno, kako gledamo na področno skladišče.

Tradicionalna arhitektura podatkovne analitike uporablja praviloma relacijsko podatkovno bazo (Gene, 2015), v našem primeru na enem strežniku (angl. *single-server*). V te podatkovne baze štejemo najbolj in že dolgo razširjene podatkovne baze: Oracle DB, IBM DB2, SQL Server, MySQL in PostgreSQL. Ker bomo predstavili arhitekturo podatkovnih baz, moramo predstaviti tudi osnovne gradnike podatkovnih baz, saj se pojavljajo tako v tradicionalnih podatkovnih bazah kot tudi v arhitekturah pomnilniških podatkovnih baz.

3.1.1 Omejitve tradicionalnih podatkovnih baz

Glavna pomanjkljivost, ki smo jo opazili tekom raziskovalnega dela, je v pomanjkanju horizontalne skalabilnosti. Na podlagi izkušenj, dejstev iz projektov in prispevkov (Nolting, 2013; Allen, 2015) relacijske podatkovne baze niso bile nikoli načrtovane za uporabo vodoravne skalabilnosti. Relacijske podatkovne baze so bile načrtovane za hranjenje manjših količin podatkov in delujoče na enem strežniku za zagotavljanje maksimalne integritete med tabelami uporabnikov. Napredek sistemov za upravljanje relacijskih podatkovnih baz je v nekaterih primerih že tako fleksibilen, da z delnimi rešitvami (npr. strežnik – odjemalec med več strežniki ali porazdelitev celotnega podatkovnega modela na več strežnikov) poskuša izpolnjevati osnovne zahteve vodoravne skalabilnosti. Zagotavljanje vertikalne skalabilnosti relacijske baze navadno ne prinaša težav, dejansko so še cenejše od pristopa zagotavljanja horizontalne skalabilnosti za vsako ceno (Burleson, b.l.; Burleson, 2015).

3.1.2 Primer arhitekture z enim strežnikom

Tradicionalna arhitektura se navadno uporablja na enem strežniku. Podatkovna baza deluje na strežniku, aplikacije, uporabniki ali spletni servisi pa dostopajo do tega strežnika z namenom pridobivanja in manipuliranja s podatki.

Transakcije (Database Concepts, b.l.) v podatkovnih bazah so, kot smo že predstavili, osnovna komunikacijska enota ali osnovna enota dela pri sodelovanju (angl. *interaction*) s podatkovno bazo. Namen vsake transakcije je sodelovanje s podatkovno bazo. Iz naslova te naloge sta pomembni dve operaciji – branje podatkov in zapisovanje (nov zapis, sprememba) podatkov. Vsaka transakcija je, bodisi uspešna bodisi neuspešna oziroma prekinjena. Vmesno stanje zaradi tradicionalnih ACID (Gene, 2015; ACID, 2016) lastnosti podatkovnih baz, ki jih bomo v tem poglavju podrobneje opisali, ne sme obstajati.

Uspešna transakcija je uspešna, ko so uspešne vse posamezne operacije znotraj transakcije. Logično to pomeni, da so se vse spremembe (preko različnih operacij) znotraj transakcije zapisale v podatkovno bazo. V tem primeru pride do sprožitve različnih mehanizmov v

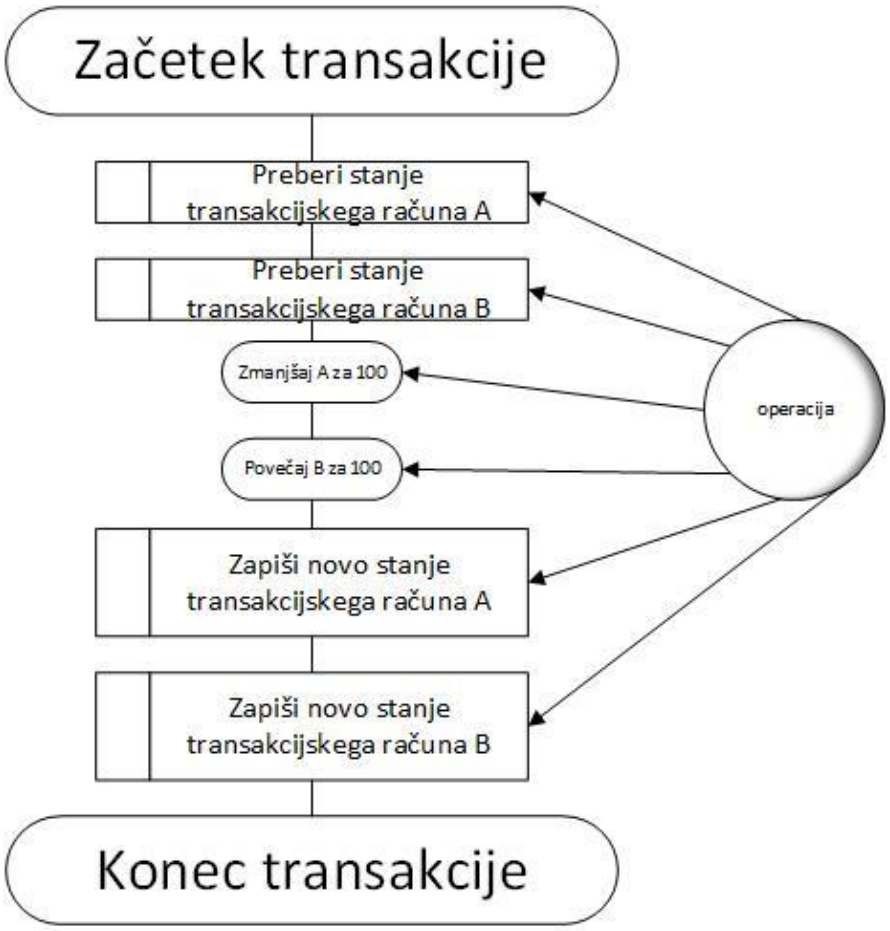
različnih podatkovnih bazah, saj je splošno znano, da transakcija ob uspešnosti izvede potrditev (angl. *commit*).

Na tem mestu je potrebno definirati še dejstvo, da govorimo o tradicionalnih podatkovnih bazah ali tradicionalnih sistemih za podatkovno analitiko, ki so v svetu najbolj razširjeni in jim je še vedno skupno to, da za vsako potrditveno oziroma uspešno transakcijo spremembe zapiše preko različnih procesov – v zelo bližnji prihodnosti tudi na trdi disk. Kako se zapisujejo spremembe v končni fazi na trdi disk, kakšne so zakasnitve, kakšni so pravzaprav boljši mehanizmi od čakanja zapisovanja na trdi disk, so že vprašanja in pomisleki, s katerimi se je zadnja leta ukvarjala množica ponudnikov podatkovne analitike.

Neuspešna transakcija je neuspešna, kadar se prekine, predno se izvedejo vse operacije, definirane znotraj posamezne transakcije. Ob neuspešni transakciji se razveljavijo vse operacije, ki so do trenutka prekinitve že opravile svoje delo. Stanje podatkov v podatkovni bazi, konkretno npr. v tabelah, mora biti tako, kot je bilo pred pričetkom neuspešne transakcije. Neuspešne oziroma prekinjene transakcije se zgodijo zaradi uporabniške razveljavitve, aplikativne napake ali zaradi internih napak v sistemu podatkovne analitike. To v sosledju pomeni, da uporabnik zavestno prekine izvajanje določene transakcije, da aplikacija prepreči spremembo podatkov zaradi pomanjkanja določenih pravic ali pa zmanjka prostora na trdem disku, ipd. Tako kot uspešna transakcija mora tudi neuspešna akcija razveljaviti spremembe in sporočiti internemu sistemu podatkovne baze, da so podatki neuporabni in da se spremembe ne smejo zgoditi oziroma vpisati v podatkovno bazo. Take operacije terjajo določen čas, vendar ne vplivajo na zapis sprememb podatkov v tabele in kasneje fizično še na trdi disk, saj se dejansko ne zapiše ničesar (zapišejo se samo sistemski zapisi - logi, da je prišlo do napake v določenem delu programske kode ali določenem delu aplikacije).

Večina podatkovnih baz pri transakcijah sproti zapisuje operacije v t.i. ponovitveni log (angl. *redo-log*), kar logično pomeni, da sistem podatkovne baze skrbi za pravilnost in logičnost podatkov v vsakem primeru – ali pride do potrditve transakcije ali razveljavitve transakcije (angl. *abort*). Kaj dejansko so operacije in kaj je transakcija, pa je predstavljeno na Sliki 21. Število operacij, ki se dejansko izvede med začetkom transakcije in koncem transakcije ni vnaprej določeno – lahko so kratke transakcije, možne pa so tudi transakcije, ki lahko vključujejo dodatne klice novih transakcij. Pomembno je le to, da se jih obravnava kot celoto in se je ne sme deliti.

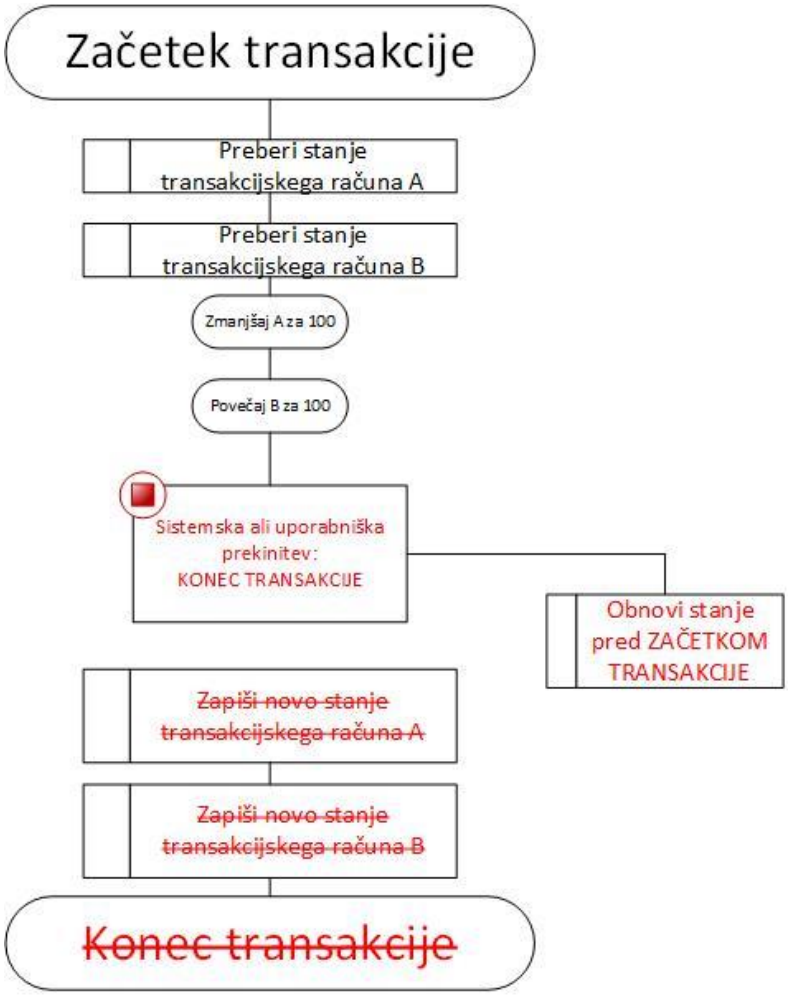
Slika 21: Primer uspešne transakcije in operacij



Ilustrativen primer na Sliki 21 je sestavljen iz preprostih šestih operacij. Navedeni so kratki opisi operacij, ki se izvedejo zaporedoma. V primeru, da ne pride do prekinitve, se s transakcijskega računa A in B prebere stanje, poveča oziroma zmanjša se A oziroma B, potem se zapišejo nova stanja transakcijskih računov A in B v podatkovno bazo (v tabelo). V tem primeru se ni zgodila ne uporabniška napaka ne katerakoli druga (nenapovedana) prekinitev.

Na Sliki 22 pa prikazujemo stanje transakcije z operacijami, kjer je prišlo do uporabniške oziroma sistemske (nenapovedane) napake. V tem primeru se stanji transakcijskih računov A in B v podatkovni bazi (npr. v tabelah stanj) ne smeta spremeniti, saj se je transakcija prekinila. Vsi mehanizmi različnih sistemov upravljanja podatkovnih baz morajo s svojimi mehanizmi poskrbeti, da ostali uporabniki ne vidijo sprememb, dokler ni transakcija ali potrjena ali prekinjena.

Slika 22: Primer neuspešne transakcije in operacij



Iz obeh primerov, uspešne in neuspešne transakcije, lahko definiramo štiri osnovne lastnosti katerekoli podatkovne baze, ki dela s transakcijami. Da se vse operacije izvedejo v celoti oziroma da se ne izvedejo vse (torej vse ali nobena), imenujemo **atomarnost**. Vzdrževanje in zagotavljanje atomarnosti je ena izmed glavnih lastnosti podatkovne baze, saj mora podatkovna baza skrbeti za celovitost operacij in transakcij. Ko se izvede uspešna transakcija, se že, kot definirano zgoraj, zapišejo podatki logično v dotične tabele in fizično na trdi disk, da se ob pomanjkanju npr. napetosti ne izgubijo – tej lastnosti rečemo **trajnost**. Poleg teh dveh osnovnih lastnosti sta še **konsistentnost** in **izolacija**.

Osnove lastnosti podatkovne baze **ACID** (Gene, 2015; ACID, 2016) bomo zapisali kot štiri posebej obrazložene lastnosti.

Atomarnost (angl. ***A**tomicity*) je lastnost podatkovne baze, ko se v primeru uspešne transakcije izvedejo vse operacije v dotični transakciji, ali ko se v primeru neuspešne transakcije (zaradi uporabniške, aplikativne ali sistemske prekinitve) ne spremeni noben uporabniški podatek v podatkovni bazi. Ta lastnost je izrednega pomeni pri razvijalcih, saj

jim ni potrebno skrbeti, da bi jih uspešna ali neuspešna transakcija pustila v dvomih, katere operacije so ali niso bile uspešno zapisane. V primeru, ko se transakcija zaradi kateregakoli razloga ne more uspešno zaključiti, sistem upravljanja podatkovne baze poskrbi, da se neuspele operacije znotraj transakcije ob prekinitvi razveljavijo – podatki ohranijo prvotne vrednosti. Ob tem se povrne tudi celotno stanje podatkov, kot je bilo pred začetkom transakcije. Razvijalcem ni potrebno skrbeti in urejati stanja, kot je bilo pred začetkom transakcije. Tak način zelo poenostavlja razvoj in delo razvijalcev, ki delajo s klasično oziroma tradicionalno podatkovno bazo.

Konsistentnost (angl. ***C****onsistency*) je lastnost, s katero podatkovna baza zagotavlja, da so podatki v množici uspešnih in neuspešnih transakcij konsistentni. Konsistentnost širše pomeni, da podatkovna baza prehaja iz enega konsistentnega stanja v drugo konsistentno stanje, kar enostavneje pomeni, da je podatkovna baza vedno prehajala iz enega v drugo stanje na način, ki je ponovljiv, torej se stanje podatkovne baze ne spreminja naključno, ampak nadzorovano in je podvrženo določenim pravilom.

Izolacija (angl. ***I****solation*) je lastnost, s katero podatkovna baza zagotavlja, da se v trenutku izvajanja določene transakcije ta ne ozira na učinke ostalih transakcij, ki se prav tako izvajajo v istem trenutku. Vsaka transakcija mora praviloma obravnavati tako stanje podatkov v tabelah, kot če bi bila edina transakcija v sistemu. Sistem upravljanja podatkovne baze mora zato zagotavljati, da se sočasne transakcije konec koncev obnašajo na način, kot če bi se izvajale zaporedno. Vedno tako ne gre. V določenih primerih (ki niso redki) transakcije namreč čakajo med seboj, saj se popravlja natanko isti podatek, zato mora druga transakcija v izvajanju počakati prvo transakcijo, bodisi da se zaključi bodisi da se razveljavi. Če se zaključi, pride navadno do ponovnega branja začetnega stanja pri drugi transakciji, saj se je začetno stanje za drugo transakcijo vmes že spremenilo, ker ga je prva transakcija v izvajanju spremenila. Takim situacijam se v razvojnih okoljih načrtno izogibamo , saj se preveliko čakanje transakcij med seboj rezultira v slabem podatkovnem modelu, lahko pa tudi v slabi razvijalski praksi.

Trajnost (angl. ***D****urability*) je lastnost ki zagotavlja, da podatki, ki jih transakcija spreminja, niso nikoli izgubljeni – seveda samo v primeru uspešne transakcije (po akciji potrdi – *commit*). Trajnost ima še drugo dobro lastnost, in sicer zagotavlja spremembo podatkov uspešne transakcije kljub temu, da se sama podatkovna baza nepričakovano zaustavi oziroma preneha delovati (angl. *database system failure/abort*). Za zagotovitev te lastnosti potrebujemo določen proces v podatkovni bazi, ki neprestano skrbi za zapisovanje operacij, katere so bile izvedene v uspešnih transakcijah (angl. *redo-log*). Zato je logiranje zaporedno in obvezno, ker je potrebno zagotoviti zaporedje vseh operacij, ki so se zgodile. V primeru nenadne zaustavitve podatkovne baze (angl. *database system failure/abort*) se mora vzpostaviti ponovno v stanje, ki je bilo pred zaustavitvijo. Tak log mora biti obvezno zapisan oziroma mora obstajati na medijih, kjer se podatki po izklopu električne energije ne izgubijo. Najpogosteje so to klasični trdi diski ali pomnilniški disk (SSD). Zelo pogost

je princip, kjer se v log najprej zapiše operacija, šele nato (to je praktično istočasno) pa odraža operacija spremembo podatka v podatkovni bazi. Prav tako se vnaprejšnje zapisovanje v log opravi v enakem vrstnem redu, kot se v podatkovno bazo. Iz naslova zagotavljanja pravilnega stanja podatkovne baze po ponovnem zagonu ob njeni nenadni zaustavitvi se operacije restavrirajo (obnovijo) iz loga ravno zato, ker so se v ta log zapisovale pred zapisovanjem podatka v podatkovno bazo. To pomeni, da v primeru, ko se je podatkovna baza nenadno zaustavila, še predno je bila potrjena transakcija zapisa v podatkovno bazo (angl. *commit*), se podatki obnovijo iz loga, ker so se v log zapisali v vsakem primeru. To je zelo preprost princip, ki je preverjen in deluje v večini podatkovni baz.

3.1.3 Skalabilnost

Relacijske podatkovne baze zaradi svoje narave nikoli niso bile dizajnirane, da bodo vodoravno in vertikalno skalabilne, vsaj ne v taki smeri, kot so podatkovne baze novejših tehnologij in pristopov. Svojo največjo mero skalabilnosti izkazuje v veliki večini v vertikalni skalabilnosti, saj s tem le dodajamo resurse obstoječemu strežniku (npr. RAM, trdi disk, SSD disk, procesorje). Ker relacijske podatkovne baze primarno zagotavljajo lastnosti ACID, so prvotno delovale na enojnih strežnikih in zagotavljale integriteto med podatki med tabelami. To so poglavitni razlogi, da ni bilo potrebno podatke podvojevati na več strežnikih. Nekaj poskusov je sicer bilo po principu strežnik – odjemalec (angl. *master – slave*), kjer je bil strežnik na voljo za glavni del obdelovanja podatkov, odjemalec pa je bil sinhroniziran in je prevzemal prekomerne obremenitve svojega glavnega strežnika. Poskusi vodoravne skalabilnosti relacijske baze so bili opravljeni tudi s tako imenovano razdrobljeno strukturo podatkov na več različnih podatkovnih bazah na različnih strežnikih, vendar so take rešitve kompleksne in težje izvedljive.

Glavne značilnosti in dejstva, ki so pomembna za to arhitekturo (Burleson, b.l.; Lavezzo, b.l.; Loshin, b.l.; Allen, 2015; Gene, 2015; Richards, 2015; Burleson, 2016; Can MySQL scale horizontally?, 2016):

- obstaja in uporablja se že več kot dvajset let ter je med najbolj razširjenimi,
- napisano je veliko literature ter člankov, obstaja pa tudi veliko znanja (angl. *know how*),
- programska oprema se je v teh letih toliko pozicionirala na trgu, da obstajajo že tako imenovane paketne rešitve (angl. *Off-the-shelf* ali *Out of the box*) za tovrstno arhitekturo,
- zagotavlja konsistentnost podatkov,
- zagotavlja izvajanje transakcij v celoti (atomarnost transakcije – izvedejo se vse operacije v transakciji),
- izolacija transakcije (izvajanje transakcije ne vpliva na ostale transakcije),
- trajnost transakcije (ko je transakcija potrjena, se spremenjeni podatki ne izgubijo) ter

- manjša področna skladišča, ki so namenjena točno določenih podatkom, oddelkom ali problemom; lahko jih je več, ki se gradijo sproti v skladu s potrebami po analizah.

3.2 Arhitektura sistema podatkovne analitike z uporabo pomnilniške podatkovne baze

Arhitektura sistema podatkovne analitike z uporabo pomnilniške podatkovne baze (angl. *in-memory database*) je dejansko med novejšimi tehnologijami. Lahko zapišemo, da trenutno še osvaja svet informatike (Oracle Database In-Memory, 2016). V poglavju 1.3.4 smo že predstavili podatkovno analitiko v pomnilniku ter opisali, kako dejansko deluje ta koncept, vendar moramo zapisati še težave oziroma omejitve, s katerimi se spopadajo pomnilniške baze. V nadaljevanju bom angleški termin »in memory database« poimenoval pomnilniška podatkovna baza.

Pomnilniška podatkovna baza je postala novejša različica tradicionalne podatkovne baze (ki je prvotno hranila vse svoje podatke na trdih diskih) in s katero se je v informatiki začelo ukvarjati čedalje več ponudnikov. Pri tem moramo paziti predvsem na njeno pravilno poimenovanje, saj obstajata vsaj dva termina, ki ju je potrebno poznati in ločevati ter uporabljati v pravem kontekstu. Prvi termin je pomnilniška podatkovna baza (angl. *in-memory*), drugi termin pa je pomnilniška analitika podatkov. Pomnilniška podatkovna baza tako primarno uporablja glavni pomnilnik (RAM) kot hrambo vseh potrebnih operativnih podatkov za delovanje, kar je v nasprotju s konceptom tradicionalne podatkovne baze, ki hrani podatke za operativno delovanje na disku (čeprav v resnici uporablja pomnilnik kot medpomnilnik za izvajanje operacij). Torej pomnilniška podatkovna baza zapisuje in bere podatke samo v pomnilniku, kar drastično poveča odzivnost delovanja transakcij. Ko pa opazujemo pomnilniško analitiko (angl. *in-memory analytics*), pa je glavna lastnost ta, da so podatki naloženi v pomnilnik z namenom, da se bodo izvajale analitične operacije v pomnilniku in ne na disku ter se tako hitreje izvedle.

3.2.1 Omejitve pomnilniških podatkovnih baz

Pomnilniškim podatkovnim bazam je skupno, da morajo obvladovati zapisanje na trdi disk, kadar zmanjka pomnilniškega prostora ali je zaradi algoritma bolje v nekem trenutku zapisati stanje pomnilnika tako, kot je, na trdi disk. Skupno jim je tudi to, da se morajo ukvarjati na optimalen način, kako pravilno izrabljati pomnilnik, da pomnilniška podatkovna baza ne zaide v svojem delovanju v težave pri zasedanju in sproščanju pomnilnika, ter tudi to, kako bo podatkovna baza obvladovala izvajanje ukazov, bodisi SQL bodisi kakšnih drugih, ki so pomembni za samo delovanje arhitekture. Več o teh težavah bom zapisal v nadaljevanju.

Že v prejšnjih poglavjih smo opisovali težave, ki jih imajo pomnilniške podatkovne baze (še vedno), med drugimi tudi s pomnilnikom. Ne glede na velikost delovnega pomnilnika

(RAM) se v nobenem primeru podatki podatkovne baze ne »preselijo« v delovni pomnilnik, ampak se prenašajo po algoritmih na način, da so – ali najbolj pogosti ali pa najbolj novi v pomnilniku, še vedno pa ne govorimo o popolnem prenosu cele podatkovne baze v delovni pomnilnik. Če ne zaradi drugega, že zaradi tega ne, ker mora biti del podatkovne baze vedno na trdem disku zaradi zagotavljanja optimalnega delovanja in povrnitve po incidentih.

Poleg omejitev delovanja celotne podatkovne baze v pomnilniku se moramo omejiti tudi na dejstva, da v primeru, ko želimo imeti podatkovno bazo v pomnilniku, nastopi težava v primeru vodoravne skalabilnosti – sinhronizacije (med strežniki), ko želimo uskladiti različice podatkovnih baz v pomnilnikih več strežnikov hkrati. To pomeni, da moramo kopije npr. glavne (angl. *master*) podatkovne baze v pomnilniku neprestano osveževati, kar ima za posledico odvisnost ostalih podrejenih (angl. *slave*) pomnilniških podatkovnih baz. Alternativni algoritem pa omogoča hkratno sinhrono osveževanje vseh sprememb na vseh pomniških podatkovnih bazah hkrati. Posledica je veliko podatkovnega prometa med strežniki, za kar potrebujemo zelo široko internetno povezavo med strežniki, ki je odvisna od trenutnega prometa na internetu.

3.2.2 Primeri sistemov za pomnilniške podatkovne baze

Kot primer bom predstavil nekaj podatkovnih baz, ki delujejo v pomnilniku (Zhang et al., 2015; NoSQL, 2016): relacijske in NoSQL podatkovne baze (angl. *Not only Structured Query Language*, v nadaljevanju NoSQL). Relacijske podatkovne baze, bodisi OLTP ali OLAP (v bistvu ROLAP), imajo že same po sebi uravnavanje obremenjenosti, kar je logično, saj so na trgu že več desetletij. To pa za NoSQL podatkovne baze ne moremo trditi – na tem področju je še vedno velik izziv operacij podatkovne analitike (v NoSQL).

Večje in najbolj razširjene ter znane relacijske pomnilniške podatkovne baze so po mojem mnenju še vedno Oracle TimesTen, IBM SolidDB in SAP HANA. Poleg teh treh poglavitnih je še H-Store, ki je optimizirana predvsem za transakcije OLTP (Kallman et al., 2008; H-Store, 2016) in je bila nekaj časa znana pod komercialnim imenom VoltDB (VoltDB, 2016), ter Hekaton, ki je integrirana v Microsoft SQL Server.

Podatkovna baza Oracle TimesTen (TimesTen, 2016) je zasnovana za delovanje kot samostojna podatkovna baza ali pa aplikacijski pomnilniški vmesnik (angl. *Application-Tier Database Cache*) z lastnostima podatkovne baze. TimesTen je relacijska podatkovna baza, ki je zasnovana za delo znotraj aplikacijskega prostora in shranjuje vse svoje podatke v glavnem pomnilniku, saj uporablja optimizirano delovanje v pomnilniku za sisteme z velikimi obremenitvami. Aplikacije uporabljajo za dostop do TimesTen standardni vmesnik SQL, OCI, Pro*C in PL/SQL. Za uporabnike, kateri že imajo aplikacije, ki za svoje podatke uporabljajo Oraclovo podatkovno bazo, se TimesTen uporablja kot

pomnilniški predpomnilnik (angl. *cache*) z avtomatsko sinhronizacijo med TimesTen in podatkovno bazo Oracle.

Trajnost in povrnitev stanja po prekinitvi sistema je dosežena s kombinacijo loga transakcij in zapisovanja podatkov na disk. Zaradi lastnosti podvojenosti je TimesTen namenjen in konfiguriran za visoko razpoložljive sistema z možnostjo takojšnjega sekundarnega zagona rezervnega delovanja (angl. *instant failover*).

Tipično je TimesTen implementiran kot aplikacijska knjižnica, iz katere se povezujemo na aplikacije in zato ne potrebuje posegov administratorja. To nam pomaga odpraviti nivo in delo na komunikaciji strežnik – odjemalec, ko se prijavljamo na podatkovno bazo. Obstaja seveda možnost, da se uporabi standardni način strežnik – odjemalec, ko podatkovno bazo uporablja večje število aplikacij.

UNICOM solidDB oziroma SolidDB je bila prvenstveno pomnilniška podatkovna baza podjetja IBM. 30. junija 2014 pa je to podatkovno bazo kupilo podjetje UNICOM System Inc (UNICOM SolidDB, 2016). Ta baza je relacijska in pomnilniška z visoko odzivnostjo ter zanesljivostjo za aplikacije, ki potrebujejo podatkovno bazo v realnem času, tj. hipno. SolidDB podatkovno bazo sestavljata dva bazna jedra: eden je tradicionalen na osnovi delovanja na disku, drugi pa deluje v glavnem pomnilniku in omogoča, da se tabele trajno zapisujejo samo v pomnilnik.

Omenili smo že, da so pri pomnilniških podatkovnih bazah pomembni tudi indeksi, ki se izdelajo za potrebe iskanja podatkov v tabelah. Pri solidDB so indeksi shranjeni v celoti v glavnem pomnilniku. Pri uporabi pomnilnika lahko za vsako tabelo izberemo, kje bo tabela shranjena – bodisi na disku bodisi v glavnem pomnilniku. V primerih, ko imamo postavljeno instalacijo z dvema strežnikoma, drugemu strežniku rečemo vroča kopija podatkov (angl. *hot-standby).* SolidDB ves čas vzdržuje dve identični sinhronizirani kopiji podatkov. Ko pride do nepričakovanega izpada enega izmed strežnikov, lahko aplikacije obnovijo dostop do solidDB v manj kot sekundi brez kakršnekoli izgube podatkov. Podatki se zapisujejo na disk preko potrditvenih točk (angl. *checkpoints*) in loga transakcij. Ponuja nam močno podporo pri mapiranjih in pretvorbah podatkov ter je tako vodoravno kot navpično skalabilna.

Horizontalna skalabilnost z dvema strežnikoma (zapisana v prejšnjem odstavku) omogoča, da oba strežnika delita dostop do glavnega pomnilnika z namenom povečati performance in zmanjšati odzivne čase. SolidDB s svojimi metodami dostopa izvaja učinkovit mehanizem kontrole sočasnega dostopa ter tako lahko zagotavlja integriteto transakcij v množici sočasno izvajanih operacij. Prav tako solidDB popolnoma izrablja 64-bitni glavni pomnilnik in s pridom izrabljanja arhitekturo več-jedrnega in več-procesorskega procesiranja.

Ta podatkovna baza se zelo enostavno namesti in administrira, saj lahko teče implementirana neposredno v aplikaciji. Povezovanje aplikacije in solidDB neposredno s kodo pomeni, da programski zahtevki na nivoju aplikacije lahko naslavljajo isti pomnilniški prostor. To logično pomeni, da aplikacija neposredno dostopa do istega pomnilniškega prostora kot pomnilniška podatkovna baza, zaradi česar odpade nepotrebno usklajevanje med deli pomnilnika za aplikacije in delom pomnilnika za podatkovno bazo. Ostale aplikacije lahko dostopajo do solidDB preko standardnih vmesnikov: ODBC (ODBC, b.l.), JDBC (JDBC Basics, 2016) in vmesniki SQL, prav tako pa solidDB podpira standard SQL-92 z že nekaj funkcionalnostmi SQL-98 in SQL 2003. Ob vsem tem pa polno zagotavlja vse lastnosti ACID. Zanimivo je tudi dejstvo, da se lahko solidDB uporablja kot predpomnilniška programska oprema za pospešeno delovanje podatkovnih baz IBM DB2 ter Informix, Oracle, Microsoft SQL Server in Sysbase.

SAP HANA (What is SAP HANA, 2016) je pomnilniška podatkovna baza, ki združuje transakcijsko obdelovanje podatkov, analitično obdelovanje podatkov in funkcionalnost procesiranja aplikativne logike v pomnilniku. Arhitekturno je SAP HANA sistem za upravljanje podatkovne baze s standardnim vmesnikom SQL, visoko razpoložljivostjo in dvema lastnostima ACID – izolacija transakcije in obnovitev podatkov (trajnost oziroma angl. *durability*). Podpira večino zahtev standarda SQL-92, tiste aplikacije SAP, ki uporabljajo t.i. Open SQL, lahko SAP HANO uporabljajo brez kakršnihkoli sprememb. Dodatno funkcionalnost, tj. prosto iskanje po podatkih, pa so implementirali z uporabo razširjenih vmesnikov SQL. Aplikacije za poslovno inteligenco so podprte skozi Microsoftov jezik MDX, kar omogoča uporabo tudi v programu Excel.

Principal SAP je izdal prav tako usmeritve, kako izvajati določene funkcionalnosti neposredno v pomnilniku, kot sta npr. pretvorba denarnih valut in različni poslovni koledarji. Ena izmed velikih posebnosti je tudi ta, da SAP HANA podpira tako vrstično kot stolpčno shranjevanje podatkov z enakim programskim jedrom hkrati. To v praksi pomeni, da lahko uporabi shranjevanje na način vrstičnega shranjevanja ali pa stolpčnega shranjevanja, tako da je prvi primeren za transakcijske aplikacije, drugi pa je boljši za poročila in analitiko. Dejstvo je, da je stolpčno stiskanje podatkov bolj učinkovito od vrstičnega. Z zelo učinkovitimi vzporednimi večjedrnimi procesorji SAP HANA optimizira paralelno izvajanje SQL-ov, kar se lahko skalabilno povečuje s številom jeder, izvaja agregacijske operacije s povečevanjem števila niti (angl. *threads*) ki se izvajajo vzporedno (paralelno). HANA prav tako podpira distribucijo po več različnih računalnikih (gostiteljih), kjer se lahko velike tabele (particije) procesirajo vzporedno.

Kar dodatno dela SAP HANO posebno, je dejstvo, da je v literaturi in praksi zadnje leto zelo priljubljena. Na prvi pogled se niti ne razlikuje od preostalih (Oraclove TimesTen, UNICOM-ove solidDB), enako shranjevanje podatkov v delovnem pomnilniku niti ni tako novo. Glede na prispevek (What is SAP HANA, 2016) lahko pridemo do naslednjih zaključkov, ki opredeljujejo SAP HANO tako posebno:

- Kombinacije shranjevanja podatkov – vrstično in stolpčno; pri HANI je to poglavitna razlika od ostalih podatkovnih sistemov, saj omogoča obe funkcionalnosti shranjevanja hkrati.
- SAP HANA je že privzeto optimizirana tako, da vse deluje v realnem času, kar pomeni, da morajo biti podatki na voljo ob izvajanju statistik. Čeprav je delovni pomnilnik (RAM) v vsakem primeru hitrejši od diska, procesor (CPU) vseeno troši delovne cikle (na osnovi svojega delovnega takta oziroma frekvence delovanja) za čakanje podatkov iz delovnega pomnilnika. HANA zmanjšuje te neizkoriščene procesorske cikle (takte) na način, da je v procesorju oziroma njegovem predpomnilniku (angl. *cache*) toliko in dovolj podatkov, da jih bo kmalu tudi uporabil. Prav tako uporablja zakasnitve pri stiskanju ali razširjanju podatkov z namenom, da čim kasneje izvede to operacijo (ko je procesor prost), ali pa izvaja operacije neposredno na stisnjenih podatkih.
- Obstoj rešitve HANE, ki jo prodaja kot samostojno rešitev (angl. *appliance*), vendar samo na mikroprocesorju tipa Intel Xeon, kar zoža nabor možnih platform. Razvijalci Intela so iz tega naslova več mesecev, skupaj s SAP ekipo, programirali jedro HANA za samostojne rešitve, poglavitno z namenom, da HANA uporabi vse možne funkcionalnosti Intel Xeon arhitekture čipa. SAP HANA t.i. »High Performance Analytic Appliance« lahko izvede analize v arhitekturi vodoravne skalabilnosti na več kot petsto milijard zapisih v manj kot eni minuti. To pomeni, da ima HANA celotno podatkovno skladišče v glavnem pomnilniku in vrača rezultate v realnem času.

Uvodoma smo že predstavili obe imeni, tako H-Store (H-Store, 2016) kot tudi VoltDB (VoltDB, 2016). Zanimivost pri tej kombinaciji imen je predvsem v življenjskem ciklu prototipa H-Store. Slednji je eksperimentalna pomnilniška podatkovna baza, ki je optimizirana za sisteme OLTP. Njena poglavitna lastnost je visoka distributivnost (porazdeljenost), uporablja pa vrstično shranjevanje podatkov (za razliko od pomnilniško najbolj optimalnega stolpčnega shranjevanja podatkov), ki deluje na principu nedeljenosti podatkov. VoltDB pa tako ponuja dodatno masovno paralelno obdelovanje podatkov z vodoravno skalabilnostjo, zagotavlja lastnosti ACID, pripravljenost za porazdelitev podatkov na več geografsko ločenih lokacijah, hiter uvoz podatkov ter močno podporo razvojnim rutinam in podporo SQL ter programskem jeziku Java.

Microsoft je predstavil članek (Get started with In-Memory (Preview) in SQL Database, 2016), kjer Hekaton opisuje kot polno integriranega s strežnikom SQL. Predstavljen je kot pomnilniška podatkovna baza za potrebe velikih obremenitev v okoljih OLTP. Glede na preprosto uporabo, kjer uporabnik prosto določi, katera tabela naj bo v pomnilniku, katera pa je lahko na trdem disku, so tabele v pomnilniku trajne in dostopne preko programskega vmesnika T-SQL na enak način, kot da bi dostopali do navadnih tabel v strežniku SQL. Jedro delovanja Hekatona je optimizirano za visoko sočasnost delovanja transakcij, uporablja pa tudi nove optimistične metode in metode zagotavljanja visokega nivoja

sočasnosti z več različicami podatkov v pomnilniku. Posebnost te pomnilniške podatkovne baze je v tem, da prevede stavke in rutine v strojno kodo, ki jo potem učinkovito izvaja.

Na trgu so prisotne tako komercialne (licenčna politika) kot tudi odprto-kodne pomnilniške podatkovne baze (brezplačne). Že leta 2013 so obstajale večje (Gupta et al., 2013) pomnilniške podatkovne baze, ki jih razdelimo glede na:

- že omenjene komercialne TimesTen od Oracle, SolidDB od IBM, HANA od SAP še na naslednje:
  - extremeDB,
  - SQLFire.
- odprto-kodne:
  - SQLite,
  - CSQL,
  - MonetDB.

Konkretne primere uporabe pomnilniške baze pa lahko povzamemo po strokovnem prispevku Henschena (Henschen, 2014), ki je predstavil velike in že uveljavljene ponudnike, kako bodo v svoje poslovno informacijske sisteme vključili podatkovne baze v pomnilniku ter na kakšen način – predvsem v stroškovnem ali arhitekturnem smislu. SAP s svojo HANO obljublja pravzaprav prehod na pomnilniško bazo brez pravih sprememb v podatkovni arhitekturi klasične podatkovne baze, v kar odjemalci (stranke) precej dvomijo. Po drugi strani so ostali velikani: Oracle, Microsoft in IBM bolj previdni z izjavami o brezšivnem prehodu na podatkovno bazo v pomnilniku in omenjajo vsaj minimalno migracijo na novo različico (npr. Microsoft SQL Server 2014). SAP s svojo HANO obljublja neopazno migracijo, Oracle, IBM in Microsoft pa predlagajo migracijo na novejšo podatkovno bazo z opcijo »v pomnilniku« (angl. *in-memory*) ter delitev produktov na uporabo transakcijskega in analitičnega dela. Večje stranke prav tako previdno migrirajo svoja poslovanja iz obstoječih podatkovnih baz v podatkovne baze v pomnilniku, saj ne želijo ogroziti svojega trdnega poslovanja. V prispevku so predstavljena tudi večja ameriška podjetja, ki za svoje nove in popularne asortimaje prodajnih izdelkov niso čakala na nočne dolge obdelave obstoječih podatkov, ampak so še isti dan – po zaključku rednega dela, v roku nekaj minut že pridobila analizirane podatke ter se takoj pripravila za naslednji poslovni dan. Zanimiv je primer, ko se je podjetje, ki se ukvarja s stavami, lahko razširilo na nove trge tudi zato, ker so poslovanje oziroma sklepanje športnih stav povečali z 12.000 v sekundi na 150.000 v sekundi. Glede na vse primere v prispevku lahko povzamemo, da so podatkovne baze v pomnilniku podjetjem odprle neslutene možnosti za izredno hiter (znotraj nekaj ur) sprejem odločitev, kaj lahko ponudijo v svojih spletnih trgovinah, kaj lahko ponudijo v določeni regiji, ker prihaja nevihtno obdobje, in kako namesto v dvajsetih minutah splanirati dostavo proizvedenih produktov v samo treh sekundah.

3.2.3 Skalabilnost

V literaturi in virih posebnih ugotovitev o skalabilnosti nisem zasledil, je pa skupno virom predvsem to, da se pomnilniška podatkovna baza širi zlasti v smeri vertikalne skalabilnosti, kar pomeni, da se obstoječi infrastrukturi strežnika dodaja več glavnega pomnilnika, npr. iz 128 GB na 512 GB. To naj bi pripomoglo, da lahko v glavni pomnilnik zapišemo več podatkov (tabel), kot bi jih zapisali v manjši glavni pomnilnik. Stično področje opisanim pomnilniškim podatkovnim bazam je omenjeno predvsem na področju vodoravne skalabilnosti, ko se kopije pomnilniških podatkovnih baz razprostrejo na več lokacij, vendar gre le za kopije enih in istih podatkov.

3.3 Arhitektura sistema podatkovne analitike z distribuirano podatkovno bazo NoSQL

Kljub temu, da so Li, Qian in Ye (2011) predlagali novo arhitekturo podatkovne analitike, se še niso dotaknili (ampak samo omenili) vse večjih potreb po porazdeljenosti podatkov po celem spletu/svetu oziroma potreb za časovno najbolj optimalno izvajanje standardnih poizvedb ali kompleksnih analiz. Prav tako Gene (2015) opisuje NoSQL kot tehniko, s katero se povečajo performanse in kapacitete računalniškega sistema.

NoSQL podatkovne baze v pomnilniku uporabljajo drugačne algoritme hranitve in pridobivanja podatkov kot relacijske podatkovne baze v pomnilniku. Podatki (v NoSQL) so strukturirani kot drevesne strukture, grafi ali ključ-vrednost in ne kot tabele, poizvedovalni jezik pa prav tako ni standardni SQL. Glavna prednost je enostavnost, vodoravna skalabilnost ter podrobnejša kontrola razpoložljivosti in nadzora.

Splošno so znane večje relacijske podatkovne baze, kot so: Oracle, IBM DB2, MS SQL, PostgreSQL in MySQL. Med novejše relacijske podatkovne baze pa sodijo tako imenovane NewSQL – Google Spanner (Corbett et al., 2012) in H-Store (Kallman et al., 2008). Ti dve novejši podatkovni bazi želita zagotoviti enako skalabilnost, kot jo želijo NoSQL za OLTP sisteme z zagotavljanjem lastnosti ACID za relacijske podatkovne baze. Google Spanner uporablja Google za svoje oglaševanje (angl. *Google Ads*) ter je nadomestila že očitno premalo zmogljivo in skalabilno MySQL podatkovno bazo.

3.3.1 Omejitve NoSQL podatkovnih baz

Največja in najbolj očitna pomanjkljivost je predvsem nezmožnost zagotavljanja lastnosti ACID (Lavezzo, b.l.). Ena izmed pomanjkljivosti je tudi ne-univerzalnost v vlogi podatkovne baze, saj je primerna za posebne primere in iz tega razloga nima splošnega namena. Kernochan (2015) je predstavil podobne omejitve, ki jih je že kategoriziral:

- pogosta podatkovna nekonsistentnost,

- ne vsebuje globoke uporabe analitične podpore in
- pomanjkljivost zagotavljanja visoke razpoložljivosti v oblačnem delovanju.

Podobno meni tudi Richards (2015), ki je predstavil realen in kritičen pogled na trenutno še zelo obetajočo tehnologijo NoSQL, saj je opisal dejstva, ki jih dejansko pri izbiri najustreznejše arhitekture preprosto moramo poznati. Med najpomembnejšimi so izpostavljena naslednja:

- manj zrela in pripravljena za trg – Relacijske podatkovne baze obstajajo dlje časa, na trgu so že več kot 25 let. Kljub nekaterim pomislekom, da je to že starost, ki bi lahko označila arhitekturo kot zastarelo, je v tem obdobju relacijska podatkovna baza pridobila veliko funkcionalnosti, ki so danes nujno potrebne. NoSQL podatkovne baze pa v realnosti šele pridobivajo obljubljene funkcionalnosti, ki so bile za to tehnologijo predstavljene.
- manj podpore – Glede na dolgoletni obstoj relacijskih podatkovnih baz je področje NoSQL še tako nepodprto s strani velikih ponudnikov, da podjetja z uporabo NoSQL ne dobijo vedno prave in takojšnje podpore, kot so bila morda navajena pri tehnologijah relacijskih podatkovnih baz. Torej je lahko pomanjkanje podpore v ključnih trenutkih delovanja oziroma izpada zelo pomemben faktor.
- izbira orodij za obdelavo in analitiko – Na področju obdelave podatkov pri NoSQL pogrešamo množico orodij, med katerimi bi se odločali glede na njihove funkcionalnosti. Ker je podlaga za NoSQL moderna spletna arhitektura t.i. Web 2.0, vsa podjetja v kratkem času tem zahtevam ne morejo slediti, ampak bi morala predhodno izvesti investicijo v prenovo svojih spletnih portalov, ki bi bili potem pripravljeni za pravilen zajem podatkov (npr. v formatu XML). Na žalost je pri obdelavi potrebno imeti vseeno še nekaj znanja o programiranju, obstoječa orodja in aplikacije, ki jih podjetja že imajo kupljena, pa ne omogočajo povezovanja na NoSQL podatkovne baze.
- obvladovanje administracije – Kljub obljubam NoSQL, da ne bo potrebno imeti nikakršne administracije podatkovne baze, je potrebno imeti veliko znanja – tako pri nameščanju kot za vzdrževanje te podatkovne baze.
- ni še dobre prakse – Glede na dolžino obstoja NoSQL še ni dovolj dobrih razvijalcev, vendar se ta vpliv s časom zmanjšuje, v vsakem primeru pa je več razvijalcev v relacijskih podatkovnih bazah kot v NoSQL podatkovnih bazah.

Vendarle pa Richard (2015) poudari tudi nekatere ključne lastnosti, zaradi katerih se podjetje kljub slabostim lahko vseeno odloči za NoSQL podatkovno bazo, in sicer zaradi:

- zelo prilagodljive skalabilnosti – Relacijskim podatkovnim bazam ni mogoče enostavno povečevati skalabilnost, NoSQL podatkovne baze pa so bile že osnovane na način, da se skalabilnost enostavno povečuje.

- obvladovanja velikih količin podatkov – NoSQL brez težav obvladuje masovni zajem podatkov ter obdelavo, za kar relacijske podatkovne baze niso najbolj primerne.
- administracije podatkovne baze – Najboljše relacijske podatkovne baze potrebujejo precej drago vzdrževanje, nadgrajevanje in nameščanje, po drugi strani pa NoSQL potrebuje manj administracije zaradi poenostavljenih modelov in manj optimiziranja.
- ekonomije nakupa – Relacijske podatkovne baze potrebujejo dražja diskovna polja, NoSQL pa lahko uporablja diske, ki so cenovno zelo dostopni in se jih lahko ceneje dokupuje, kar ima za posledico preprosto dejstvo, da lahko pri NoSQL ceneje obvladujemo prostor za velike količine podatkov.

3.3.2 Primer NoSQL podatkovnih baz

Najbolj pogosto omenjene NoSQL podatkovne baze v literaturi (NoSQL, 2016), ki so že dovolj stabilne in uporabne, lahko po naši lastni izbiri zvrstimo v:

- Cassandro kot predstavnico stolpčne NoSQL podatkovne baze,
- MongoDB kot predstavnika dokumentne NoSQL podatkovne baze in
- Oracle NoSQL Database kot predstavnika NoSQL podatkovne baze tipa ključ – vrednost.

Cassandra (NoSQL, 2016) je namenjena za ogromne količine podatkov, ki jih hranimo v stolpcih, lahko uporablja particije in arhitekturo, kjer so vsi strežniki v gruči med seboj neodvisni. Performance ima linearno padajoče – glede na količino podatkov. Ima več točk povrnitve v prvotno stanje po incidentih, lahko bere in piše v več podatkovnih strežnikov ter strežnikih v oblaku, napisana je v programskem jeziku Java. Za dostop do podatkov se uporablja API in poizvedovalni metodi, kot sta CQL in Thrift. Za kompresijo podatkov uporablja že vgrajeno stiskanje podatkov ter primarne in sekundarne indekse.

Ker kot osnovo uporablja dokumente, MongoDB (NoSQL, 2016), lahko zagotavlja podporo več različnim shemam, kjer imajo lahko dokumenti različne strukture in polja. Uporablja torej shranjevanje podatkov v dokumentih in izmenjavo podatkov preko BSON (angl. *Binary JSON*, v nadaljevanju BSON), ki je dejansko binarna predstavitev JSON (angl. *JavaScript Object Notation*, v nadaljevanju JSON) dokumentov. Uporablja avtomatično drobljenje podatkov v zbirkah dokumentov, kar pomeni, da avtomatsko določa, katera polja se zapisujejo v določene dokumente. To posledično pomeni, da so lahko podatki razdrobljeni na več različnih sistemih. To omogoča tudi vodoravno skalabilnost – drugače povedano: MongoDB se razdrobi po različnih strežnikih na različnih lokacijah.

Zelo znan ponudnik relacijskih podatkovnih baz je ponudil tudi tovrstno tehnologijo. Oracle NoSQL Database je skalabilna, porazdeljena NoSQL podatkovna baza, narejena za visoko zanesljivo delovanje, fleksibilnost in razpoložljivo na različnih konfiguracijskih

zasnovah. Lahko jo modeliramo kot tabele relacijskega podatkovnega modela, kot JSON dokumente ali kot parice ključ – vrednost. Oracle NoSQL je porazdeljena (nič ne deli z ostalimi strežniki) preko uniformno porazdeljenih gruč (po določenem ključu glede na primarni ključ). Vsak delček celotne podatkovne baze ima replikacijo in tako zagotavlja visoko zanesljivost delovanja, ob izpadih podatkovnih sistemov hiter preklop na rezervno kopijo podatkov in optimalno uravnavanje obremenjenosti strežnikov. Zagotavlja programske jezike kot so: Java, C in Python ter standardne API-je za lažje delo razvijalcev.

3.3.3 Skalabilnost

Vodoravna skalabilnost pomeni, da se ob povečanju obremenitev nad računalniško arhitekturo dodajajo v računalniško arhitekturo novi strežniki brez zaustavitve računalniškega sistema. Po drugi strani pa navpična skalabilnost pomeni, da se obstoječim strežnikom dodaja več moči (večji pomnilnik, hitrejši CPU z več jedri).

Malce več besed bom namenil vodoravni skalabilnosti, saj je to zelo pomembno pri NoSQL podatkovnih bazah. Pri prebiranju člankov in komentarjev (Burleson, b.l.; Nolting, 2013; Bordei, 2014; Burleson, 2016; Can MySQL scale horizontally?, 2016; NoSQL, 2016) je vsem skupno vsaj naslednje:

- NoSQL relacijske podatkovne baze niso bile nikoli pripravljene niti dizajnirane za uporabo in potrebe trga so se spremenile, saj morajo biti aplikacije oziroma programi na voljo kjerkoli na svetu ob vsakem trenutku.
- Relacijsko podatkovno bazo sicer lahko relativno enostavno preklopimo z enega strežnika v NoSQL – torej v delovanje na več strežnikih (pretvorba vertikalne skalabilnosti neposredno v vodoravno skalabilnost), vendar aplikacija, ki je delovala na originalni podatkovni bazi (pred pretvorbo), niti slučajno ne bo delovala na več strežnikih, saj ni več relacijska podatkovna baza, ki hrani podatke v relacijah, ampak shranjuje podatke v formatu ključ-vrednost.
- Uporabljajo delno rešitev za vodoravno skalabilnost s centralnimi sinhronizacijskimi tabelami za aplikacijo na večjem številu strežnikov po svetu.
- NoSQL je primeren za obvladovanje (pisanje, branje) večjega števila podatkov, poizvedbe med milijardami zapisov in grafično obdelavo podatkov.
- V relacijski podatkovni bazi je še vedno možno napisati poizvedbo (angl. *query*), ki slabo sestavljena lahko povzroči – ali zastoj delovanja ali pa zelo počasno delovanje podatkovne baze, velikokrat pa je napaka na strani kode v programih (aplikacijah).
- Trendi gredo večinoma v smeri kombinacije podatkovne baze v pomnilniku in NoSQL, kot sta Cassandra in MongoDB.

Primeri uporabe NoSQL - predstavili smo delovanje relacijske podatkovne baze ter poslovne potrebe, ki se pojavljajo v današnjem poslovnem okolju. Relacijske podatkovne baze so sicer v svoji osnovi strukturirane in nespremenjene desetletja, vendar se v

določenih poslovnih primerih ne morejo kosati s koncepti NoSQL, za katere lahko predstavimo klasične primere uporabe (Gupta et al., 2013; Bordei, 2014; Can MySQL scale horizontally?, 2016):

- indeksiranje obiskanih spletnih strani v spletu dotičnega podjetja,
- kako biti stroškovno učinkovit in zagotavljati visoko sistemsko razpoložljivost ter restavriranje podatkov ob nesrečah,
- zapisovanje z več kot 300.000 zapisi na sekundo,
- kako prebrati učinkovito iz podatkovne baze velikosti PT (angl. *Peta Byte*), kar pomeni 1 PB = 1.000.000.000.000.000 bajtov = 10^{15} bajtov = 1.000 T bajtov),
- pridobivanje podatkov za risanje grafov (x os, y os),
- vpisovanje in branje mora biti neodvisno od količine podatkov, ki je že shranjena v podatkovni bazi,
- aplikacije za napovedovanje vremena, ki potrebujejo hitre odgovore – glede na spremembe okoliščin za napoved vremena,
- aplikacije za trgovanje, kjer so potrebne hitre »kaj če« analize – glede na količino podatkov,
- aplikacije za socialna omrežja,
- aplikacije za namen umetne inteligence,
- preusmerjanje številk IP (angl. *Internet Protocol*) v omrežjih telekom podjetij.

Novodobni pristop do obdelave velike količine podatkov pa, poleg novejšega pristopa z NoSQL, ponujajo pristope, ki ne potrebujejo velikega predznanja informatike, po drugi strani pa tudi ne potrebujejo velikih vložkov v infrastrukturo IT z drago strojno in programsko opremo. Zadostujejo le orodja, ki intuitivno pomagajo uporabnikom obdelovati velike količine podatkov in samodejno predlagajo tiste analize ter statistike, ki jih na osnovi pred-analiz določijo z vzorčenjem. Dva primera, ki ju bomo predstavili, sta Qlik in Microsoft Power BI.

Qlik (Qlik, b.l.) je močno in zmogljivo analitično orodje, ki za izdelavo portfelja za pripravo poročil ne potrebuje izkušenj strokovnjakov IT, ampak naj bi bil dovolj intuitiven, da ga tudi nepodkovan IT uporabnik lahko sam uporablja. To rešitev podjetja izbirajo za svoja poslovna orodja iz naslova poslovne inteligence (angl. *business intelligence*), saj ni potrebno graditi posebnega odlagališča podatkov, kot npr. pri tradicionalni arhitekturi sistema podatkovne analitike. Qlik omogoča preprost zajem metapodatkov na virih, ki jih nato preprosto prestavi (vizualizira) v berljivo obliko, npr. v grafe, tabele … Obdeloval oziroma analiziral naj bi veliko količino in jih tudi prikazoval, podporo pa naj bi nudil različnim nivojem uporabnikov. Orodje je prirejeno za uporabnike, ki poznajo poslovne zahteve, vendar zaradi svojega predznanja ne znajo konstituirati zahtev za izdelavo sistema podatkovne analitike, bodisi zaradi časovnih stisk ali pa hitro spreminjajočih se zahtev po različnih poročilih in analizah. Za uporabo ni potrebno poznati orodij ne jezika SQL, niti

ne povezovanja med tabelami (podatkov). Vse rezultate poizvedovanj Qlik shranjuje v centralen repozitorij, od koder so poizvedbe na voljo v oblačni rešitvi ali kot strežniška aplikacija. Ponuja integracijo z različnimi podatkovnimi bazami (MySQL, Oracle, PostgreSQL, Sybase, Teradata) in tudi generičnimi povezovalniki (angl. *ODBC*). Grafično je vmesnik za delo z Qlik privlačnega izgleda in je lahko razumljiv ter preprost za uporabo, isto izdelano analizo pa lahko delimo na več (mobilnih) naprav. Slednje pomeni, da se vsebina prikazuje različno od tipa naprave, saj so na mobilnih telefonih ekrani manjši od prenosnikov ter mora zato biti prikazovanje podatkov prilagojeno medijem oziroma aparatu, ki skrbi za predstavitev (vizualizacijo). Odprta podpora za vmesnike API omogoča hitro prilagodljivost in povezljivost (integracijo) z že obstoječimi aplikacijami podjetij, omogoča pa tudi implicitne (angl. *embedded*) aplikacije.

Microsoft je poleg podatkovnega strežnika SQL Server ponudil še orodje Power BI (Microsoft Power BI, b.l.) za poslovno analitiko podatkov. Slednje obljublja, da lahko podjetje pomen podatkov, ki so v tabelah Excel, oblačnih servisih ali v podatkovnih bazah na lokaciji podjetja, enostavno združi na skupen imenovalec in jih predstavi. Predstavitev podatkov z najboljšo uporabniško izkušnjo potrebuje tako obliko, da bo privlačna – tako na vpogled kot tudi z množico ponujenih funkcionalnosti. Orodje zagotavlja enostavno izdelavo preprostih analiz, prikaz iz različnih virov podatkov pa naj bi bil izdelan na način, da uporabnik vedno pridobi pravilne podatke iz Power BI. Z vmesniki API se lahko Power BI poveže z aplikacijami, ki jih podjetje že poseduje – bodisi v oblaku bodisi na namenskih podatkovnih strežnikih (angl. *on-premises*). Za uporabo naj bi bil preprost tudi na različnih prenosnih (mobilnih) napravah, saj ima Power BI vgrajen močen grafični vmesnik, ki kompleksne podatkovne strukture prikaže v očem prijazni obliki. Orodje zelo dobro, brez podrobnega poznavanja virov, prenese metapodatke v Excel ali v lastno okolje, kjer lahko enostavno in hitro izdelate analizo npr. poslovanja prodaje v preteklem letu, preverite, zakaj so se določene anomalije dogajale, kakšna je napoved in kaj lahko storite za preprečitev slabe napovedi. Ena izmed lastnosti, ki je bila resnično pravi preboj pri analitiki, je pisanje povpraševanja, kjer ni potrebno poznati osnov jezika SQL, ampak je orodje narejeno tako intuitivno, da pri pisanju poizvedbe sproti ponuja možnosti, katere podatke želimo pridobiti iz množice podatkov v tabelah. Na prvi pogled lahko to rešitev uporabimo kot osamosvojitev od storitev informatikov oziroma IT specialistov, kar se na dolgi rok lahko izkaže kot slaba praksa.

Z uvedbo sistemov, ki avtomatsko črpajo podatke iz različnih virov ter jih shranjujejo in nam dajo na voljo lepa grafična orodja za intuitivne urejevalce poizvedb SQL, ne odpravimo težav pojavljanja t.i. informacijskih silosov. Avtomatsko zajemanje in pripravljanje prednastavljenih poročil ni problematično zaradi miselnosti, da v informatiki potrebujemo manj kadra. Problem se pojavi šele kasneje, ko tak sistem že uporabljamo in se nam lahko zgodi, da imamo zaradi večkratnih iteracij priklopa virov različne skupne imenovalce. Naše mnenje je, da je potrebno taka avtomatična orodja uporabljati in vpeljevati previdno, najbolj na način, da imamo še vedno vpliv na tiste začetne gradnike

virov metapodatkov, preko katerih sami določimo, kako združevati vire ter poskrbeti, da imamo vedno isto granularnost podatkov.

4 PRIMERJALNA ANALIZA IZVEDENK ARHITEKTUR

4.1 Povezave in soodvisnosti določenih arhitektur

Vse arhitekture so povezane na osnovni načelni ravni – zajeti morajo podatke na virih ter jih primerno distribuirati in omogočiti za analize, nadaljnje obdelave in statistike. Glede na predstavljene najbolj pogoste arhitekture je na trgu dostopnih že več tehnologij, ki se nanje več ali manj že prilegajo, v vsakem izmed primerov pa je potrebno vedno gledati na maksimalne ekonomske učinke posamezne arhitekture in tehnologije, kjer je poudarek na odločitvi podjetja, katero arhitekturo ter tehnologijo bo izbralo v danem trenutku ob poznanih dejstvih.

Povedano še malce drugače, se lahko arhitekture razlikujejo v podrobnostih, saj se za arhitekturo podjetje odloči glede na svoje potrebe, vire in znanja, izbere pa tehnologijo, ki je trenutno na trgu najbolj ali popularna ali uporabna. Pomemben faktor pri odločanju mora biti tudi obdobje, za koliko časa v prihodnosti želimo imeti podatkovno analitiko. V primeru, če izberemo tehnologijo ki je v zatonu, ne moremo pričakovati, da bo ta še sledila svetovnim trendom. Po drugi strani pa se moramo zavedati, da v primeru ta trenutek najbolj nove in sveže tehnologije ne bomo posedovali dobrih praks, ampak bomo na tem področju vlagali veliko v razvojne aktivnosti, katerih učinki bodo do izraza prišli malce kasneje. Konkurenčna prednost pred podjetji je mnogokrat tudi odločitev podjetja, koliko želi vlagati v analitiko svojega delovanja in se želi s pridobljenimi rezultati razlikovati od konkurence. Prepričani smo, da je obdobje spoznanja podjetja, ko ugotovi, da brez poslovne analitike ni mogoče več (učinkovito) poslovati z mislijo biti korak pred konkurenco, neprecenljivo in bi se moralo zgoditi vsakemu podjetju, ki želi še dolgo poslovati in ponujati napredne, trgu prilagojene, rešitve ter izdelke.

4.2 Način primerjave med posameznimi arhitekturami

Zorni kot, iz katerega smo primerjali med seboj arhitekture sistemov podatkovne analitike, je pravzaprav več kot samo eden. Arhitekturo lahko gledamo iz zornega kota, kateri izmed najboljših arhitektur se lahko približamo s svojimi sredstvi in omejitvami, vendar moramo upoštevati tudi ponudbo tehnologij, ki nam bodo izbrano arhitekturo sistema podatkovne analitike tudi podprle.

O tradicionalni arhitekturi lahko razmišljamo predvsem v primerih, ko imamo ali že vzpostavljeno strojno in programsko opremo v dotičnem primeru, saj je ta pristop trenutno najdražji, ker zahteva velika vlaganja v opremo. Veliko je odvisno tudi od branže, pa tudi katere podatke želimo analizirati, saj lahko v določenih primerih najdemo že namenske arhitekture, ki so bile namenoma izdelane.

Poudarjamo, da so si arhitekture v zasnovi podobne, saj vse obravnavajo zajem podatkov v heterogenih virih, pretvorbo in čiščenje podatkov ter pretok podatkov v področje obdelave podatkov. Slednje je tisto področje, zaradi katerih se arhitekture med seboj najbolj razlikujejo, saj vsaka arhitektura obravnava pretočene podatke na lastni način.

4.3 Kriteriji za izbiro izvedenke arhitekture in izbira kriterijev

Skozi literaturo smo spoznali in združevali različne kriterije, ki se pojavljajo kot povzetki ali kot ideje, kaj dejansko je pomembno za podjetje, da se odloči za dotično arhitekturo. Naši predlogi temeljijo na osnovi literature in lastnih izkušenj, zato v tem poglavju združujemo teorijo, prakso (v okoljih IT) po svetu ter lastni doprinos izkušenj.

Na podlagi literature in preučenih primerov uporabe smo povzeli pogoste dejavnike, ki so pomembni pri izbiri izvedenke arhitekture (Buyer's Guide, b.l.; Hwang & Capel, 2002; Earls, 2003; Sen & Sinha, 2005; Clarry, 2006):

- kakovost poslovnih zahtev v smislu natančne opredeljenosti želenega stanja,
- pričakovan čas dobave sistema podatkovne analitike v uporabo,
- viri v oddelku IT,
- stroški razvoja, uvedbe in vzdrževanja,
- kako dobro poznamo podatkovni model podatkov v virih,
- skalabilnost,
- lastnosti ACID ter
- enostavnost uporabe in razumljivost podatkov v podatkovnem skladišču.

V ožjem izboru so dejavniki (Knabke & Olbrich, b.l.; Sullivan, b.l.; Kernochan, 2015; Knowlton, 2015; Richards, 2015), na osnovi katerih lahko dovolj točno predvidimo, za kakšno vrsto podatkovne analitike (sprva arhitekturo, nato tehnologijo) naj se podjetje odloči. Iz naslova neodvisnosti se posebej za vodilne ponudnike (Oracle, Microsoft, IBM, SAP) poslovne analitike nismo opredeljevali, tako da bomo obravnavali dejavnike ne glede na ponudnika oziroma njegov prodajni katalog in cenovno politiko (licence, vzdrževanje).

Glavni dejavniki so naslednji:

- skalabilnost,
- moč oddelka IT,
- kakovost oziroma strukturiranost vira podatkov,
- lastnosti ACID in
- hitrost delovanja.

4.4 Primerjalna analiza ter njen pomen za odločanje

Primerjalno analizo bomo izvedli glede na predstavljene arhitekture in upoštevaje najpomembnejše kriterije ter predstavili, katera arhitektura bi bila najprimernejša za določeno kombinacijo kriterijev. V delu so predstavljene osnovne arhitekture in izpeljanke, vendar bomo pod drobnogled uvrstili tradicionalno arhitekturo sistema podatkovne analitike, arhitekturo sistema podatkovne analitike z uporabo pomnilniške podatkovne baze ter arhitekturo sistema podatkovne analitike z distribuirano podatkovno bazo – NoSQL. Analizo bomo predstavili na različnih primerih, ki se po našem mnenju najpogosteje pojavljajo in so, glede na dolgoletno prakso, v okoljih IT najpogostejši.

Skalabilnost rezultira v vodoravni ali vertikalni skalabilnosti, vodoravni za postavitev podatkovne analitike na geografsko razpršenih lokacijah (več strežnikov), vertikalni pa za razširitev obstoječe infrastrukture IT (več delovnega pomnilnika, SSD, itd).

Moč oddelka IT rezultira v ocenah med slabo in močno, s čimer se lahko prepričamo, kako dober oddelek IT na področju tehnologije ima dotično podjetje; izraža pa se tudi v smeri, kako dobro pozna podjetje podatke v virih ter kako visok je nivo znanja zaposlenih v oddelku IT.

Vir podatkov obravnavamo kot polno strukturirane zajete podatke (npr. poslovno informacijski sistem, informacijski sistem), delno strukturirane zajete podatke, ko ni vnaprej znano, kako se bodo podatki polnili zaradi različnih odjemalcev, operacijskih sistemov ipd, ter nestrukturirane zajete podatke, ko je skupen le osnovni atribut zajema (npr. identifikator zajema) ter vrednost (vsebina kot je tekstovna datoteka, glasbena datoteka, datoteka v formatu XML, grafična datoteka). Slednji primer je precej pogost v raziskovanju raznovrstnosti vsebin, ki se zajemajo.

Lastnosti ACID, ki smo jih že podrobneje predstavili, lahko dosegamo v popolnosti ali pa le delno za določeno področje, atomarnost (A), konsistentnost (C), izolacijo (I) in trajnost (D). V primerjalno analizi smo zapisali, katere od ACID lastnosti priporočamo za izbrani tip podjetja. Omeniti moramo še to, da so ocene subjektivne in se lahko razlikujejo od mnenj bralca.

Hitrost delovanja je želen in pričakovan čas od začetka vpeljave podatkovne analitike do prvih rezultatov, ko so na voljo uporabnikom sistema podatkovne analitike. Hitrost lahko tudi določa zahtevo, kako hiter mora biti odziv na dopolnjevanje in nadgrajevanje uvedene podatkovne analitike z novimi analizami.

Tabela 5: Predlagana izvedenka arhitekture sistema

Tip	**Skalab.**	**IT**	**Vir podatkov**	**ACID**	**Hitrost delovanja**	**Predlagana izvedenka arhitekture sistema**	**Primeri ponudnikov**
A	Horiz.	Močan	Delno strukturiran (zajem s spleta)	A, D	Hitro	Pomnilniška arhitektura z distribuirano podatkovno bazo NoSQL	Aerospike GemfireDB Hazelcast eXtremeDB OpenLDAP Redis
B	Horiz.	Slab	Nestruktur.	Karkoli	Manj pomembna	Arhitektura z distribuirano podatkovno bazo NoSQL	Oracle NoSQL MongoDB Cassandra MarkLogic Server Amazon Dynamo DB
B	Horiz.	Dober	Profili za mobilne naprave Spletni nakupni katalogi Upravljanje z vsebino	A,C,D	Hitro	Arhitektura z distribuirano podatkovno bazo NoSQL	Oracle NoSQL MongoDB Cassandra MarkLogic Server Amazon Dynamo DB
C	Vertik.	Slab	Strukturiran (PIS) Delno strukt.	A,C,D	Karkoli	Tradicionalna arhitektura z relacijsko podatkovno bazo	Oracle DB Microsoft SQL Server MySQL IBM DB2 SAP
C	Vertik.	Dober	Delno strukt.	A,C,I,D	Hitro	Tradicionalna arhitektura z relacijsko podatkovno bazo	Oracle DB Microsoft SQL Server MySQL IBM DB2 SAP
C	Vertik.	Močan	Strukturiran (PIS, IS)	A,C,I,D	Karkoli	Tradicionalna arhitektura z relacijsko podatkovno bazo	Oracle DB Microsoft SQL Server MySQL IBM DB2 SAP
D	Vertik.	Močan	Strukturiran (PIS, IS)	A,C,I,D	Hitro	Pomnilniška arhitektura z relacijsko podatkovno bazo	Oracle TimesTen SAP HANA SolidDB VoltDB

Podjetja (tip A), ki v osnovi zahtevajo horizontalno skalabilnost, imajo močan IT s stališča razpoložljivih sredstev za vlaganje v strojno opremo in sistemsko programsko opremo, kot glavno zahtevo funkcionalnosti pa imajo predvsem hitro zajemanje delno strukturiranih

podatkov s spleta. Ker je v takih okoliščinah – zaradi raznolikosti ponudnikovih arhitektur spletnih odjemalcev, težko zajemati iste podatke od vseh spletnih odjemalcev, pridobljeni podatki ne morejo biti strukturirani, zahtevana pa sta atomarnosti (A – podatek se mora zajeti v celoti) in trajnost (D – da se zajeti podatek tudi shrani za nadaljnjo obdelavo). Ti dve ACID lastnosti sta nujni, saj mora podjetje zajeti in shraniti vse, kar je možno zajeti v danem trenutku. Delovanje pri tem mora biti izredno hitro, saj gre za zajemanje podatkov s spletnih strani, kjer je lahko obiskov spletnih strani podjetja nekaj deset tisoč na sekundo. Za tako delovanje ob predpostavki, da imamo močan oddelek IT v smislu dobre tehnologije in zaposlene z izkušnjami, bi bila v večini primerov ustrezna uporaba pomnilniške arhitekture, kjer zaradi nestrukturiranosti in vnaprejšnjega podatkovnega modela zajemamo podatke, ki se pretakajo v NoSQL podatkovno bazo. Izbrani ponudniki te izvedenke so zbrani na podlagi najbolj pogostih omemb v posameznih spletnih virih. Predstavniki podjetij, ki so kandidati za tip A, so praktična vsa podjetja s podatkovno analitiko, ki so želela zajeti ogromno količino podatkov v zelo kratkem času in še niso imela na voljo podatkovne baze NoSQL.

Pri tipu B gre za enak princip zahteve po horizontalni skalabilnosti. Ker imajo zaradi narave dela podjetja slabše organiziran oddelek IT z manj sredstvi za vlaganje v tehnologijo, zajemajo nestrukturirane podatke (zaradi narave zajema ni mogoče predvideti kakovosti strukture podatkov). Lastnosti ACID in hitrost delovanja so manj ali celo nepomembni, zato za to izvedenko priporočamo arhitekturo z distribuirano podatkovno bazo NoSQL. Ponudniki za tako izvedenko so dobro znani in se med seboj razlikujejo po različnih implementacijah. Pri tem pa priporočamo, da taka podjetja – zaradi slabšega oddelka IT, izberejo licenčno ugodne ponudnike (uporaba rešitev z brezplačnimi uporabniškimi licencami).

Poleg že omenjene kombinacije smo zasledili še nekaj primerov, kateri se od prejšnje kombinacije razlikujejo v tem, da imajo dober IT v smislu znanja svojih zaposlenih (ali preko izvajanja storitev zunanjih svetovalcev) in več sredstev za investiranje v tehnologijo kot v prejšnjem primeru (ko se zaradi pomanjkanja sredstev podjetje odloči za brezplačne uporabniške licence). Zajemajo se podatkovne strukture, ki so že dobro definirane in v precejšnji meri dobro vnaprej pripravljene, saj gre za vsebinsko pomembne podatke in so zaradi tega zahtevane lastnosti atomarnosti, konsistentnosti (da so podatki med seboj vedno v določenem urejenem stanju) ter trajnost (D – saj morajo biti podatki zaradi vsebine v vsakem primeru zapisani v podatkovno bazo). Velika hitrost je pomembna, saj je hipna odzivnost aplikacij s strani uporabnikov pričakovana in ključna, še posebej ker uporabniki potrebujejo hitre odzivne čase pridobivanja podatkov ter hitre vpise uporabniških sprememb. V tem primeru uporabniki pričakujejo hitro odzivnost, sicer spletne strani ali spletne aplikacije ne bodo več uporabljali. V tem primeru podjetju zaradi boljše tehnologije v IT že lahko priporočamo vpeljavo komercialne (licenčno plačljive) različice opisanih ponudnikov strojne in programske opreme. Tipični kandidati za tip B bi bila podjetja, ki se primarno ukvarjajo s profiliranjem uporabnikov ter z vsebino (komentarji na

blogih in forumih, slikah, video posnetkih) nudijo rešitve za socialno mreženje uporabnikov in zagotavljanje storitev na mobilnih napravah ipd.

Tip C zahteva vertikalno skalabilnost, kar pomeni, da so zaradi slabega IT potrebni manjši vložki v infrastrukturo (ker še ne potrebujemo distribuiranega okolja). Podatki so delno strukturirani ali gre le za poslovno informacijske sisteme, kjer ne potrebujemo ACID lastnosti I (izolacije – ker ne pričakujemo, da bo isti uporabnik na dveh različnih napravah dostopal hkrati ob istem času do istega podatka). Hitrost sicer ni pomembna, je pa pomembno dejstvo, da gre za izvedenko tradicionalne arhitekture z relacijsko podatkovno bazo. Ravno slednje nam v tem primeru nudi različice arhitektur, ki se predvidoma ne bi smele hitro spreminjati. Zaradi slabega oddelka IT je na voljo tudi manj finančnih sredstev za nakup tehnologije in zato predlagamo na osredotočenje izbire ponudnikov relacijskih podatkovnih baz v smeri, kjer so licence brezplačne ali vsaj brezplačne za določeno količino shranjenih podatkov.

Možna je tudi kombinacija, ko imamo poleg zahteve po vertikalni skalabilnosti z dobrim oddelkom IT z več sredstvi za nakup licenc ter delno strukturiranimi podatki – zaradi hitrega delovanja pri ACID lastnostih, obvezen tudi I (kar pomeni, da lahko hkrati dostopamo do istih podatkov). Tu še vedno priporočamo izbiro izvedenke tradicionalne arhitekture podatkovne analitike s plačljivimi licencami za relacijsko podatkovno bazo pri izbiri ponudnika.

Dodatno smo opredelili tudi možnost, da se glede na kombinacijo tipa C, definiranega v prejšnjem odstavku, lahko z dobrim oddelkom IT, delno strukturiranimi podatki ter polno zahtevanimi lastnostmi ACID še vedno odločimo za izvedenko s tradicionalno arhitekturo z relacijsko podatkovno bazo.

Nasprotno od zadnjih treh kombinacij predstavljamo kombinacijo, ki določa tip D, in sicer z zahtevo po vertikalni skalabilnosti, močnim oddelkom IT s tehnologijo in znanjem, strukturiranimi podatki v virih, polno zahtevo po lastnostih ACID ter hitrim delovanjem. V večini primerov priporočamo za tovrstno delovanje izvedenke pomnilniške arhitekture z relacijsko podatkovno bazo. Kandidati za tip D so sistemi, ki so primerni za preusmerjanje internetnega prometa, podatkovne baze glasbene industrije (npr. predvajalniki glasbe), blagovne tržnice oziroma borze blaga, rezervacije letalskih kart, klicni centri in področje igralništva preko spleta.

SKLEP

Arhitekturo sistema podatkovne analitike največkrat določa lastna arhitektura transakcijskega sistema. Ta teza skozi leta izgublja svoj pomen, saj se zadnja leta spreminjajo tako obsegi podatkov ki jih analiziramo, kot tudi nove poslovne potrebe, ki jih narekujejo novi poslovni modeli ter nove tržne niše. Če je bilo pred prihodom oziroma razcvetom spleta in spletnih tehnologij dovolj za svoje potrebe imeti zagotovljeno hrambo vseh poslovnih dogodkov, npr. prodaje izdelkov po državah, se potrebe po analiziranju obstoječih podatkov selijo tudi na področje, ki pred desetletji ni bilo še tako pomembno. Spletne tehnologije in hitrejša ter cenovno dostopnejša strojna oprema so pospešile razmišljanja in potrebe, v katera področja se bo analiza podatkov v prihodnjih letih še hitreje premikala. Če nas pred npr. desetimi leti še ni tako podrobno zanimalo, kaj dejansko se dogaja s kupci na naših spletnih straneh (ali se tega nismo vprašali pri analizi nakupovalnih navad ali pa še ni bilo prave tehnologije za izvedbo analize le-tega), je dandanes slika popolnoma drugačna.

Dejstvo, da moramo še vedno spremljati količino posameznega prodanega blaga ali storitev po lokacijah, mesecih in še katerih drugih dimenzijah, ostaja Kot najmočnejša potreba pa v ospredje sili ozaveščanje, da je na osnovi pridobljenih podatkov npr. iz prodaje, lokacije, strank in časa težko izluščiti ali poiskati informacije, kako so podatki pravzaprav nastali, oziroma kakšne so navade obiskovalcev spletnih strani in po čem pravzaprav obiskovalci spletnih strani poizvedujejo (npr. pri iskanju primerljivih izdelkov, iskanju po ključnih besedah, itd). Konkreten primer, ko navadna relacijska podatkovna baza (angl. *Relational Database Management System*, v nadaljevanju RDMS) ne pokriva več potreb po analizi – skladno s svojo arhitekturo in optimiziranim iskanjem, je primer Wikipedie in prikaz iskanja določenih tekstovnih nizov (Loaiza, 2015). Kljub temu, da so pomnilniške podatkovne baze na pohodu in bodo verjetno kmalu temelji arhitektur sistemov podatkovne analitike, to še ne pomeni, da bodo tradicionalna podatkovna skladišča izginila, saj je Loshin (b.l.) dobro povzel dejstva, da so novi denarni vložki v novo infrastrukturo pomnilniških podatkovnih baz, nivo zrelosti tehnologije pomnilniških podatkovnih baz (in velika količina že izvedenih projektov s to tehnologijo) ter dobro poznavanje obstoječega podatkovnega skladišča pravzaprav tehtni razlogi, zaradi katerih tradicionalna podatkovna skladišča ne bodo ukinjena in bodo verjetno še vedno obstajala.

Povzetek ugotovitev primerjalne analize arhitektur sistemov podatkovne analitike smo zapisali v tabelo 5, kjer smo vpisali možne kombinacije izvedenke podatkovne analitike na osnovi v nalogi izbranih kriterijev. Izvedenko podatkovne analitike kot prvi izbrani kriterij določa zahtevana skalabilnost sistema, saj je to kriterij, ki nas pri izbiri potencialnih arhitektur najbolj usmerja, da v primeru razširitve obstoječega sistema podatkovne analitike ne trčimo v arhitekturne prepreke oziroma ne zaidemo v stanje, ko bi morali zaradi tehničnih omejitev spreminjati arhitekturo sistema podatkovne analitike. Glede na

preučeno literaturo so podatkovne baze NoSQL najbolj primerne za distribuirane transakcije po geografsko razpršenih področjih, kjer dobijo s podporo delovanja v pomnilniku najboljši možni izkupiček zmogljivosti IT, ki so ta trenutek na voljo. Kljub temu so podatkovne baze NoSQL še vedno tako sveže na trgu, da prave podpore pravzaprav še ni na voljo (še posebej ne v odprto kodnih različicah ali brezplačnih različicah programske opreme), posledično tudi še ni pravega strokovnega kadra, ki bi vse temelje in dobre prakse že poznal. Še vedno pa na področju zagotavljanja vertikalne skalabilnosti najboljšo prakso zagotavljajo izvedenke tradicionalne arhitekture z relacijsko podatkovno bazo (z delovanjem na medijih, kot so SSD ali klasični hitri trdi diski). Velik korak naprej so doprinesle izvedbe pomnilniške arhitekture z relacijskimi podatkovnimi bazami. Opisali smo tudi več različnih obstoječih kombinacij ostalih kriterijev, s katerimi pokrijemo večino scenarijev, ki se pojavljajo v podjetjih.

Uporabnost v praksi primerjalna analiza arhitekture sistemov podatkovne analitike dosega na področju, kjer lahko, glede na potrebe, ki si jih postavimo kot obvezne parametre uspešnega projekta, določimo na obstoječih, že izvedenih projektih, kateri pristop oziroma katera izvedenka arhitekture nam, glede na preostale kriterije, najbolj ustreza. Ob raziskovanju in preučevanju literature na spletu (knjižne različice žal prepočasi sledijo čedalje hitrejšim trendom in tehnologijam) smo naleteli na nekaj omejitev dela, saj se na spletne vire, ki imajo pogosto v ozadju trženjske vzvode, v popolnosti težko zanašamo. Zato smo pri delu obravnavali več različnih pogledov z namenom, da smo lahko postavili na osnovi preučenega gradiva čim bolj objektivne kriterije ter njihovo pomembnost za primerjalno analizo. Prispevek k teoriji je pregled trenutnega stanja na področju arhitektur podatkovne analitike in zbir kriterijev, kateri se objektivno največkrat pojavljajo kot pomembni kriteriji, ki v kontroliranih okoliščinah prinašajo najboljše rezultate.

Nadaljnje delo bo analiza produktnih rešitev različnih ponudnikov platform, saj določeni proizvajalci in snovalci platform poskušajo zagotoviti podporo na vseh možnih izvedenkah arhitektur sistemov podatkovne analitike. Kljub temu, da bosta v prihodnjih letih verjetno prevladovali pomnilniška arhitektura sistema podatkovne analitike in arhitektura sistema podatkovne analitike z distribuirano podatkovno bazo NoSQL, bo tradicionalna arhitektura sistema podatkovne analitike vendarle soobstajala z obema naprednejšima arhitekturama podatkovne analitike, bomo pa spremljali napredek obeh naprednejših arhitektur podatkovne analitike ter s tem dobre in slabe prakse s tema arhitekturama izvedenih projektov. Pokrivati tako širok spekter različnih arhitektur, pristopov in dobrih primerov je osnova za raziskavo, kateri uspešni projekti, ki bodo javno objavljeni, so bili izvedeni z določeno izvedenko arhitekture in kakšni so bili realni kriteriji uspeha glede na kriterije, določene v tem delu.

LITERATURA IN VIRI

1. *ACID*. Najdeno 18. julija 2016 na spletnem naslovu https://msdn.microsoft.com/en-us/library/aa480356.aspx
2. Allen, M. (2015, 9. november). Relational Databases Are Not Designed For Scale. Najdeno 15. junija 2016 na spletnem naslovu http://www.marklogic.com/blog/relational-databases-scale/
3. Ardales, M. (2009, 11. avgust). Types Of Information Systems. Najdeno 30. junija 2016 na spletnem naslovu http://www.slideshare.net/mannyardales/types-of-information-systems
4. *Arhitecture.* Najdeno 27. junija 2016 na spletnem naslovu http://whatis.techtarget.com/definition/architecture
5. Ariyachandra, T., & Watson, H.J. (b.l.). Which Data Warehouse Architecture Is Most Successful? *Business Intelligence Journal*. *11*(1), 4-6.
6. Bordei, A. (2014, 24. september). Scaling NoSQL databases: 5 tips for increasing performance. Najdeno 13. februarja 2016 na spletnem naslovu http://radar.oreilly.com/2014/09/scaling-nosql-databases-5-tips-for-increasing-performance.html
7. Burleson, D. (2015, 15. avgust). Best practices for scaling Oracle VLDB systems. *Oracle tips by Burleson*. Najdeno 3. julija 2016 na spletnem naslovu http://www.dba-oracle.com/t_best_practices_oracle_scaling_vldb.htm
8. Burleson, D. (2016). Scaling an Oracle database: What is the best strategy for you? Najdeno 13. februarja 2016 na spletnem naslovu http://searchbusinessintelligence.techtarget.in/tip/Scaling-an-Oracle-database-What-is-the-best-strategy-for-you
9. Burleson, D. (b.l.). Vertical vs. Horizontal scalability for Oracle. *Oracle Tips by Burleson Consulting*. Najdeno 13. februarja 2016 na naslovu http://www.dba-oracle.com/t_Vertical_vs_Horizontal_scalability_Oracle.htm
10. Buyer's Guide. (b.l.). *A buyer's guide to selecting the best data warehouse.* Najdeno 10. aprila 2016 na spletnem naslovu http://searchdatamanagement.techtarget.com/buyersguide/A-buyers-guide-to-selecting-the-best-data-warehouse-product
11. *Can MySQL scale horizontally?* Najdeno 13. februarja 2016 na spletnem naslovu https://www.quora.com/Can-MySQL-scale-horizontally
12. Clarry, M. (2006). Ten Mistakes to Avoid When Building A Data Warehouse Team. *Connect - The Knowledge Network*. Najdeno 12. decembra 2006 na spletnem naslovu http://download.101com.com/pub/tdwi/Files/10_Mistakes_Avoid_Building_a_DW_Team.pdf
13. Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., & Woodford, D. (2012). Spanner: Google's Globally-Distributed Database. *Proc. USENIX Symp. Operating Syst. Des. Implementation*, *31*(3), 251-264.
14. Database Concepts. (b.l.). *10 – Transactions*. Najdeno 1. julija 2016 na spletnem naslovu https://docs.oracle.com/database/121/CNCPT/transact.htm#CNCPT016
15. Deliwala, H. (2010, 14. september). Developing a Business Intelligence Roadmap. Najdeno 18. junija 2016 na spletnem naslovu http://www.information-management.com/infodirect/2009_176/BI_CRM_analytics_roadmap-10018719-1.html

16. Deshpande, A. H. (september 2015). Efficient Compression Techniques for an In Memory Database System. *International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)*, *9*(3), 8975-8983.
17. Di Mauro, C., Nordvik, J.P., & Lucia, A.C. (2006). Multi-criteria decision support system and Data Warehouse for designing and monitoring sustainable industrial strategies - an Italian case study. *Institute for the Protection and Security of the Citizen (IPSC)*. Najdeno 12. decembra 2015 na spletnem naslovu http://www.iemss.org/iemss2002/proceedings/pdf/volume%20uno/434_dimauro.pdf
18. Diaconu, C., Freedman, C., Ismert, E., Larson, P. Å., Mittal, P., Stonecipher, R., Verma, N., & Zwilling, M. (2013, 27. junij). Hekaton: SQL Server's Memory-Optimized OLTP Engine. Najdeno 2. maja 2016 na spletni strani http://research.microsoft.com/pubs/193594/Hekaton%20-%20Sigmod2013%20final.pdf
19. Earls, A.R. (2003, 25. avgust). ETL: preparation is the best bet. *Computerworld*. Najdeno 26. aprila 2016 na spletnem naslovu http://www.computerworld.com/article/2571283/business-intelligence/etl--preparation-is-the-best-bet.html
20. *Future of Data Warehouse, Data Mining and Data Visualisation*. Najdeno 22. junija 2016 na spletnem naslovu http://taufiq-loves-bi.tumblr.com/post/397377637/future-of-data-warehouse-data-mining-and-data
21. Gene, P. (2015). *Scalable Transactions for Scalable Distributed Database Systems*. University of California: Berkeley. Najdeno 17. aprila 2016 na spletnem naslovu http://search.proquest.com.nukweb.nuk.uni-lj.si/docview/1730862919/DECF587FD96C43B4PQ/4?accountid=16468
22. *Get started with In-Memory (Preview) in SQL Database*. Najdeno 2. maja 2016 na spletnem naslovu https://azure.microsoft.com/en-us/documentation/articles/sql-database-in-memory/
23. Golfarelli, M., & Rizzi, S. (1998). *A Methodological Framework for Data Warehouse Design, DOLAP*. Bologna: DEIS – University of Bologna.
24. Gorla, N. (2003). Feature to consider in a data warehouse system. *Association for Computing Machinery Communication of the ACM, 46*(11), 111-115.
25. Gowan, J. A., Mathieu, R. G., & Hey, M. B. (2006). Earned value management in a data warehouse project. *Information Management & Computer Security, 14*. USA: Emerald Group Publishing Limited.
26. Grad, J., & Jaklič, J. (1996). *Baze podatkov*. Ljubljana: Ekonomska fakulteta.
27. Gu, J., Goetschalckx, M., & McGinnis, L.F. (2006, 3. maj). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research, 177*(1), 1-21.
28. Gupta, M. K., Verma, V., & Verma, M. S. (2013). In-Memory Database Systems - A Paradigm Shift. *International Journal of Engineering Trends and Technology (IJETT), 6*(6), 333-336.
29. Gutiérrez, A., & Marotta, A. (2000). *An Overview of Data Warehouse Design Approaches and Techniques*. Uruguay: Instituto de Computación. Najdeno 1. aprila 2016 na spletnem naslovu http://repository.binus.ac.id/content/M0274/M027447974.pdf
30. Hackney, D. (2000, 1. april). Data Warehouse Delivery. Najdeno 30. aprila 2016 na spletnem naslovu http://www.egltd.com/dmrarchive/2000-04.pdf
31. Hackney, D. (2000, 1. januar). Data Warehouse Delivery. Najdeno 30. aprila 2016 na spletnem naslovu http://www.egltd.com/dmrarchive/2000-01.pdf

32. Hackney, D. (2000, 1. maj). Data Warehouse Delivery: When to Federate. Najdeno 30. aprila 2016 na spletnem naslovu http://www.egltd.com/dmrarchive/2000-05-2.pdf
33. Haertzen, D. (2008). Data Warehousing Business Requirements. Najdeno 15. aprila 2015 na spletnem naslovu http://www.infogoal.com/datawarehousing/requirements.htm
34. Haertzen, D. (2008). ETL (Extract-Transform-Load) for Data Warehousing. Najdeno 11. aprila 2015 na spletnem naslovu http://www.infogoal.com/datawarehousing/etl.htm
35. *Hard disk drive.* Najdeno 7. februarja 2016 na spletnem naslovu http://searchstorage.techtarget.com/definition/hard-disk-drive
36. Henschen, D. (3. marec 2014). In-Memory Databases: Do You Need The Speed? Najdeno 22. maja 2016 na spletnem naslovu http://www.informationweek.com/big-data/big-data-analytics/in-memory-databases-do-you-need-the-speed/d/d-id/1114076?page_number=1
37. Howard, P. (2012, 14. maj). What exactly is in-memory? Najdeno 26. novembra 2015 na spletnem naslovu http://www.bloorresearch.com/blog/im-blog/memory/
38. *H-Store*. Najdeno 2. maja 2016 na spletnem naslovu http://hstore.cs.brown.edu/
39. Hwang, M., & Capel, J. J. (2002). Data warehouse development and management: Practices of some large companies. *Journal of Computer Information Systems, 43*(1), 3-4.
40. *In which situations is NoSQL better than relational databases such as SQL?* Najdeno 1. julija 2016 na spletnem naslovu https://www.quora.com/In-which-situations-is-NoSQL-better-than-relational-databases-such-as-SQL
41. Inmon, W. H. (2002). *Building the Data Warehouse*, (3rd edition). USA: Wiley.
42. Inmon, W. H., Welch, J. D., & Glassey, K. L. (1997). *Managing the Data Warehouse*. Canada: Wiley.
43. Jagarinec, D. (2005). OLAP in podatkovna skladišča. *Moj mikro*, 78-80.
44. *JDBC Basics*. Najdeno 17. junija 2016 na spletnem naslovu https://docs.oracle.com/javase/tutorial/jdbc/basics/
45. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E. P. C., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J., & Abadi, D. J. (2008). H-store: A high-performance, distributed main memory transaction processing system. *Proc. VLDB Endowment, (1)*, 1496-1499.
46. Kernochan, W. (2015, 23. julij). 3 Limits of NoSQL Data Processing. Najdeno 11. junija 2016 na spletnem naslovu https://www.quora.com/What-are-the-limitations-of-NoSql-databases
47. Kimball, R., & Ross, M. (2002). *The Data Warehouse Toolkit* (2nd Edition). Canada: Wiley.
48. Kimball, R. (1996). Dangerous Preconceptions. *The Data Warehouse Architect, DBMS Magazine (9)*9. Najdeno 14. februarja 2016 na spletnem naslovu http://inethub.olvi.net.ua/ftp/pub/books/programming/db/storage/taki2/olap-course/kimball/9608d05.html
49. Kimball, R., Reeves, L., Thronthwaite, W., & Ross, M. (1998). *The Data Warehouse Lifecycle Toolkit*. USA: Wiley.
50. Knabke, T., & Olbrich, S. (b.l.). Towards agile BI: applying in-memory technology to data warehouse architectures. Najdeno 26. novembra 2015 na spletnem naslovu http://cs.emis.de/LNI/Proceedings/Proceedings193/101.pdf
51. Knowlton, B. (2015, 15. junij). 5 Steps to Start Developing your Business Intelligence Roadmap. Najdeno 18. junija 2016 na spletnem naslovu https://www.linkedin.com/pulse/5-steps-start-developing-your-business-intelligence-roadmap-knowlton

52. Krueger, J., Wust, J., Linkhorst, M., & Plattner, H. (2012). Leveraging Compression in In-Memory Databases. Germany: University of Potsdam.
53. Laurent, W. (2001, 1. december). Best Practices for Data Warehouse Database Developers. *Information Management.* Najdeno 15. decembra 2015 na spletnem naslovu http://www.information-management.com/issues/20011201/4340-1.html
54. Lavezzo, N. (b.l.). What are the limitations of NoSql databases? Najdeno 11. junija 2016 na spletnem naslovu https://www.quora.com/What-are-the-limitations-of-NoSql-databases
55. Li, G., Qian, X. S., & Ye, C. M. (2011). A new architecture for large data warehouse. *Applied Mechanics and Materials, 55-57,* 87-90.
56. Lin, H.Y., Hsu, P.Y., & Sheen, G.J. (2007). A fuzzy-based decision-making for data warehouse system selection. *Expert Systems with Applications, 32*(3), 939-953.
57. Loaiza, J. (2015, 8. februar). Oracle Database In-Memory Demo. Najdeno 10. aprila 2016 na spletnem naslovu https://www.youtube.com/watch?v=mF-h26iKTYY
58. Loshin, D. (b.l.). Data warehouse software not ready for retirement in age of big data. Najdeno 10. aprila 2016 na spletnem naslovu http://searchdatamanagement.techtarget.com/feature/Data-warehouse-software-not-ready-for-retirement-in-age-of-big-data
59. Ma, C., Chou, D.C., & Yen, D.C. (2000). Data warehousing, technology assessment and management. *Industrial Management & Data Systems, 100*(3). UK: MCB UP, 125-135.
60. Mansharamani, R. (2013, april). In Memory Databases: An Overview. Najdeno 9. aprila 2016 na spletnem naslovu http://www.softwareperformanceengineering.com/uploads/1/2/6/6/12667295/imdboverview.pdf
61. Marco, D. (1998). Starting a Data Warehousing Project? *Information management*. Najdeno 11. marca 2016 na spletnem naslovu http://www.information-management.com/infodirect/19980701/923-1.html
62. Microsoft Power BI (b.l.). *Bring your data to life with Microsoft Power BI*. Najdeno 17. junija 2016 na spletnem naslovu https://powerbi.microsoft.com/en-us/
63. Mullins, C. S. (b.l.). Five factors to help select the right data warehouse product. Najdeno 2. junija 2016 na spletnem naslovu http://searchdatamanagement.techtarget.com/feature/Five-factors-to-help-select-the-right-data-warehouse-product
64. Mullins, C.S. (2016). The benefits of deploying a data warehouse platform. Najdeno 10. aprila 2016 na spletnem naslovu http://searchdatamanagement.techtarget.com/feature/The-benefits-of-deploying-a-data-warehouse-platform
65. Nolting, D. (2013, 21. februar). 5 Best Practices for Database Scaling. Najdeno 13. februarja 2016 na naslovu https://www.bluelock.com/blog/5-best-practices-database-scaling/
66. *NoSQL*. Najdeno 6. aprila 2016 na spletnem naslovu http://nosql-database.org/
67. ODBC. (b.l.). *Microsoft Open Database Connectivity (ODBC)*. Najdeno 17. junija 2016 na spletnem naslovu https://msdn.microsoft.com/en-us/library/ms710252(v=vs.85).aspx
68. OLAP. (b.l.). *What is the definition of OLAP?* Najdeno 1. julija 2016 na spletnem naslovu http://olap.com/olap-definition/
69. *Oracle Database In-Memory - ORACLE WHITE* PAPER. Najdeno 22. maja 2016 na spletnem naslovu http://www.oracle.com/technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633.html

70. *Oracle Database In-Memory*. Najdeno 1. maja 2016 na spletnem naslovu http://www.oracle.com/us/products/database/options/database-in-memory/overview/index.html
71. Perkins, A. (2003). Critical Success Factors for Data Warehouse Engineering. Najdeno 2. junija 2016 na spletnem naslovu http://www.visible.com/company/whitepapers/dwcsf.pdf
72. *Platform*. Najdeno 27. junija 2016 na spletnem naslovu http://searchservervirtualization.techtarget.com/definition/platform
73. Qlik. (b.l.). *The complete Qlik® product family*. Najdeno 14. junija 2016 na spletnem naslovu http://www.qlik.com/products
74. Richards, J. (2015, 24. september). Advantages and Disadvantages of NoSQL databases – what you should know. Najdeno 11. junija 2016 na spletnem naslovu http://www.hadoop360.com/blog/advantages-and-disadvantages-of-nosql-databases-what-you-should-k
75. Robbins, S. (2008). RAM is the new disk.... *InfoQ*. Najdeno 13. februarja 2016 na spletnem naslovu http://www.infoq.com/news/2008/06/ram-is-disk
76. Robinson, S. (2004, 13. december). Seven highly effective steps to a smooth data warehouse implementation. Najdeno 1. aprila 2016 na spletnem naslovu http://www.techrepublic.com/article/seven-highly-effective-steps-to-a-smooth-data-warehouse-implementation
77. Robinson, S. (2004, 15. marec). The seven deadly sins of data warehouse implementation development. Najdeno 1. aprila 2015 na spletnem naslovu http://www.techrepublic.com/article/the-seven-deadly-sins-of-data-warehouse-implementation-development/
78. Sen, A., & Sinha, A. P. (2005). A Comparison of Data Warehousing Methodologies. *Association for Computing Machinery, marec 2005, 48*(3), 79-84.
79. Sen, A., Ramamurthy, K. (Ram), & Sinha, A. P. (2012). A Model of Data Warehousing Process Maturity. *IEEE Computer Society, marec/april 2012, 38*(2), 336-353.
80. Shih-Wen, C. (2010, 16. september). Solid-state disk - US 20100235559 A1. Najdeno 7. februarja 2016 na spletnem naslovu http://www.google.com/patents/US20100235559
81. SSD. (b.l.). *SSD (solid-state drive, solid-state disk, solid-state storage drive).* Najdeno 7. junija 2016 na spletnem naslovu http://searchstorage.techtarget.com/definition/solid-state-drive
82. Sullivan, D. (b.l.). Types of NoSQL databases and key criteria for choosing them. Najdeno 15. junija 2016 na spletnem naslovu http://searchdatamanagement.techtarget.com/feature/Key-criteria-for-choosing-different-types-of-NoSQL-databases
83. *The Software Development Life Cycle (SDLC) For Small To Medium Database Applications*. Document ID: REF-0-02. Najdeno 7. februarja 2016 na spletnem naslovu http://www.pelicaneng.com/devdocs/sdlc.pdf
84. *TimesTen.* Najdeno 2. maja 2016 na spletnem naslovu http://www.oracle.com/technetwork/database/database-technologies/timesten/overview/index.html
85. *UNICOM SolidDB.* Najdeno 2. maja 2016 na spletnem naslovu http://unicomsi.com/products/soliddb/
86. *VoltDB.* Najdeno 2. maja 2016 na spletnem naslovu https://voltdb.com/
87. Watson, H. J., Goodhue, D. L., & Wixom, B. H. (2002). The benefits of data warehousing: Why some organizations realize exceptional payoffs. *Information and Management, 39*(6). Nizozemska, 491-502.

88. *What is SAP HANA?* Najdeno 2. maja 2016 na spletnem naslovu http://scn.sap.com/docs/DOC-60338
89. White, C. (2000, 1. marec). The Federated Data Warehouse. *DM Review*. Najdeno 14. aprila 2016 na spletnem naslovu http://www.information-management.com/issues/20000301/1953-1.html
90. Widom, J. (1995). Research Problems in Data Warehousing. *Proc. of 4th Int'l Conference on Information and Knowledge Management (CIKM)*. USA: Stanford University, 25-30.
91. Wu, M.C., & Buchmann, A. P. (1997). Research Issues in Data Warehousing. BTW: German Database Conference, 61-82.
92. Zhang, H., Chen, G., Ooi, B.C., Tan, K.L., & Zhang, M. (2015). In-Memory Big Data Management and Processing: A Survey. *IEEE Transactions on Knowledge and Data Engineering (27)*,7.
93. Zilka, A. (2011). Ari Zilka on RAM is the New Disk & BigMemory. *InfoQ*. Najdeno 13. februarja 2016 na spletnem naslovu http://www.infoq.com/interviews/zilka-bigmemory