Nacionālais veselības dienests

|  |  |
| --- | --- |
|  |  |

Integrācijas platformas informācijas sistēmas izstrāde

Programmatūras izstrādes vadlīnijas

Standarts

NVD.STD.VAD

14.03.2013. versija 1.02

Rīgā 2013

|  |  |  |
| --- | --- | --- |
|  | **IEGULDĪJUMS TAVĀ NĀKOTNĒ** |  |

 Dokumenta identifikācija

|  |  |
| --- | --- |
| Dokumenta ID: | NVD.STD.VAD.1.02  |
| Dokumenta nosaukums: | Integrācijas platformas informācijas sistēmas izstrāde.Programmatūras izstrādes vadlīnijas.Standarts. |
| Dokumenta kods: | NVD.STD.VAD |
| Versija: | Versija 1.02, Laidiens 14.03.2013. |

Saskaņojumi

|  |  |  |  |
| --- | --- | --- | --- |
| Organizācija | Vārds, uzvārds, amats  | Datums | Paraksts |
| Nacionālais veselības dienests | L.Freimane, projekta vadītāja no Pasūtītāja puses |  |  |
| SIA "ABC software" | J.Korņijenko, projekta vadītājs no Izpildītāja puses | 14.03.2013. |  |
| SIA "ABC software" | J.Gekišs, sistēmas arhitekts un projektētājs | 14.03.2013. |  |
| SIA "ABC software" | E.Blumberga, projekta kvalitātes kontroles vadītāja | 14.03.2013. |  |

|  |
| --- |
| *Visas tekstā izmantotās preču zīmes pieder to īpašniekiem un ir izmantotas tikai kā atsauces.* |

Izmaiņu vēsture

|  |  |  |  |
| --- | --- | --- | --- |
| Versija | Datums | Apraksts | Autors |
| 0.01 | 10.10.2011. | Izveidota dokumenta sākotnējā versija  | J.Gekišs, A.Pontags |
| 0.02 | 10.11.2011. | Redakcionālas izmaiņas | J.Gekišs, A.Pontags |
| 1.00 | 19.07.2012. | Publicēta versija 1.00 | J.Korņijenko |
| 1.01 | 15.01.2013. | Redakcionālas izmaiņas | J.Korņijenko |
| 1.02 | 14.03.2013. | Iekļauti labojumi saskaņā ar nodevumu pakas IP standarti izvērtējums Nr. 66 (NVD.IP.NI.66):Sadaļās: 2.1.3.., 2.4.1., 2.4.5.;Paragrāfos: 2.1.1.1., 2.1.4.6;VEC vietā iekļauts NVD (t.sk. dokumenta kodā). | J.Korņijenko |

**Satura rādītājs**

[Attēlu saraksts 6](#_Toc353451124)

[1. Ievads 7](#_Toc353451125)

[1.1. Dokumenta nolūks 7](#_Toc353451126)

[1.2. Termini un pieņemtie apzīmējumi 7](#_Toc353451127)

[1.3. Saistība ar citiem dokumentiem 7](#_Toc353451128)

[1.4. Citu nosaukumu piešķiršanas noteikumu lietošana 8](#_Toc353451129)

[1.5. Aprakstu uzmetums 8](#_Toc353451130)

[1.6. Dokumenta pārskats 8](#_Toc353451131)

[2. MS SQL datu bāzes un.NET programmēšanas vadlīnijas 9](#_Toc353451132)

[2.1. Kopējas kodēšanas vadlīnijas 9](#_Toc353451133)

[2.1.1. Kopējās prasības 9](#_Toc353451134)

[2.1.1.1. Programmēšanas vide 9](#_Toc353451135)

[2.1.1.2. Komponentes atkārtota izmantošana 9](#_Toc353451136)

[2.1.2. Konfigurācijas vadība 9](#_Toc353451137)

[2.1.2.1. Konfigurācijas krātuve 9](#_Toc353451138)

[2.1.2.2. Konfigurācijas informācijas raksturs 10](#_Toc353451139)

[2.1.2.3. Tehnoloģija 10](#_Toc353451140)

[2.1.3. Izpildes trasēšana 10](#_Toc353451141)

[2.1.3.1. Izpildes skaitītāja izvietojums 10](#_Toc353451142)

[2.1.3.2. Izpildes skaitītāju nosaukšana 10](#_Toc353451143)

[2.1.3.3. Izpildes skaitītāja datu analizēšana 10](#_Toc353451144)

[2.1.3.4. Tipiskie eksponētie izpildes skaitītāji 11](#_Toc353451145)

[2.1.4. Datu bāzes piekļuve 11](#_Toc353451146)

[2.1.4.1. Uzticamais lietotājs 11](#_Toc353451147)

[2.1.4.2. Datu bāzes izolācijas līmenis pēc noklusējuma 11](#_Toc353451148)

[2.1.4.3. Glabājamās procedūras 11](#_Toc353451149)

[2.1.4.4. Datu bāzes resursi 12](#_Toc353451150)

[2.1.4.5. Transakcijas 12](#_Toc353451151)

[2.1.4.6. Bloķēšana 12](#_Toc353451152)

[2.2. .NET programmēšanas standarti 13](#_Toc353451153)

[2.2.1. C# projektu izstrādes vadlīnijas 13](#_Toc353451154)

[2.2.1.1. Projektu nosaukumi 13](#_Toc353451155)

[2.2.1.2. Produktu versiju piešķiršana 13](#_Toc353451156)

[2.2.1.3. Projekta informācijas datne 14](#_Toc353451157)

[2.2.1.4. Projekta izmaiņu aprakstīšanas datne 15](#_Toc353451158)

[2.2.2. Vienošanās par nosaukumu piešķiršanu 16](#_Toc353451159)

[2.2.3. Vienošanās par izvietošanas secību 18](#_Toc353451160)

[2.2.4. Remontējamības nosacījumi 18](#_Toc353451161)

[2.2.4.1. C# avota faili (Sourcefiles) 18](#_Toc353451162)

[2.2.4.2. Katalogu izkārtojums (Directory Layout) 19](#_Toc353451163)

[2.2.5. C# izvietojuma nosacījumi 19](#_Toc353451164)

[2.2.6. Komentāru nosacījumi 22](#_Toc353451165)

[2.2.6.1. Blokveida komentāri (Block Comments) 22](#_Toc353451166)

[2.2.6.2. Vienrindas komentāri (Single Line Comments) 22](#_Toc353451167)

[2.2.6.3. Dokumentācijas komentēšana (Documentation Comments) 23](#_Toc353451168)

[2.2.6.4. Koda datnes virsraksts (Copyright) 23](#_Toc353451169)

[2.3. Nosaukumu piešķiršana MS SQL datu bāzes objektiem 24](#_Toc353451170)

[2.4. MS SQL Serveru programmēšanas vadlīnijas 29](#_Toc353451171)

[2.4.1. Programmatūras koda lasāmība un formāts 29](#_Toc353451172)

[2.4.2. Datu tipi 29](#_Toc353451173)

[2.4.3. Saglabātās procedūras 30](#_Toc353451174)

[2.4.4. Veiktspēja 30](#_Toc353451175)

[2.4.5. Dažādi ieteikumi 32](#_Toc353451176)

[3. Oracle datu bāzes un Java programmēšanas vadlīnijas 34](#_Toc353451177)

[3.1. Java projektu izstrādes vadlīnijas 34](#_Toc353451178)

[3.1.1. Programmēšanas vide 34](#_Toc353451179)

[3.1.2. Maven vadlīnijas 34](#_Toc353451180)

[3.1.2.1. Projekta struktūra 34](#_Toc353451181)

[3.1.2.2. Projekta īpašības 34](#_Toc353451182)

[3.1.2.3. Produktu versiju piešķiršana 35](#_Toc353451183)

[3.1.3. Projekta izmaiņu apraksts 35](#_Toc353451184)

[3.1.4. Java koda izstrādes vadlīnijas 35](#_Toc353451185)

[3.1.5. Java EE vadlīnijas 35](#_Toc353451186)

[3.2. Vienošanās par nosaukumu piešķiršanu Oracle datu bāzes objektiem 35](#_Toc353451187)

[3.3. Oracle PL/SQL programmēšanas vadlīnijas 36](#_Toc353451188)

# Attēlu saraksts

[1.attēls. Maven direktoriju struktūra 34](#_Toc353451119)

# Ievads

## Dokumenta nolūks

Standarts „Programmatūras izstrādes vadlīnijas” satur vadlīnijas un noteikumus, kas ir obligāti jāņem vērā, izstrādājot projekta „Integrācijas platformas informācijas sistēmas izstrāde” programmnodrošinājumu, piešķirot nosaukumus.

Šī dokumenta auditoriju veido:

* projekta Pasūtītāja (NVD) darbinieki, kuri ir atbildīgi par projekta nodevumu pieņemšanu un izvērtēšanu;
* projekta Izpildītāja darbinieki, kuri ir atbildīgi par projekta realizāciju: sistēmas projektēšanu, implementēšanu un ieviešanu;
* citu e-veselības aktivitāšu (Elektroniskā veselības kartes informācijas sistēmas, Veselības aprūpes elektronisko nosūtījumu/elektronisko pierakstu informācijas sistēmas, e-receptes informācijas sistēmas) darbinieki, kuri ir atbildīgi par atbilstošās sistēmas projektēšanu, implementēšanu un ieviešanu.

## Termini un pieņemtie apzīmējumi

Apzīmējumu un terminu vārdnīca ir pieejama saistītajā [7] dokumentā.

## Saistība ar citiem dokumentiem

Dokuments ir izstrādāts, balstoties uz šādiem dokumentiem:

1. .NET Programming Standards and Naming Conventions (atrodams: <http://www.irritatedvowel.com/Programming/Standards.aspx>).
2. Database object naming conventions (atrodams: <http://vyaskn.tripod.com/object_naming.htm>).
3. Koplietojuma bibliotēku apraksts.
4. Code Conventions for the Java TM Programming Language (<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>).
5. Naming and Coding Standards for SQL and PL/SQL (<http://www.williamrobertson.net/documents/plsqlcodingstandards.html>).
6. Guidelines, Patterns, and code for end-to-end Java applications

<http://java.sun.com/blueprints/code/namingconventions.html>.

1. Integrācijas platformas informācijas sistēmas izstrāde. Terminu un saīsinājumu indekss. Versija1.01., Laidiens 22.09.2011. (VEC.IP.TSI.1.01).

## Citu nosaukumu piešķiršanas noteikumu lietošana

Atsevišķos gadījumos, veicot programmatūras izstrādi, varētu tikt definētas citas izstrādes rekomendācijas vai nosaukumu piešķiršanas noteikumi. Ja tas netiek darīts, tad jāizmanto vadlīnijas, kas ir apkopotas šajā dokumentā.

## Aprakstu uzmetums

Vadlīnijas, kas saistītas ar dažādiem shēmu izveides aspektiem, tiek iedalītas vairākās grupās:

* MS SQL datu bāzes un .NET programmēšanas vadlīnijas, kas savukārt ietver NET programmēšanas standartus un vienošanās par nosaukumu piešķiršanu MS SQL datu bāzes objektiem.
* Oracle datu bāzes un Java programmēšanas vadlīnijas, kas apraksta Java projektu izstrādes vadlīnijas un Oracle PL/SQL vienošanu par nosaukumu piešķiršanu un izstrādes vadlīnijas.

## Dokumenta pārskats

Dokuments sastāv no trim nodalījumiem:

* Pirmajā nodalījumā – „Ievads” – ir aprakstīts dokumenta nolūks, dota informāciju par terminiem un pieņemtajiem apzīmējumiem, sniegtas norādes par saistītajiem dokumentiem, citu nosaukumu piešķiršanas noteikumu lietošana; dots apraksta uzmetums, kā arī dokumenta pārskats.
* Otrajā nodalījumā – „MS SQL datu bāzes un .NET programmēšanas vadlīnijas” – ir dotas kopējās kodēšanas vadlīnijas .NET programmēšanas standarti, vienošanās par nosaukumu piešķiršanu MS SQL datu bāzes objektiem, kā arī MS SQL Serveru programmēšanas vadlīnijas.
* Trešajā nodalījumā – „Oracle datu bāzes un Java programmēšanas vadlīnijas” – ir dotas *Java* projektu izstrādes vadlīnijas, vienošanās par nosaukumu piešķiršanu Oracle datu bāzes objektiem, kā arī Oracle PL/SQL programmēšanas vadlīnijas.

# MS SQL datu bāzes un.NET programmēšanas vadlīnijas

## Kopējas kodēšanas vadlīnijas

### Kopējās prasības

Šajā nodaļā aprakstītas galvenās projektēšanas un realizācijas metodes, kas tiek izmantotas visām risinājuma komponentēm.

#### Programmēšanas vide

Risinājuma dažu moduļu izstrāde notiek *Visual Studio 2010* vai *Visual Studio 2012* vidē. Risinājuma komponentes izstrādāšanai jāizmanto *Visual C#*, izņemot specifiskus līdzekļus, kuri prasa ciešu integrāciju ar *Windows* zema līmeņa API.

#### Komponentes atkārtota izmantošana

Visur, kur vien iespējams, komponentes jāprojektē, paredzot atkārtotas izmantošanas iespēju; tas nozīmē:

* gadījumi, kad viens un tas pats kods tiek kopēts no vienas komponentes otrā, ir uzskatāmi par sliktu kodēšanas stilu;
* izveidojot protokola specifiskās komponentes, tādas kā: *Web Service*, *ASPX page* vai priekšgala (*front-end*) *WinForms*, kods, kas izpilda reālas biznesa validācijas, pieprasījumus citām servera puses komponentēm un servisiem, ir ievietots atsevišķā biznesa klasē (komponentē), kuru iespējams izmantot atkārtoti (arī šī protokola specifiskās komponentes ietvaros);

izveidojot biznesa kodu, kuram ir pieeja datu bāzei vai citai krātuvei (LDAP), kods, kurš veica pieeju datu bāzei, tiek pārvietots specifiskā datu pieejas klasē (komponentē), ja tas ir iespējams.

Risinājuma komponentes kodam ir jābūt bāzētam uz pašdokumentējoša koda ideju, t.i., kodam ir jābūt viegli lasāmam un jābūt komentētam tikai tad, kad nepieciešams skaidrāk aprakstīt algoritma ideju. Kodā nav jābūt komentāriem, kuri apraksta acīmredzamas lietas, piemēram, vērtību 5 nododam mainīgam *loopCounter*.

Dokumentācijas mērķis *Visual Studio* - atbalstīt koda dokumentāciju, izmantojot iegultos XML tagus, tie tiek izvietoti pēc metodes paraksta vai pēc citiem elementiem. API dokumentācija tiek ģenerēta, izmantojot rīkus, kuriem ir piekļuve informācijai XML komentāros. Lai ieslēgtu dokumentēšanas iespēju *Visual Studio*, jāizvēlas Projects | <Projekta> properties | Build | „XML documentation file”. Ievadot „///” Visual Studio, kods automātiski tiks papildināts ar nepieciešamiem tagiem.

Dokumentācijas valoda ir latviešu valoda. Automātiski ģenerētie komentāri jāatstāj nemainītā veidā, kur tas ir nepieciešams.

### Konfigurācijas vadība

Šajā nodaļā aprakstītas risinājuma komponenšu informācijas konfigurācijas vadības metodes.

#### Konfigurācijas krātuve

Risinājuma komponenšu konfigurācija ir bāzēta uz failiem failu sistēmā, t.i., visai konfigurācijas informācijai jābūt pierakstītai XML failos. Darbvirsmas (*desktop*) aplikācijām tas ir *app.config*, Web aplikācijām tas ir *web.config*. Ja nepieciešams, ir pieņemami konkrēto konfigurāciju saglabāt cita veida failā XML.

#### Konfigurācijas informācijas raksturs

Konfigurācijas informācijai ir jābūt statiskai, tas nozīmē, lai tiktu pielietoti jaunie uzstādījumi, ir nepieciešams palaist komponenti. Tas nodrošinās labāku izpildi, jo trasēšanas informācija var būt kešēta, un arī vienkāršumu, jo dažas konfigurācijas izmaiņas varētu būt kritiskas tās komponentes iekšējam stāvoklim.

Konfigurācijas informācijas statiskais raksturs ietekmē komponenšu pieejamību, kad konfigurāciju nepieciešams izmainīt. Augstas pieejamības komponentēm jābūt vairākos eksemplāros un konfigurāciju maina vienam eksemplāram pēc otra, pārbaudot, ka strādā vismaz viens aktīvais eksemplārs, kamēr tiek mainīta citu eksemplāru konfigurācija.

Konfigurācijas informācijai nav jābūt aizsargātai kādā speciālā veidā, jo:

* no servera puses ir nepieciešama dubulta kontrole servera komponentes vadībai;
* no klienta puses nav jūtīgas konfigurācijas informācijas.

#### Tehnoloģija

Uz .NET bāzētām komponentēm jāizmanto .NET *System.Configuration* konfigurācijas funkcionalitāte, lai nodrošinātu konfigurācijas informācijas piekļuvi.

### Izpildes trasēšana

Dažām risinājuma komponentēm ir jāiztur ļoti liela slodze un liels skaits vienlaicīgu lietotāju, tāpēc ir svarīgi, lai papildus pastāvīgai trasēšanai, kuru izmanto risinājuma atkļūdošanai, tiktu izmantota arī izpildes trasēšana. Parasti izpildes trasēšanai izmanto dokumentā [3] aprakstīto koplietojamo bibliotēku.

Izpildes trasēšana risinājumā jāizpilda, izmantojot:

* *Windows* izpildes skaitītājus, kurus eksponē ar komponentēm, un
* izpildes skaitītājus, kurus eksponē ar *Windows* servisiem un MS standarta programmatūru, piemēram, .NET *Framework,* ASP.NET, SQL *Server*, *Certificate Services*.

Izpildes skaitītājus nepieciešams izmantot servera puses komponentēs, bet specifiskos scenārijos tie var būt izmantoti arī klienta lietojumā.

#### Izpildes skaitītāja izvietojums

Sakarā ar to, ka visām risinājuma komponentēm ir biznesa komponentes daļa, kas atbildīga par komponentes reālu funkcionalitāti, izpildes skaitītāji jāpalielina biznesa komponentes kodā.

Tas nodrošinās, ka, gadījumos, kad viena komponente tiek pakļauta izsaucējiem caur saliktiem kanāliem, vieni un tie paši skaitītāji tiks izmantoti atkārtoti.

#### Izpildes skaitītāju nosaukšana

Izpildes skaitītājus, kurus eksponē ar risinājuma komponentēm, jāpievieno izpildes skaitītāju grupām, kuras nosauktas atbilstoši šablonam "<namespace>.<klases nosaukums>:<metodes nosaukums>".

#### Izpildes skaitītāja datu analizēšana

Trasēšanas informācija, kas eksponēta, izmantojot izpildes skaitītājus, jāanalizē:

* izmantojot *Windows Performance Monitor* rīku reālajā laikā un/vai
* reālā laikā vai vēlāk, izmantojot *Microsoft Operations Management* serveri, kurš savāc visu izpildes informāciju no visiem infrastruktūras serveriem.

#### Tipiskie eksponētie izpildes skaitītāji

Komponentes eksponē izpildes skaitītājus atbilstoši izpildes trasēšanas prasībām, kuras jāidentificē katras komponentes fiziskajā projektējumā. Nepieciešams arī ņemt vērā, kādā veidā komponentes ir izmantotas, to izpildītās kritiskās operācijas, kuras varētu izraisīt izpildes problēmas.

Tipiskākie izpildes skaitītāji, kurus eksponē tipisku komponenšu grupas, ir definēti 1.tabulā.

1.tabula

Izpildes skaitītāji, kurus eksponē tipiskās komponentes

|  |  |
| --- | --- |
| **Komponentes tips** | **Izpildes skaitītājs** |
| Tīkla izsaucamais serviss:* Web Serviss
* *Custom* tīkla serviss
 | Pieprasījumi sekundē |
| Pieprasījumu skaits |
| Vidējais reakcijas laiks |
| Procesora serviss | Apstrādāto objektu kopējais skaits |
| Iepriekšējā norisē apstrādāto objektu skaits |
| Viena objekta vidējais apstrādes laiks  |

### Datu bāzes piekļuve

Nodaļā aprakstītas risinājuma datu bāzes informācijas piekļuves koda realizācijas metodes.

#### Uzticamais lietotājs

Uzticama datu bāzes lietotāja metode tiek izmantota, lai piekļūtu pie informācijas datu bāzē, t.i., neskatoties uz gala lietotāja izpildāmo operāciju, lietojuma servera kods, izpildot operāciju, tiek autentificēts datu bāzē kā tā paša servisa lietotājs.

*Windows Integrated Security* izmanto, lai apliecinātu servisu datu bāzē. Katram servisa kontam ir tiesības izpildīt glabājamu procedūru komplektu, kurš ir nepieciešams, lai izpildītu darbības, par kurām serviss ir atbildīgs.

#### Datu bāzes izolācijas līmenis pēc noklusējuma

Pēc noklusējuma datu bāzes izolācijas līmenis ir *Snapshot* izolācija. Tas nodrošina labāku caurlaidspēju un izpildi lasīšanas operācijām.

Rakstīšanas operācijas saskarne ir realizēta ar manuāli specificētu izolācijas līmeni, definējot transakcijas izolācijas līmeni lietojuma kodā vai glabājamā procedūrā.

#### Glabājamās procedūras

Glabājamās procedūras jāizmanto piekļuvei visām datu bāzēm, t.sk., arī lasīšanas operācijām. Specifiskos scenārijos, kur tiešas SQL SELECT izteiksmes varētu būt efektīvākas, ir atļauts izmantot SQL izteiksmes ar arhitektūras komandas apstiprinājumu.

#### Datu bāzes resursi

Piekļūstot datu bāzei, *acquiring late* un *releasing soon* princips jāizmanto cauri datu bāzes kodam. Tas nozīmē, ka sakaram ar datu bāzi ir jābūt atvērtam tikai pirms glabājamās procedūras izpildes un jābūt atbrīvotam, kad izpilde ir pabeigta.

#### Transakcijas

Ja vairākām datu bāzes operācijām, kuras tiek izsauktas no lietojuma servera komponentēm, jābūt vienā datu bāzes transakcijā, tad datu bāzes operācijas ir apvienotas, izmantojot *System.Transactions.TransactionScope* objektu. Piemēram, bāziskas datu izgūšanas operācijas, kas izpilda tikai vienu glabājamo procedūru, nav apvienotas, izmantojot *TransactionScope*. Šīm vienkāršajām lasīšanas operācijām nav jādarbojas ar SQL transakciju vai transakcijas kopas objektiem.

Kad sarežģītākas operācijas tiek izpildītas, izmantojot vienkāršas lasīšanas operācijas (vai vairākas lasīšanas operācijas), tad vienkāršo operāciju izsaukšana jāizpilda *TransactionScope* definētā izsaucējā. Tas ļauj atkārtoti izmantot vienkāršās operācijas sarežģītākos scenārijos un arī mainīt sēriju veidošanas metodi, lai nodrošinātu nepieciešamo resursu aizsardzību.

#### Bloķēšana

Datu bāzes pieprasīto datu bloķēšanu iespējams izpildīt divos veidos:

* Metode pēc noklusējuma – nav bloķēšanas;
* Speciāli gadījumi:
* optimistiskā bloķēšana – entītijām, kuras bieži vienlaicīgi atjaunina daudzi cilvēki;
* pesimistiskā bloķēšana (*check-out, check-in*) – objektiem, kurus no objektu apvienojuma apstrādā daudzi cilvēki, lai samazinātu operāciju sadursmes iespēju uz vienu ierakstu.

Bloķēšana pēc noklusējuma nav pieņemama risinājumam, jo:

* informācijas atjauninājumi sistēmā ir reti – sistēma ir *insert* un *select* intensīva;
* objekti datu bāzē parasti tiek atjaunināti pēc klienta pieprasījuma no klientu servisa punkta vai pēc zvana no palīdzības dienesta (tas nodrošina, ka tikai viens operators strādās ar konkrētas personas datiem);
* datu bāzes objekti, kuri tiek atjaunināti, pārsvarā ir pamata entītijas statusi – tāpēc iespējamu statusu izmaiņu validācija pasargā no nekorektām statusa izmaiņām un zināmā mērā nodrošina optimistisko sinhronizāciju.

Optimistiskā bloķēšana jārealizē, izmantojot laikspiedolu katrā ierakstā un salīdzinot laikspiedolu ar pašreizējo vērtību datu bāzē, iesniedzot izmaiņas atpakaļ datu bāzē.

Pesimistiskā bloķēšana jārealizē, izmantojot laika apvienojuma laikspiedolu un tā lietotāja informāciju, kurš bloķēja katru ierakstu. Entītijām, kuras var būt bloķētas pesimistiski, risinājums piedāvā funkcionalitāti, lai manuāli vadītu sinhronizāciju (*cancel check-out*) lietotājiem ar iepriekš definētām lomām. Speciālu bloķēšanas gadījumu identificēšana tiek atstāta sistēmas loģikas projektējumam.

## .NET programmēšanas standarti

### C# projektu izstrādes vadlīnijas

#### Projektu nosaukumi

Izstrādes gaitā tiek izstrādāts produkts. Produkts var sastāvēt no daudziem projektiem. Projektiem rekomendējam izmantot sekojošus nosaukumus:

* **Product** – produkta nosaukums vai produkta sagrupēta daļa;
* **Product.<ProjectType>** - produkta projekta nosaukumi, kur <ProjectType>:
* WebService – Web servisa projekts;
* WebSite – Web lietojuma projekts;
* DataBase – datubāzes projekts;
* BusinessLogic – biznesa loģikas projekts;
* utt.
* **Product.<ProjectType>.UnitTests** – produkta projekta testa projekts;
* **Product.<ProjectType>.Deployment** – projekta publikācijas projekts;
* **Product.<ProjectType>.Setup** – produkta laidiena projekts;
* **Product.<ProjectType>.Update** – produkta laidiena izmaiņas projekts.

#### Produktu versiju piešķiršana

Produktam un tā projektiem tiek uzstādītas attiecīgas versijas.

##### Produkta versija

Produkta versijas formāts „M.m”, kur

* M- pamatversijas numurs;
* M - otrais versijas numurs.

Pirmoreiz izstrādātam produktam versija ir 1.0

##### Projekta versija

Projekta versijas formāts „M.m.b.r”, kur

* M -pamatversijas numurs;
* m - otrais versijas numurs;
* b - kompilācijas numurs;
* r - revīzijas numurs.

Pirmoreiz projekta versija ir 1.0.build.0, kur *build* ir dienu skaits no 01/01/2000.

##### Laidiena versija

Projekta versija formāts „M.m.b.r” vai „M.m.r”, kur

* M -pamatversijas numurs;
* m - otrais versijas numurs;
* b - kompilācijas numurs;
* r - revīzijas numurs.

Pirmoreiz izveidotam laidienam versija ir 1.0.0 vai 1.0.b.0 atkarībā kāds laidiens tiek izveidots.

##### Datubāzes versija

Datubāzes versijas formāts „M.m.rrrr”, kur

* M -pamatversijas numurs;
* m - otrais versijas numurs;
* rrrr - revīzijas numurs no četriem cipariem.

Pirmoreiz izstrādātai datubāzei versija ir 1.0.0000

##### Versiju numuru izmaiņas nosacījumi

Komponentes revīzijas numuru izmaina, ja tiek veiktas izmaiņas. Izmaiņas var būt saistītas ar kļūdu novēršanu vai koda pārstrādāšanu, kas nepasliktina tā funkcionalitāti.

Laidiena revīzijas numurs tiek mainīts pēc jaunā laidiena sagatavošanas. Parasti produkta laidiena versija tiek mainīta pēc produkta galvenās komponentes izmaiņas.

Otrais versijas numurs tiek mainīts, ja tiek veiktas izmaiņas, kas noved pie funkcionalitātes izmaiņām vai jaunās funkcionalitātes pievienošana.

Pamatversijas numurs tiek mainīts, ja tek mainīta produkta struktūra un izveides principi.

#### Projekta informācijas datne

Projektam jāpievieno informācijas datne. Tās piemērs:

// ----------------------------------------------------------------------------

// <copyright file="AssemblyInfo.cs" company="ABC software">

// Copyright © ABC SOFTWARE. All rights reserved.

// The source code or its parts to use, reproduce, transfer, copy or

// keep in an electronic form only from written agreement ABC SOFTWARE.

// </copyright>

// ----------------------------------------------------------------------------

using System;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following

// set of attributes. Change these attribute values to modify the information

// associated with an assembly.

[assembly: AssemblyTitle("Product.BusinessLogic")]

[assembly: AssemblyDescription("Product business logic.")]

[assembly: AssemblyCompany("ABC software")]

[assembly: AssemblyProduct("Product")]

[assembly: AssemblyCopyright("Copyright \x00a9 ABC software 2007-2008")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("")]

[assembly: AssemblyConfiguration("")]

// Setting ComVisible to false makes the types in this assembly not visible

// to COM components. If you need to access a type in this assembly from

// COM, set the ComVisible attribute to true on that type.

[assembly: ComVisible(false)]

[assembly: CLSCompliant(true)]

// The following GUID is for the ID of the typelib if this project is exposed to COM

[assembly: Guid("6c05645c-69c5-46d6-a030-f2029cbc5abc")]

// Version information for an assembly consists of the following four values:

//

// Major Version

// Minor Version

// Build Number

// Revision

[assembly: AssemblyVersion("1.0")]

[assembly: AssemblyFileVersion("1.0.0.2")]

Projekta informācijas datnes atribūtu saraksts:

* **AssemblyVersion** - produkta versija;
* **AssemblyFileVersion** - projekta versija;
* **AssemblyCulture** – *asemples* kultūra;
* **AssemblyTitle** – projekta nosaukums;
* **AssemblyDescription** – projekta apraksts;
* **AssemblyConfiguration** – kompilācijas konfigurācija;
* **AssemblyCompany** – kompānijas nosaukums, kurš izveidoja projektu;
* **AssemblyProduct** – produkta nosaukums;
* **AssemblyCopyright** – autortiesības;
* **AssemblyTrademark** – preču zīme;
* **AssemblyInformationVersion** – *asembles* informatīvā versija (parasti sakrīt ar *AssemblyFileVersion*);
* **AssemblySateliteVersion** – parasti sakrīt ar produkta versiju; tiek izmantota daudzu valodu resursu izmantošanas gadījumā.

#### Projekta izmaiņu aprakstīšanas datne

Katram projektam jābūt pievienotai izmaiņu aprrakstīšanas datnei *Histroy.txt*. Datne tiek veidota tekta formātā. Vēstures datnes formāts:

Date: dd/mm/yyyy

Assembly: assemblyname

[File: filename]

BUGFIX | NEW | CHANGE : comment

Release Vx.x.x.x at dd/mm/yyyy

, kur

* x.x.x.x – relīza projekta versija;
* *dd/mm/yyyy* – projekta izmaiņas vai relīza izlaiduma datums;
* *assemblyname* – *assemblies* nosaukums (obligātas, ja versija maiņas produkta grupai);
* filename – datnes nosaukums, kurā tika veiktas izmaiņas (neobligāts);
* BUGFIX – kļūda;
* NEW – pievienojumi;
* CHANGE – izmaiņas;
* *comment* – darbību apraksts.

Vēstures datnes piemērs:

Date: 5/02/2007

Assembly: Product.BusinessLogic.dll

NEW: Pieveinota jauna funkcija

CHANGE: Izmainīts vecās funkijas fromāts.

NEW: XMLSchemaTransformation klasej pievienoja īpašība XSLTFile

Release 1.0.2344.0 no 06/02/2007

### Vienošanās par nosaukumu piešķiršanu

Dokumentā ir aprakstīti projektā pieņemtie nosaukumu piešķiršanas standarti, kuri attiecas uz programmām, kas rakstītas, izmantojot J#, C# un VB.NET (papildinformācijai skatīt MSDN palīgdokumentāciju). Pastāv *Microsoft* rekomendācijas publiskiem un privātiem izstrādātājiem, tomēr individuālās vienošanās par nosaukumu piešķiršanu kārtību kompānijas iekšienē var atšķirties. Paredzams, ka 2.tabulā ietverto vadlīniju saraksts drīz kļūs par nozares *de-facto* standartu. Šos nosaukumu piešķiršanas standartus nepieciešams pielietot visiem .NET izstrādājumiem, t.sk., ASP.NET Web lietojumos un .NET *Windows Forms* lietojumos. Visi nosaukumi jāveido angļu valodā.

2. tabula

C# un VB.NET nosaukumu vadlīniju saraksts

| Tips | Standarts / Noteikums | Piemērs |
| --- | --- | --- |
| Vārdtelpas (*namespace*) | *Pascal* reģistrā bez pasvītrojuma. Izmantojam Kompānijas Nosaukums. Tehnoloģijas Nosaukums par sakni. Gadījumā, ja nav kompānijas, izmantojam domēna nosaukumu. Visiem nosaukumiem jābūt *pascal* reģistrā (izmantojam Xml, nevis XML). | Application.SistemJournalApplication.Controllers |
| Asamblejas | Ja asambleja satur tikai vienu vārdtelpu, tad asambleja ir jānosauc tāpat kā vārdtelpa.  | Application.SistemJournal.dll |
| Klases un struktūras | *Pascal* reģistrā bez pasvītrojuma nosaukuma sākumā nedrīkst likt "C" vai "cls". Klases nosaukums var sākties ar "I" tikai tad, ja nākamais burts ir mazais, pretējā gadījumā tas izskatīsies kā saskarne. Klases nosaukumam nav jābūt līdzīgam vārdtelpai, kurā tā uzturas. Visiem saīsinājumiem, kuri sastāv no trim vai vairāk burtiem, ir jābūt *Pascal* reģistrā, ar mazajiem burtiem. Jācenšas izvairīties no saīsinājumu lietošanas un vienmēr jācenšas izmantot lietvārdus.  | WidgetInstanceManagerXmlDocumentMainFormDocumentFormHeaderControlCustomerListDataSet (typed dataset)  |
| Kolekciju klases | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Collection*. | WidgetCollection |
| Klases *Delegate*  | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Delegate*. | WidgetCallbackDelegate |
| *Exception Classes* | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Exception*. | InvalidTransactionException |
| *Attribute Classes* | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Attribute*. | WebServiceAttribute |
| *Interfaces* | Jāseko klases nosaukumu piešķiršanas noteikumiem, pirms nosaukuma jāpievieno "I", pēc "I" nākamais burts ir lielais.  | IWidget |
| Uzskaitījums (*Enumerations*) | Jāseko klases nosaukumu piešķiršanas noteikumiem. Uzskaitījuma beigās nepievieno "Enum". Ja uzskaitījums reprezentē bitu karodziņu grupu, tad nosaukums jāpabeidz daudzskaitlī.  | SearchOptions (bitwise flags)AcceptRejectRule (normal enum) |
| Funkcijas un apakšfunkcijas(*Functions* *and* *Subs*) | *Pascal* reģistrs bez pasvītrojuma, izņemot notikumu apdarinātājus (*handlers*).Jācenšas izvairīties no saīsinājumu lietošanas. Funkcijām un apakšfunkcijām ir jāatšķiras ne tikai ar burtu reģistru, lai tās varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasic .NET*. | VB: Public Sub DoSomething(...)C#: public void DoSomething(...)  |
| Īpašības un Publiskie \* locekļu mainīgie(*Properties and Public \* Member Variables*) | *Pascal reģistrs* bez pasvītrojuma. Jācenšas izvairīties no saīsinājumu lietošanas. Locekļiem ir jāatšķiras ne tikai ar burtu reģistru, lai tos varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasic .NET* .  | VB: Public Property RecordId As IntegerC#: public int RecordId |
| Parametri(*Parameters*) | *Camel* reģistrs. Jācenšas izvairīties no saīsinājumu lietošanas. Parametriem ir jāatšķiras ne tikai ar burtu reģistru, tos varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasic .NET* . | VB: ByRef recordId As IntegerC#: ref int recordId |
| Procedūras līmeņa mainīgie (*Procedure-Level Variables*) | *Camel* reģistrs. | VB: Dim recordId As IntegerC#: int recordId; |
| Klases līmeņa privātie un aizsargātie mainīgie(*Class-Level Private and Protected Variables*) | *Camel* reģistrs ar pirmo simbolu pasvītrojumu. VB.NET vienmēr jānorāda "Aizsargāts" vai "Privāts", nevar izmantot "Dim". "m\_" izmantošana nav ieteicama, tāpat arī tādu mainīgo nosaukumu izmantošana, kuri atšķiras no īpašības vienīgi ar reģistru, īpaši ar aizsargātiem mainīgiem, jo tādējādi tiks apgrūtināts darbs, ja programmēšanu realizēs VB.NET, jo locekļus būs nepieciešams nosaukt citādi kā accessor/mutator īpašības.*Microsoft* rekomendē izmantot m\_ (un arī \_), tomēr viņi savos kodos izmanto abus variantus. Ir pieļaujams "m" likt arī sākumā. Protams, kamēr programmē, galvenokārt, ar C#, tiem varētu būt privāti locekļi, kuri atšķiras no īpašībām tikai ar reģistru.Izstrādātājiem, kas programmē ar VB, ir jādara kaut kas vēl. Lai nemēģinātu un nenolīdzinātu katrai valodai speciālus reģistrus, tiek rekomendēts pievienot priedēkli – pasvītrojuma simbolu visās valodās, kuras to atbalsta. Ja nepieciešams, lai klase būtu pilnīgi pakļāvīga CLS, varētu atmest priedēkli visiem C# aizsargātiem locekļu mainīgiem. | VB: Private \_recordId As IntegerC#: private int \_recordId; |
| Konstantes(*Constants*) | *Pascal* reģistrsNevar izmantot AUGŠĒJĀ\_REĢISTRA\_BURTUS. | SomeClass.SomePublicConstant |

### Vienošanās par izvietošanas secību

*Using* direktīvam jābūt ievietotām vārdutelpās. Koda datnes elementiem jābūt sakārtotiem sekojošā secībā:

* *Extern Alias Directives*;
* *Using Directives*;
* Vārdutelpas (*Namespaces*);
* Delegāti (*Delegates*);
* Uzskaitījumi (*Enums*);
* Interfeisi (*Interfaces*);
* Struktūras (*Structs*);
* Klases (*Classes*).

Klasēs, struktūrās un interfeisos elementiem jābūt sakārtotiem sekojošā secībā:

* Mainīgie (*Fields*);
* Konstruktori (*Constructors*);
* Finalizers (*Destructors*);
* Delegāti (*Delegates*);
* Notikumi (*Events*);
* Uzskaitījumi (*Enums*);
* Interfeisi (*Interfaces*);
* Īpašības (*Properties*);
* Indeksu īpašības (*Indexers*);
* Funkcijas (*Methods*);
* Struktūras (*Structs*);
* Klases (Cla*s*ses).

Elementiem jābūt sakārtotiem pēc pieejas līmeņa sekojošā secībā:

* Publiskie elementi (*public*);
* Iekšējie elementi (internal);
* Aizsargātie iekšējie elementi (*protected* *internal*);
* Aizsargātie elementi (*protected*);
* Privātie elementi (*private*).

Konstantiem elementiem jābūt iepriekš nekonstantiem elementiem. Statiskiem elementiem jābūt iepriekš parastiem elementiem.

### Remontējamības nosacījumi

#### C# avota faili (Sourcefiles)

Klases/failus veido īsus, nepārsniedzot 2000 rindiņu, kodu sadala, struktūras veido skaidrākas. Katru klasi ievieto atsevišķā failā un failu nosauc līdzīgi klasei (izmanto paplašinājumu .cs). Šis nosacījums ļoti atvieglo darbu.

#### Katalogu izkārtojums (Directory Layout)

Katrai vārdtelpai izveido katalogu (*MyProject*.*TestSuite*.*TestTier* lieto *MyProject*/

*TestSuite*/*TestTier* kā ceļu*,* vārdtelpas nosaukumu nelieto kopā ar punktiem(".").). Tas atvieglo vārdtelpu kartēšanu atbilstoši kataloga izkārtojumam.

### C# izvietojuma nosacījumi

Instrukcijām jābūt ievietotām starp atvērtām un aizvērtām iekavām. Ieteicamāk:

 if (true)

 {

 return this.value;

 }

Nevēlams stils:

 if (true)

 return this.value;

Instrukcijai ar atvērtām un aizvērtām iekavām jābūt atsevišķā rindā. Ieteicamāk:

 public object Method()

 {

 lock (this)

 {

 return this.value;

 }

 }

...

public object Method()

 {

 return null;

 }

Nevēlams stils:

 public object Method()

 {

 lock (this) { return this.value; }

 }

...

 public object Method() { return null; }

Elementiem jābūt atdalītiem ar tukšo rindu. Ieteicamāk:

 public void Method1()

 {

 }

 public bool Property

 {

 get { return true; }

 }

 /// <summary>

 /// Gets a value indicating whether the control is enabled.

 /// </summary>

 public bool Enabled

 {

 get { return this.enabled; }

 }

Nevēlams stils:

 public void Method1()

 {

 }

 public bool Property

 {

 get { return true; }

 }

 /// <summary>

 /// Gets a value indicating whether the control is enabled.

 /// </summary>

 public bool Enabled

 {

 get { return this.enabled; }

 }

Īpašībām un notikumiem jābūt vai vienā rindā vai atsevišķās rindās. Ieteicamāk:

 public bool Enabled

 {

 get { return this.enabled; }

 set { this.enabled = value; }

 }

 public bool Enabled

 {

 get

 {

 return this.enabled;

 }

 set

 {

 this.enabled = value;

 }

 }

Nevēlams stils:

public bool Enabled

 {

 get { return this.enabled; }

 set

 {

 this.enabled = value;

 }

 }

Pirms vienas rindas komentāra jaievieto tukšā rinda. Ieteicamāk:

 public bool Enabled

 {

 get

 {

 Console.WriteLine("Getting the enabled flag.");

 // Return the value of the 'enabled' field.

 return this.enabled;

 }

 }

Nevēlams stils:

 public bool Enabled

 {

 get

 {

 Console.WriteLine("Getting the enabled flag.");

 // Return the value of the 'enabled' field.

 return this.enabled;

 }

 }

Nelietot liekas tukšās rindas:

* nelietot tukšo rindu pirms atvērtas iekavas;
* nelietot tukšo rindu pēc atvērtas iekavas;
* nelietot tukšo rindu pēc elementa dokumentācijas;
* nelietot vairāk kā vienu tukšu rindu starp instrukcijām;
* nelietot tukšo rindu pirms aizvērtas iekavas;
* nelietot tukšo rindu pēc vienas rindas komentāra.

 Ieteicamāk:

 /// <summary>

 /// Gets a value indicating whether the control is enabled.

 /// </summary>

 public bool Enabled

 {

 get

 {

 Console.WriteLine("Getting the enabled flag.");

 // Return the value of the 'enabled' field.

 return this.enabled;

 }

 }

Nevēlams stils:

 /// <summary>

 /// Gets a value indicating whether the control is enabled.

 /// </summary>

 public bool Enabled

 {

 get

 {

 Console.WriteLine("Getting the enabled flag.");

 // Return the value of the 'enabled' field.

 return this.enabled;

 }

 }

### Komentāru nosacījumi

#### Blokveida komentāri (*Block* *Comments*)

No blokveida komentāriem parasti vajadzētu izvairīties. Aprakstiem rekomendē lietot /// komentēšanas stilu, lai iegūtu C ♯ standarta aprakstus. Ja vēlas izmantot blokveida komentārus, jālieto šāds stils:

/\* Line 1

 \* Line 2

 \* Line 3

 \*/

Tas lasītājam (cilvēkam) ļaus vizuāli bloku atšķirt no koda. Pastāv alternatīva iespēja lietot novecojušo C stilu vienrindas komentāriem, tomēr tas nebūtu ieteicami. Ja izvēlas lietot šo stilu, tad pēc komentāra jāseko rindas aplauzumam, pretējā gadījumā - būs grūti ieraudzīt kodu, kuram priekšā tajā pašā rindā atrodas komentāri:

/\* blah blah blah \*/

Blokveida komentāri varētu noderēt reti, piemēram, saistībā ar *TechNote „The fine Art of Commenting”*. Vispārīgā gadījumā blokveida komentārus izmanto, lai komentētu lielas koda sekcijas.

#### Vienrindas komentāri (*Single* *Line* *Comments*)

Tiek rekomendēts lietot // komentēšanas stilu, lai pievienotu komentāru kodam (šim nolūkam *SharpDevelop* izmanto taustiņu kombināciju - Alt+/). To varētu izmantot arī, lai komentētu koda sekcijas.

Vienrindas komentāros nepieciešams ievērot atkāpes līmeni, ja šie komentāri tiek lietoti koda dokumentēšanai.

Komentētu kodu būtu vēlams komentēt pirmajā rindā, lai palielinātu komentēta koda redzamību.

Pārlapošanas (*thumb*) likums nosaka, ka vispārīgā gadījumā komentāra garums nedrīkst daudz pārsniegt koda garumu, jo tā ir pārāk komplicētu, potenciāli kļūdainu (*potentially buggy*) kodu pazīme.

#### Dokumentācijas komentēšana (Documentation Comments)

.Net ietvaros *Microsoft* ir ieviesis uz XML komentāriem bāzētu dokumentācijas ģenerēšanas sistēmu. Formāli šos komentārus var uzskatīt par vienrindas C♯ komentāriem, kas satur XML tagus. Tie vienrindas komentāru veidošanā seko šādam paraugam:

/// <summary>

/// This class...

/// </summary>

Daudzrindu XML komentārus veido šādi:

/// <exception cref=”BogusException”>

/// This exception gets thrown as soon as a

/// Bogus flag gets set.

/// </exception>

Ikvienai rindai jāsākas ar 3 slīpsvītrām, lai tās tiktu akceptētas kā XML komentāra rindas. Izšķir divu kategoriju tagus:

* Dokumentācijas vienumi (*Documentation items*);
* Formatēšanai/Atsauču veidošanai (*Formatting/Referencing*).

Pirmajā kategorijā ietilpst tādi tagi, kā, piemēram, <summary>, <param> vai <exception>. Tie reprezentē vienumus, kuri pārstāv programmas saskarnes elementus, kurus nepieciešams dokumentēt, lai programma būtu derīga citiem programmētājiem. Parasti šiem tagiem ir tādi atribūti kā nosaukums vai *cref*, kā tas redzams iepriekš minētajā daudzrindu piemērā. Šos atribūtus pārbauda kompilators, līdz ar to, tiem ir jābūt ticamiem.

Otra kategorija pārvalda dokumentācijas izkārtojumu, lietojot tādus tagus, kā: <code>, <list> vai <para>.

Pēc tam dokumentāciju ir iespējams ģenerēt, izmantojot izvēlnes #develop „build” vienumu „dokumentēšana”. Dokumentācija tiek ģenerēta HTML Help formātā.

Detalizētāks XML komentāru skaidrojums pieejams *Microsoft .net framework SDK* dokumentācijā. Informācija par ieteicamo komentēšanas praksi un turpmākajiem jautājumiem saistībā ar komentēšanu atrodama *TechNote „The fine Art of Commenting”*.

Jābūt nokomentētiem sekojošiem elementiem:

* klases publiskiem locekļiem un tā parametriem un izejas datiem;
* uzskaitījuma locekļiem;
* *generic* tipa parametriem.

#### Koda datnes virsraksts (Copyright)

Katrā kodu datnē jābūt ievietotam virsrakstam. Virsrakstam jābūt pirmajā datnes rindā un tam jābūt komentētam xml:

//-----------------------------------------------------------------------

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

//-----------------------------------------------------------------------

Līnijas nav obligātas, un visrsraksts var izskatīties sekojoši:

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

Ir iespēja pievienot papildus xml tagus:

//-----------------------------------------------------------------------

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

// <author>John Doe</author>

//-----------------------------------------------------------------------

## Nosaukumu piešķiršana MS SQL datu bāzes objektiem

Pašlaik pastāv ļoti daudz dažādu vienošanos par nosaukumu piešķiršanu datu bāzes objektiem. Neviena no tām nav kļūdaina, tā vairāk ir izstrādātāju personīga izvēle. Tomēr uzņēmumā viens darbinieks (vai darbinieku grupa) nosaka datu bāzes nosaukumu piešķiršanas noteikumus, standartizē tos, un citiem tie jāievēro. E-veselības IP projekta pieņemtā nosaukumu piešķiršanas vienošanās MS SQL datu bāzes objektiem ir aprakstīta 3.tabulā.

3.tabula

MS SQL nosaukumu vadlīniju saraksts

| Tips | Standarts / Noteikums/ Piemērs | Nosaukumu piemērs |
| --- | --- | --- |
| Tabulas | Tabulas reprezentē entītiju instances. Piemēram, informācija par visiem klientiem tiek glabāta vienā tabulā. Tātad 'klients' (*customer*) šajā gadījumā būs entītija un visas rindas klientu tabulā reprezentē entītijas 'klients' (*customer*) instances. Kāpēc nevar nosaukt tabulu pēc tās entītijas nosaukuma, '*Customer*'? Ņemot vērā, ka tabula satur daudzas klienta instances, tās nosaukumam ir jābūt daudzskaitlī - '*Customers*'.Šī metode ir piemērotāka, nekā nosaukt tabulas, piemēram, *tblCustomers*. Skatoties uz vaicājumiem, ir skaidrs, ka atsevišķs nosaukums atteicas uz tabulu, jo tabulas nosaukuma priekšā vaicājumos vienmēr ir noteikums FROM (SELECT izteikuma daļa). Ja datu bāzē ir dažādas loģiskas funkcijas un tabulas ir nepieciešams sagrupēt atbilstoši loģiskajām grupām, kurām tās pieder, tad var izmantot divu vai trīs burtu priedēkli tabulas nosaukumā, kurš identificēs tabulas grupu. Piemēram, datu bāzē ir tabulas, kuras glabā informāciju par Cilvēku resursu (Human resource) nodaļu un arī par Realizācijas (Sales) nodaļu. Visas tabulas, kuras attiecas uz Realizācijas nodaļu, var nosaukt, piemēram, tā:SL\_NewLeadsSL\_TerritoriesSL\_TerritoriesManagersVisas tabulas, kuras attiecas uz Cilvēku resursu nodaļu, var nosaukt šādi:HR\_CandidatesHR\_PremierInstitutesHR\_InterviewSchedulesTāda nosaukumu piešķiršanas metode garantē, ka visas saistītās tabulas ir sagrupētas kopā un ka visas tabulas tiek kārtotas alfabētiskā secībā. Tomēr, ja datu bāzē ir tikai viena tabulu loģiskā grupa, tad nav vajadzības izmantot šo metodi.Jāņem vērā, ka dažreiz tabulas vertikāle tiek sadalīta divās vai vairāk tabulās, taču šīs daļas efektīvi reprezentē vienu un to pašu entītiju. Tādā gadījumā, entītijas nosaukumam jāpievieno vārds, kas vislabāk identificē konkrēto sadaļu. | CustomersOrdersErrorMessagesSL\_NewLeadsSL\_TerritoriesSL\_TerritoriesManagersHR\_CandidatesHR\_PremierInstitutesHR\_InterviewSchedules |
| Skati (*view)* | Jebkurai lietojumprogrammai, kura piekļūst skatam, skats ir tas pats, kas tabula. Tātad, nosaukumu veidošanas noteikumus, kas attiecas uz tabulām, var izmantot arī skatiem, tomēr ne visos gadījumos.Izņēmumi:1. Skati ne vienmēr reprezentē vienu entītiju. Skats var būt divu tabulu kombinācija, kas bāzēta uz JOIN nosacījuma, tādējādi, efektīvi reprezentējot uzreiz divas entītijas. Tādā gadījumā var kombinēt abu tabulu nosaukumus. Piemēram, ja skats apvieno tabulas '*Customers*' un ‘*Addresses*’, tad nosaukums ir 'CustomersAddresses'. Šo metodi var pielietot palīgtabulām, kuras tiek izmantotas, lai realizētu saiti daudzi-pret-daudziem starp tabulām.
2. Skati var apkopot esošo bāzes tabulu datus atskaites formā. Šīm atskaitēm iespējams izmantot, piemēram, šādus nosaukumus:

'Product\_Sales\_for\_1997’'Summary\_of\_Sales\_by\_Year'Atstarpes vietā objektu nosaukumos jāizmanto pasvītra( \_ ). | CustomersAddressesProduct\_Sales\_for\_1997 Summary\_of\_Sales\_by\_Year |
| Glabājamās procedūras | Glabājamās procedūras ir orientētas uz darbībām. Tātad to nosaukumiem jāattēlo darbs, kuru procedūra izpilda. Procedūras nosaukumā jāizmanto darbības vārds. Piemēram, procedūru, kura atgriež informāciju par klientu, ja tai ir nodots klienta identifikācijas numurs, vēlams nosaukt '*GetCustomerDetails*'.Līdzīgi, procedūra, kura realizē informācijas par klientu ievadi, būtu jānosauc 'I*nsertCustomerInfo*'. Vēl daži procedūru nosaukumu piemēri: ‘WriteAuditRecord', 'ArchiveTransactions', 'AuthorizeUser'Kā jau tika minēts tabulu gadījumā, iespējams izmantot priedēkli, lai glabājamās procedūras sagrupētu arī atkarībā no tā, kurai loģiskai grupai procedūra pieder. Piemēram, visām procedūrām, kuras ir saistītas ar pasūtījumu apstrādi (*'Order processing'*) varētu būt priedēklis ORD\_ (ar lieliem burtiem):ORD\_InsertOrderORD\_InsertOrderDetailsORD\_ValidateOrderIzmantojot Microsoft SQL Server, procedūrai nekad nevajag pievienot priedēkli 'sp\_', ja vien procedūra netiek glabāta galvenajā datu bāzē. Ja procedūras nosaukumam būs pievienots priedēklis 'sp\_', SQL Server vienmēr meklēs šo procedūru galvenajā datu bāzē *(Master database*) un tikai pēc galvenās datu bāzes pārbaudes (ja tur nebūs atrasts) meklēs dotajā datu bāzē.Nav ieteicams glabājamas procedūras nosaukumam pievienot priedēkli, piemēram, 'sproc\_', lai tikai parādītu, ka objekts ir glabājama procedūra. Jebkurš datu bāzes attīstītājs vai administrators var identificēt glabājamas procedūras, jo tām priekšā vienmēr ir atslēgvārds EXEC vai EXECUTE. | GetCustomerDetailsWriteAuditRecordArchiveTransactions AuthorizeUser ORD\_InsertOrderORD\_InsertOrderDetailsORD\_ValidateOrder |
| Lietotāja definētas funkcijas | Microsoft SQL Serverī lietotāja definētas funkcijas (turpmāk tekstā LDFas) ir gandrīz līdzīgas glabājamām procedūrām, izņemot faktu, ka LDFas var būt izmantotas SELECT izteikumos. Citādi glabājamās procedūras un LDFas ir līdzīgas. Tātad, nosaukumu piešķiršanas metodes, kuras tika aprakstītas glabājamu procedūru gadījumā, var pielietot arī LDFām. Iespējams izmantot priedēkli (ar maziem burtiem), lai sagrupētu LDFas. Piemēram, visas LDFas manipulācijām ar virknēm var nosaukt šādā veidā:str\_MakeProperCasestr\_ParseString | ProcessDatestr\_MakeProperCasestr\_ParseString  |
| Trigeri | Neskatoties uz to, ka trigeri ir speciāls glabājamās procedūras veids, nav jēgas izmantot tādu pašu nosaukumu piešķiršanas metodi, kā glabājamām procedūrām.Piešķirot nosaukumu trigeriem, izšķir divus variantus:1. Trigeri vienmēr ir atkarīgi no bāzes tabulas, un nevar eksistēt paši par sevi. Tātad, labāk savienot bāzes tabulas nosaukumu ar trigera nosaukumu.
2. Trigeri parasti ir asociēti ar vienu vai vairākām šādām operācijām: Insert, Update, Delete. Tātad trigera nosaukumam jāatspoguļo to būtība.

Daži trigeru nosaukumu piemēri:titles\_instrgtitles\_updtrgtitles\_deltrgMicrosoft SQL Server atļauj vairāk nekā vienu trigeri katrai darbībai ar tabulu. Tātad, ja tas ir nepieciešams, varētu būt 2 trigeri ar INSERT, 3 ar UPDATE un 4 ar DELETE vienai tabulai. SQL Serverī nevar kontrolēt trigeru palaišanas kārtību, tomēr palaišanas kārtības kontrole ir iespējama, sākot ar SQL Serverī 2000 versiju. Tātad, ja tabulai 'Titles' ir 3 trigeri ar INSERT, nepieciešams izmantot šādu nosaukumu piešķiršanas metodi, lai tos atšķirtu:titles\_ValidateData\_instrgtitles\_MakeAuditEntries\_instrgJa viens trigeris ir saistīts vairāk kā ar vienu darbību (viens trigeris INSERT un UPDATE vai UPDATE un DELETE, vai jebkura cita kombinācija), trigera nosaukumā kopā jāizmanto vārdi 'ins','upd','del'. Piemēram, ja tabulai Title viens trigeris ir INSERT un UPDATE, tad trigerim jāpiešķir šāds nosaukumstitles\_InsUpdtrg | titles\_instrgtitles\_updtrgtitles\_deltrgtitles\_ValidateData\_instrgtitles\_MakeAuditEntries\_instrgtitles\_InsUpdtrg |
| Parametri | Procedūru, trigeru un lietotāja definētas funkcijas parametru identificēšanai kā piedēkli nepieciešams izmantot ‘p’, bet parametru nosaukumu veidot, pielietojot *Camel* reģistru. | p\_titlep\_customerID |
| Indeksi | Tāpat kā trigeri, arī indeksi nevar eksistēt atdalīti un tie ir atkarīgi no bāzes tabulas. Tātad, ir vērts indeksa nosaukumā iekļaut tabulas un kolonnas nosaukumu, kurai ir izveidots indekss. Ir divu tipu indeksi: sagrupēti (*clustered*) un nesagrupēti (*nonclustered*). Šie divi indeksu tipi var būt unikāli vai neunikāli. Tātad nosaukumu piešķiršanas noteikumiem jāapraksta arī indeksu tipi. Indeksu nosaukuma piešķiršanas noteikuma piemērs:Tabulas nosaukums + Kolonnas nosaukums(i) + unikāls/neunikāls + sagrupēts/nesagrupēts (*clustered/nonclustered*)Piemēram, unikālam, sagrupētam indeksam, kas bāzēts uz tabulas Titles kolonnu TitleID, nosaukums būtu šāds:Titles\_TitleID\_U\_CidxUnikālam, nesagrupētam indeksam, kas bāzēts uz tabulas *Publishers* kolonnu *PubID*, nosaukums būtu šāds:Publishers\_PubID\_U\_NidxNeunikālam, nesagrupētam indeksam, kas bāzēts uz tabulas *OrderDetai*ls kolonnu *OrderID*, nosaukums būtu šāds:OrderDetails\_OrderID\_NU\_NidxIndeksi var būt arī kompleksi, t.i., indekss var būt bāzēts uz vairāk kā vienu kolonnu. Tādā gadījumā visi kolonnu nosaukumi jāapvieno kopā tāpat, kā tas tika aprakstīts iepriekš pie palīgtabulām un skatiem.Kompleksam, unikālam, sagrupētam indeksam, kas bāzēts uz tabulas *OrderDetails* kolonnām *OrderID* un *OrderDetailID*, nosaukums būtu šāds:OrderDetails\_OrderIDOrderDetailID\_U\_CidxProtams, tādi indeksu nosaukumi ir gari un nepatīkami. Bet, līdz ar to, nebūs nepieciešamības dot uz tiem norādes programmatūras kodā, ja vien nav vajadzības indeksu izveidot/nodzēst/pārveidot. | Titles\_TitleID\_U\_CidxPublishers\_PubID\_U\_NidxOrderDetails\_OrderID\_NU\_NidxOrderDetails\_OrderIDOrderDetailID\_U\_Cidx |
| Kolonnas | Kolonnas ir entītijas atribūti, t.i., kolonnas apraksta entītijas īpašības. Tātad kolonnas nosaukumam jābūt dabiskam un jēgpilnam. Pieļaujami šādi nosaukumu varianti:1. Piemēram, tabulai '*Customers*' varētu būt kolonnas:CustomerIDCustomerFirstNameCustomerAddressKā redzams, būtu labi kolonnas nosaukumam pievienot priedēkli ar entītijas apzīmējumu, kuru tie reprezentē.
2. Otrs variants. Katrai tabulai no datu bāzes jāizvēlas 2-4 burtu kombinācija un jāpārliecinās, ka katra kombinācija ir unikāla. Piemēram, '*Cust*' tabulai '*Customers*', '*Ord*' tabulai '*Orders*', '*OrdD*' tabulai '*OrderDetails*', '*Ad*t' tabulai '*Audit*' utt.Tabulas kodu izmanto kā priedēkli kolonnas nosaukumā. Šī varianta priekšrocība ir tā, ka vaicājumos daudzām tabulām ar kompleksiem JOIN, nav jāuztraucas par divdomīgiem kolonnu nosaukumiem un nav jāizmanto tabulas pseidonīmi, lai pievienotu priedēkli kolonnas nosaukumam. Tas padara vaicājumus viegli lasāmus. Piemēram, *CustomerID* kolonna tabulā '*Customer*': Cust\_CustomerID;vai kolonna *CustomerID* un tabula '*Orders*': Ord\_CustomerID.
3. Trešais variants (*Northwind*), kolonnu nosaukumus veido, nepievienojot tabulas nosaukumu vai priedēkli, izņemot primārās atslēgas lauku, kur jālieto entītijas nosaukums: CustomerIDFirstNameAddress
 | CustomerIDCustomerFirstNameCustomerAddress Cust\_CustomerIDOrd\_CustomerIDCustomerIDFirstNameAddress |
| Lietotāja definēti datu tipi | Lietotāja definēti datu tipi ir tikai apvāks bāzes tipiem, kurus piedāvā datu bāzes pārvaldības sistēma. Tos izmanto, lai uzturēt datu tipu secību dažādu tabulu līdzīgajiem atribūtiem. Piemēram, ja kolonna *CustomerID* parādās desmit tabulās, visos gadījumos jāizmanto līdzīgi datu tipi. Līdz ar to, lietotāja definēti datu tipi būtu ļoti ērti. Nepieciešams izveidot lietotāja definētu datu tipu kolonnai *CustomerID*, un tas jāizmanto kā datu tips visos *CustomerID* sastapšanas gadījumos.Vienkāršākā metode, lai nosauktu lietotāja definētus tipus ir: Kolonnas nosaukums + '\_type'.Piemēram, *CustomerID* tips būtu:CustomerID\_type | CustomerID\_type |
| Primārās atslēgas  | Primārā atslēga ir kolonna(s), kura var unikāli identificēt katru rindu tabulā. Tātad primārās atslēgas nosaukumam var izmantot entītijas nosaukumu, pievienojot ‘*ID*’.Primārai atslēgai kolonnas nosaukums no tabulas '*Customers*' būtu šāds:PK\_CustomerID, ko nepieciešams saskaņot ar izvēlētās kolonnas nosaukuma shēmu. | PK\_CustomerID |
| Ārējās atslēgas | Ārējās atslēgas tiek izmantotas, lai reprezentētu attiecības starp tabulām, kuras ir saistītas. Ārējā atslēga, kura norāda uz tabulas '*Customers*' kolonnu *CustomerID* no tabulas '*Orders*' kolonnas '*CustomerID*': FK\_CustomerIDFK\_Cust\_CustomerID | FK\_CustomerIDFK\_Cust\_CustomerID |
| *Default* un *Check* ierobežojumi | Katram *default/check* ierobežojumam jāizmanto kolonnas nosaukums un jāpievieno priedēkļi '*def*' un '*chk*'.Piemēram, *default* ierobežojums kolonnai *OrderDat*e būtu def\_OrderDate ,bet *check* ierobežojums kolonnai *OrderDate* būtu chk\_OrderDate. | def\_OrderDatechk\_OrderDate |
| Mainīgie | Mainīgiem, kuri glabā kolonnu saturu, varētu pielietot to pašu nosaukumu piešķiršanas metodi, kura tika aprakstīta saistībā ar kolonnu nosaukumiem. Jāizmanto *Camel* reģistrs. | @orderDate |
| Daži pamatanoteikumi | 1. Pēc iespējas neizmantot sarežģītus, garus nosaukumus tabulām, kā arī citiem datu bāzes objektiem.
2. Izmantot SajauktoReģistu, nevis pasvītrojumu, lai atdalītu vārdus nosaukumā.
3. Izmantot pasvītrojumu tikai starp piedēkli/priedēkli un objekta vārdu.
4. Neizmantot atstarpes datu bāzes objekta nosaukumā. Ja tomēr ir nepieciešams izmanto atstarpi, jāpārliecinās, ka nosaukums tiek ieslēgts kvadrātiekavās (Microsoft SQL Serverī). Piemēram, [Order Details].
5. Neizmantot rezervētus vārdus datu bāzes objektu nosaukšanā, jo tas var radīt neparedzamas situācijas.
6. Ja datubāzē eksistē dažādas objektu grupas, kas nav vai ļoti maz sasaistīti viens ar otru, vēlams katrai no šim grupām izmantot atsevišķu shēmas nosaukumu „dbo” vietā, piemēram:AUD.AuditDetailsLOG.LogDetails
 |  |

## MS SQL Serveru programmēšanas vadlīnijas

Šajā nodaļā ir aprakstītas SQL servera programmēšanas vadlīnijas un labās prakses paņēmieni. Šīm vadlīnijām un praktiskajiem paņēmieniem ir universālu likumu nozīme, bet iespējamas arī izņēmuma situācijas.

### Programmatūras koda lasāmība un formāts

1. Jāraksta komentārus saglabājamās procedūrās, trigeros un SQL paketēs ikreiz, kad ir kaut kādas neskaidrības. Tas palīdzēs citiem programmētājiem izprast programmatūras kodu, detalizētai informācijai skat.2.2.6. sadaļu.
2. Vienmēr konsekventi jālieto burtene (*case*) programmatūras kodos. Ja programmatūras kodā burtene nebūs konsekventi ievērota, tad, iespējams, uz burteni nejutīga servera programmatūras kods darbosies labi, bet uz burteni jutīga SQL servera tas cietīs neveiksmi. Piemēram, ja jāveido SQL servera tabula vai datu bāze, kura ir burteni jutīga vai bināri sakārtota secībā, visām norādēm uz tabulu jāizmanto burtene, kas ir specificēta priekšrakstā CREATE TABLE.
3. Nelietot kolonnu numurus (skaitļus) komandā (*clause*) ORDER BY.
4. Vecā stila savienojumu vietā izmantot vieglāk lasāmos ANSI standarta *Join* nobeigumus. Ar ANSI savienojumiem komanda WHERE tiek lietota tikai datu filtrēšanai. Ar vecāka stila savienojumiem komanda WHERE regulē (vada) gan savienojuma stāvokli (nosacījumu), gan datu filtrēšanu. Pirmajā no abiem zemāk esošajiem piemēriem parādīta vecā stila savienojuma sintakse, savukārt otrajā piemērā – jaunā ANSI savienojuma sintakse.

1. piemērs:

SELECT a.au\_id, t.title

FROM titles t, authors a, titleauthor ta

WHERE a.au\_id = ta.au\_id AND

ta.title\_id = t.title\_id AND

t.title LIKE '%Computer%'

2. piemērs:

SELECT a.au\_id, t.title

FROM authors a

INNER JOIN titleauthor ta ON a.au\_id = ta.au\_id

INNER JOIN titles t ON ta.title\_id = t.title\_id

WHERE t.title LIKE '%Computer%'

1. Lai SQL priekšraksti būtu vieglāk lasāmi, kad nepieciešams, katru komandu jāsāk jaunā rindā un jālieto atkāpe, piemēram:

SELECT title\_id, title

FROM titles

WHERE title LIKE '%Computer%' AND

title LIKE '%cook%'

1. Tāpat kā jebkurā citā programmēšanas valodā, nelietojiet GOTO vai arī lietojiet to ierobežoti. Pārmērīga GOTO lietošana var radīt grūti lasāmu un grūti saprotamu programmatūras kodu.

### Datu tipi

1. Ja atsevišķa kolonna atkārtojas daudzās tabulās, tad jāizmanto lietotāja definētie datu tipi, lai šīs kolonnas datu tipi būs atbilstoši visās tabulās.
2. NCHAR datu tipu kolonnai jāizmanto vienīgi tad, ja kolonna nav nenozīmīga (*non-nullable*). Ja NCHAR kolonna ir nenozīmīga, tā tiks apstrādāta kā fiksēta garuma SQL Servera 7.0+ kolonna. Līdz ar to, NCHAR(100), ja ir NULL, izmantos 100 baitus, radot telpas zudumu. Šādā gadījumā jāizmanto NVARCHAR(100). Protams, mainīga garuma kolonnām ir ļoti mazs apstrādes virstēriņš salīdzinājumā ar fiksēta garuma kolonnām. Izvēlei starp NCHAR un NVARCHAR jābūt pārdomātai, balstītai uz saglabājamo datu garumu.
3. Jāizmanto *Unicode* datu tipu, kā, piemēram: NCHAR un NVARCHAR, gadījumā, ja ir iespējams saglabāt kolonnā teksta datus vairākas valodas, kas izmanto ne-latīņu burtus. CHAR un VARCHAR datu tipi jālieto gadījumos kad kolonna tiks saglabātas tikai angļu rakstzīmes vai teksts tikai vienā valoda. Šajā gadījuma jāpārliecinās, kā datubāzes vai atsevišķas kolonnas kodējums (collation) atbilst izmantotai valodai.
4. Nedrīkst lietot TEXT ,NTEXT un IMAGE datu tipus, jo tie būs izņemti no aprites nākamas MS SQL Server versijās. Tajā vieta jāizmanto tipi VARCHAR, NVARCHAR un VARBINARY. VARCHAR un VARBINARY tipa kolonnām jānorada fiksēto izmēru gadījumā, ja datu garums nepārsniedz 8000 simboli. NVARCHAR tipa kolonnām jānorada fiksēto izmēru gadījumā, ja datu garums nepārsniedz 4000 simboli.

### Saglabātās procedūras

1. Funkcijas, kuru rezultāts nemainās visu koda izpildes ciklā, nedrīkst izsaukt vairakkārt saglabāto procedūru, trigeru, funkciju pakešu iekšienē. Piemēram, virknes mainīgā garums var būt vajadzīgs vairākās procedūras vietās, tāpēc jāizsauc LEN (datu garums) funkciju tikai vienreiz, bet rezultāts jāsaglabā mainīgajā, tā vēlākai izmantošanai. Nedrīkst izsaukt LEN funkciju vairākkārtīgi.
2. Jāpārliecinās, ka saglabātās procedūras vienmēr atgriež vērtību, kas apzīmē to statusu. Saglabāto procedūru atgrieztās vērtības par sekmīgu izpildi un kļūdu nepieciešams standartizēt. RETURN priekšraksts ir paredzēts vienīgi izpildes statusa, nevis datu atgriešanai. Ja ir nepieciešams atgriezt datus, jāizmanto OUTPUT parametri.
3. Ja saglabātā procedūra vienmēr atgriež vienrindas rezultātu kopu, jāapsver iespēja rezultātu kopas atgriešanai izmantot OUTPUT parametrus SELECT priekšraksta vietā, jo ADO izvades parametrus vada ātrāk, nekā SELECT priekšrakstu atgrieztās rezultātu kopas.
4. Kā jau tika minēts 2.3. nodalījumā, nedrīkst lietot prefiksu 'sp\_' saglabāto procedūru nosaukumos, jo prefikss 'sp\_' ir rezervēts sistēmas saglabātajām procedūrām, kas sadarbojas ar SQL serveri. Vienmēr, kad SQL serveris saskaras ar procedūras nosaukumu, kas sākas ar 'sp\_', tas šo procedūru vispirms mēģina saglabāt pamata datu bāzē, pēc tam - meklē paredzētos kvalifikatorus (datubāzi, īpašnieku), pēc tam mēģina izturēties kā īpašnieks. Neizmantojot prefiksu 'sp\_', var ietaupīt laiku saglabātās procedūras lokalizācijai (arī skat. 2.2.3. nodaļu).
5. Nepieļaut, ka klienta (*front-end*) aplikācijas pieprasa/manipulē ar datiem tieši, izmantojot SELECT vai INSERT/UPDATE/DELETE priekšrakstus. Tā vietā, jāizveido saglabātās procedūras un jāļauj savām aplikācijām piekļūt šīm procedūrām. Tas saglabās datu pieejamību ‘tīru’ un konsekventu visos aplikācijas moduļos, tajā pašā laikā centralizējot biznesa loģiku datu bāzē.

### Veiktspēja

1. Datu bāzes izstrādes laikā ir svarīgi domāt par aplikācijas veiktspēju. Reāli veiktspēju nav iespējams saskaņot vēlāk, kā vien datu bāzes izstrādes laikā, jo tas ietver tabulu un indeksu pārbūvi, pieprasījumu pārrakstīšanu u.c. Pieprasījumu analīzei jāizmanto grafisko ierobežojumu plānu Query Analyzer vai SHOWPLAN\_TEXT un SHOWPLAN\_ALL komandās. Jāpārliecinās, ka pieprasījumi izpilda indeksu meklēšanu ("Index seek"), nevis indeksu izvēršanu ("Index scan") vai tabulu izvēršanu ("Table scan”). Cik vien iespējams, vajag izvairīties no tabulu izvēršanas vai indeksu izvēršanas. Attiecīgajām kolonnām jāizvēlas pareizie indeksi.
2. Sākumā datiem jābūt normalizētiem vismaz līdz trešajai normālformai. Ja pēc tam kādus no datiem nepieciešams renormalizēt, lai uzlabotu veiktspēju, to var darīt, bet tikai pēc 3.normālformas. Nepieciešams dokumentēts loģisks pamatojums visām ar režīma pārkāpumu saistītām darbībām.
3. Pieprasījumos nedrīkst izmantot 'SELECT \*'. Nepieciešamās kolonnas nosaukums vienmēr jāraksta pēc SELECT priekšraksta, kā tas redzams piemērā:

SELECT CustomerID, CustomerFirstName, City

Šāda pieeja ļauj reducēt I/O disku un uzlabot veiktspēju.

1. Iespēju robežās jāizvairās veidot pagaidu tabulas, kamēr notiek datu apstrāde, jo pagaidu tabulu izveidei nepieciešams lielāks I/O disks. Jāapsver iespēja pagaidu tabulu vietā izmantot uzlabotu SQL, skatus, SQL servera 2000+ tabulu mainīgo vai atvasinātās tabulas.
2. Jāizvairās izmantot aizstājējzīmju rakstzīmes vārdu sākumā meklēšanas laikā, lietojot atslēgvārdu LIKE, kā rezultātā rodas pilns tabulas izvērsums, kas anulē indeksa mērķi, Pirmais, zemāk esošais, piemērs dod indeksa izvēršanu, savukārt, otrais – indeksa meklēšanu.

1. piemērs:

SELECT LocationID

FROM Locations

WHERE Specialties LIKE '%pples'

2. piemērs:

SELECT LocationID

FROM Locations

WHERE Specialties LIKE 'A%s'

Funkciju pielietojums SELECT priekšrakstos neradīs priekšrocības indeksēšanai.

1. Turklāt, meklēšanai jāizvairās lietot neatbilstošus operatorus (<> and NOT), jo tie rezultātā dod tabulu un indeksu izvērses.
2. Jālieto atvasinātās tabulas (derived tables) un CTE (Common Table Expressions), īslaicīgo tabulu un apakšvaicājumu „WHERE” apgalvojumā vietā, jo atvasinātās tabulas un CTE izpilda labāk. Kā piemēru var minēt šādu pieprasījumu, kuru izmanto, lai atrastu otru augstāko algu Personālsastāva tabulā:

SELECT MIN(Salary)

FROM Employees

WHERE EmpID IN (SELECT TOP 2 EmpID FROM Employees ORDER BY Salary Desc)

Izmantojot CTE, šo pieprasījumu iespējams pārrakstīt, kā redzams zemāk, un tas tiek izpildīts divreiz ātrāk, kā augstāk norādītais pieprasījums:

SELECT MIN(Salary)

FROM (SELECT TOP 2 Salary FROM Employees ORDER BY Salary Desc) AS A

Rezultāti var atšķirties dažādos scenārijos atkarībā no datu bāzes projektējuma, indeksiem, datu ietilpības utt. Jāizmēģina visus iespējamos pieprasījuma pieraksta veidus un jāizvēlas efektīvāko no tiem.

1. SQL pakešu sākumos jālieto SET NOCOUNT ON, vides izstrādei - saglabātās procedūras un trigerus, līdzīgi kā noklusētos ziņojumus '(1 row(s) affected)' pēc INSERT, UPDATE, DELETE un SELECT priekšrakstu izpildes. Tas uzlabo saglabāto procedūru izpildi, reducējot tīkla trafiku.
2. Jāizpilda visas integritātes pārbaudes un datu validāciju, lietojot ierobežojumus (svešas atslēgas un pārbaudes ierobežojumi) nevis trigerus, jo tie ir lēnāki. Trigerus jāizmanto vienīgi auditēšanai, klienta uzdevumiem un validācijai, ko nav iespējams izpildīt, lietojot ierobežojumus. Ierobežojumi ietaupa laiku, jo nav jāraksta programmatūras kodi šīm validācijām, viss darbs tiek atstāts *Relational Database Management System* (turpmāk tekstā – RDBMS) ziņā.

### Dažādi ieteikumi

1. Iespēju robežās jāmēģina izvairīties no servera puses kursoriem. Datu pieejamībai un manipulācijām (*procedural approach*) vietā vienmēr jāpielieto (*set-based approach*). Bieži no kursoriem var izvairīties, lietojot SELECT priekšrakstu. Ja kursors ir nenovēršams, jāizmanto WHILE cilpa, jo tā vienmēr ir ātrāka par kursoru. Lai kursoru aizstātu ar WHILE cilpu, nepieciešama kolonna (primārā atslēga vai unikālā atslēga) katras rindas unikālai identifikācijai. Jebkurā gadījumā ikvienai tabulai nepieciešama primārā atslēga vai unikālā atslēga.
2. Skatus parasti lieto, lai specifiskiem lietotājiem, balstoties un viņu interesēm, parādītu specifiskus datus. Skatus parasti lieto arī, lai ierobežotu pieeju datu bāzes tabulām, piešķirot atļauju vienīgi uz skatiem. Vēl viena nozīmīga skatu lietošanas iespēja – tie vienkāršo pieprasījumus. Iekļaujot bieži nepieciešanos, komplicētos savienojumus un aprēķinus skatā, tie nebūs jāatkārto katrā pieprasījumā, bet vienkārši jāizvēlas no skata.
3. Ja vien ir iespējams, jāizvēlas bināros vai attēlu failus (*Binary Large Objects or BLOBs*) nesaglabāt datu bāzē. Tā vietā jāsaglabā datu bāzē ceļu pie binārā vai attēla faila un jāizmanto to kā norādi uz aktuālo, kaut kur citur uz servera saglabāto, bināro failu. Ir labāk, ja šo lielo bināro failu izguve un darbības ar tiem notiek ārpus datu bāzes. Jāpatur prātā, ka datu bāze nav paredzēta failu uzglabāšanai. E-veselības IP aplikāciju ietvaros plānots bināros failus saglabāt Elektroniskajā dokumentu krātuvē.
4. Iespēju robežās jāizvairās no dinamiskajiem SQL priekšrakstiem. Dinamiskie SQL tiecas būt lēnāki par statiskajiem, tā rezultātā SQL serverim jāģenerē izpildes plāns katram izpildlaikam. IF un CASE priekšraksti ir parocīgi, lai izvairītos no dinamiskajiem SQL. Otra dinamisko SQL lietošanas nepilnība ir nepieciešamība, ļaut lietotājam tieši piekļūt visiem piekļuves objektiem., līdzīgi kā tabulām un skatiem. Parasti lietotājiem piešķir pieeju saglabātām procedūrām, kurās ir norāde uz tabulām, bet nevis piešķir piekļuvi pašām tabulām. Šajā gadījumā dinamiskie SQL nedarbosies.
5. Iespēju robežās jāizvairās no NULL vērtības izmantošanas. Piemēram, ja nav uzdota teksta vērtība, jāizmanto tukšo simbolu virkne. Šī pieeja samazina lieko konvertējumu daudzumu starp priekšgalā (front-end) attēloto informāciju un datubāzē glabājamo informāciju.
6. INSERT priekšrakstā vienmēr jālieto kolonnas saraksts. Tas palīdzēs izvairīties no problēmām, mainot tabulas struktūru (pievienojot vai izslēdzot kolonnu).
7. Piekļuve pie tabulām visās glabājamās procedūrās un trigeros vienmēr notiek vienā un tajā pašā veidā. Tas ļauj izvairīties no strupsaķerēm (*deadlock)*. Tālāk aprakstīti citi pasākumi, kas jāņem vērā, lai izvairītos no strupsaķerēm:
	1. Transakcijām jābūt iespējami īsām.
	2. Datu apmēram transakcijas laikā jābūt iespējami mazākam.
	3. Nedrīkst gaidīt, ka lietotājs ievadīs kādu informāciju transakcijas laikā.
	4. Nedrīkst izmantot augstākā līmeņa slēdzošas norādes (*locking hints*) vai ierobežojošus izdalīšanas līmeņus (*restrictive isolation levels*), ja vien tie nav kritiski nepieciešami.
	5. Priekšgala (*front-end*) lietotnes jāveido strupsaķeres inteliģentas (*deadlock-intelligent*), t.i., šīm lietotnēm jāspēj atkal iesniegt (*resubmit*) transakciju, ja gadījumā iepriekšējā transakcija ir cietusi kļūdu 1205.
	6. Visi SQL servera atgrieztie rezultāti nekavējoties ir jāapstrādā lietojumā, lai iespējami ātrāk atbrīvotu bloķētos datu bāzes ierakstus.
8. Uzdevumi, piemēram, virknes (*string*) manipulācijas, konkatenācija, rindu numurēšana, burtenes konversija u.c., jāpārnes uz lietojumu no datu bāzes servera pie nosacījuma, ka tie patērēs pietiekami daudz CPU laika. Turklāt, vienkāršajām validācijām jānotiek datu ievades laikā lietojumā vai lietojuma biznesa loģikas klasē. Tas samazinās nevajadzīgu tīkla apriti (*roundtrips*).
9. MSSQL 2000 serveris tūlīt pēc datu manipulācijas priekšraksta (INSERT/UPDATE/DELETE) izpildes vienmēr pārbauda MSSQL 2000 globālo mainīgo, līdz ar to, kļūdas gadījumā iespējams atritināt (*rollback*) transakciju (kļūdas gadījumā @@ERROR vienmēr būs lielāks par 0). Tas ir svarīgi, jo pēc noklusējuma SQL serveris nevar atritināt visas iepriekšējās izmaiņas transakcijas ietvaros, ja specifiskais priekšraksts neizdodas. Šādu norisi iespējams mainīt, izpildot XACT\_ABORT ON. MSSQL 2005 serverim izmantojiet “*try – catch*” blokus.
10. @@ROWCOUNT mainīgajam parasti ir liela nozīme, lai noteiktu, cik rindas ir skāris iepriekšējais datu manipulācijas priekšraksts (arī izguve) un, balstoties uz to, var izvēlēties nodot (*commit*) vai atritināt (*rollback*) specifisku transakciju.
11. Datumos 2 ciparu gadskaitļa vietā vienmēr jāsaglabā 4 ciparu gadskaitli (īpaši, ja tiek lietotas CHAR vai INT datu tipu kolonnas), lai izvairītos no problēmām un neskaidrībām. DATETIME kolonnu gadījumā tā nebūs problēma, jo gadsimts tiks saglabāts pat, ja būs specificēts 2 ciparu gadskaitlis. Vienmēr labāk ir saglabāt 4 ciparu gadskaitli, arī DATETIME kolonnu gadījumā.

Nedrīkst aizmirst izpildīt unikālos ierobežojumus savām paralēlatslēgām (*alternate keys*).

# Oracle datu bāzes un Java programmēšanas vadlīnijas

## Java projektu izstrādes vadlīnijas

### Programmēšanas vide

Risinājuma dažu moduļu izstrāde pamatā tiek veikta *Netbeans* izstrādes vidē. Atkarībā no projekta prasībām, var tikt izmantota *JDeveloper* vai *Eclipse* vide.

Java lietojumu izstrādē ir ieteicams izmantot *Apache* *Maven* programmatūru un rīkus. *Maven* nodrošina standartizētu lietojuma projekta un atkarību pārvaldību, automatizētu testu un būvējumu pārvaldību. *Maven* projekti nav atkarīgi no konkrētas izstrādes vides un projekta konfigurācija tiek aprakstīta centrālā XML datnē *pom.xml.*

### Maven vadlīnijas

#### Projekta struktūra

*Maven* projektam ir sekojoša standarta direktoriju struktūra (atkarībā no projekta veida, var būt specifiskas apakšdirektorijas):



1.attēls. Maven direktoriju struktūra

Projekta saknes direktorijā ir projekta konfigurācijas datne *pom.*xml, un divas direktorijas – *src*, kas satur visu projekta kodu un *target*, kas satur kompilētos un ģenerētos objektus.

Projektā var būt sekojošas apakšdirektorijas:

* src/main/java – saknes direktorija Java pirmkodam;
* src/main/resources – dažādi lietojumā izmantojamie resursi – piemēram, attēli;
* src/main/filters – resursu filtri;
* src/main/config – konfigurācijas datnes;
* src/main/webapp – projekta Web lapas (ja tādas ir);
* src/test/java – moduļu testi;
* src/test/resources – moduļu testu resursi;
* src/main/filters – moduļu testu resursu filtri;

#### Projekta īpašības

Projekta konfigurācijas datnē *pom.xml* kā minimums ir jānorāda šādas konfigurācijas īpašības:

* **groupId** – izstrādātājs – lv.abcsoftware;
* **artifactId** – projekta identifikators;
* **version** – ģenerējamā objekta versija;
* **name** – projekta nosaukums

#### Produktu versiju piešķiršana

Ģenerējamajam produktam tiek uzstādītas versijas šādā formātā:

##### Produkta versija

Projekta versijas formāts „M.m.b[-r]”, kur

M -pamatversijas numurs;

m - otrais versijas numurs;

b - laidiena numurs;

r – iespējama vērtība „SNAPSHOT”, kas nozīmē, ka ģenerētais objekts vēl nav fināla versija, bet vēl ir izstrādē;

##### Versiju numuru izmaiņas nosacījumi

Izstrādes laikā, ģenerējam produktam versijai tiek pievienots „SNAPSHOT”.

Laidiena revīzijas numurs tiek mainīts pēc jaunā laidiena sagatavošanas. Laidiena versija tiek mainīta pēc mazām izmaiņām vai labojumiem *produkt*

Otrais versijas numurs tiek mainīts, ja tiek veiktas izmaiņas, kas noved pie funkcionalitātes izmaiņām vai jaunās funkcionalitātes pievienošana.

Pamatversijas numurs tiek mainīts, ja tek mainīta produkta struktūra un izveides principi.

### Projekta izmaiņu apraksts

*Maven* projektam tiek ģenerēta projektu aprakstošā vietne HTML formātā, izmantojot “mvn site” mērķi. Projekta izmaiņas tiek ģenerētas projekta vietnes ietvaros, izmantojot *Maven* spraudni “Maven changes plugin”.

Katram laidienam ir jāapraksta izmaiņas, kas veiktas tā ietvaros.

### Java koda izstrādes vadlīnijas

Rakstot Java kodu, ir vēlams ievērot Java valodas izstrādes vadlīnijas, kuras aprakstītas [4] avotā. Tādējādi izstrādātais kods būs labāk pārskatāms un uzturams. [4] avots iekļauj arī komentāru nosacījumus. Publiskajām klasēm un metodēm ir ieteicams rakstīt dokumentēšanas komentārus, no kuriem pēc tam būs iespējams uzģenerēt API dokumentāciju.

### Java EE vadlīnijas

Izstrādājot Java EE lietojumus, jāievēro lietojumu un komponentu nosaukumu piešķiršanas vadlīnijas, kuras aprakstītas [6] avotā.

## Vienošanās par nosaukumu piešķiršanu Oracle datu bāzes objektiem

Ieteicamā nosaukumu piešķiršanas vienošanās Oracle datu bāzes objektiem ir aprakstīta 3.tabulā.

**4. tabula**

**Oracle nosaukumu vadlīniju saraksts**

|  |  |  |
| --- | --- | --- |
| **Tips** | **Standarts / Noteikums** | **Piemērs** |
| Tabulas | Tabulas nosaukumam jābūt daudzskaitlī un tam saprotami jāapraksta glabājamie objekti. (izņēmums varētu būt kaut kādas pagaidu tabulas, vai tabulas, kurām nevar nospecificēt glabājamo objektu)Ja gadījumā, nosaukums paredz vairākus vārdus, tad tas jāatdala ar apakšsvītru „\_” | CUSTOMERSCUSTOMER\_ORDERS |
| Saīsinājumi (alias) | Katrai tabulai ir unikāls saīsinātais nosaukums, kas tiek izmantots ārējo atslēgu, trigeru, indeksu nosaukumu veidošanā. | EMPLOYEES -> EMPCUSTOMERS -> CUSTDEPARTMENTS -> DEP |
| Atribūti/ kolonas | Kolonas nosaukumam īsi jāapraksta tās nolūks. Kolonas nosaukumā kā prefikss nav jālieto tabulas saīsinājums | NAMEBIRTH\_DATEnepareizi - CUST\_NAME |
| Ierobežojumi, atslēgas (constraints, keys) | Primārās atslēgas nosaukums sastāv no tabulas nosaukuma ar „\_ID” sufiksu. Primārās atslēgas ierobežojuma nosaukums sastāv no tabulas saīsinājuma ar „\_PK” sufiksu.Unikālās atslēgas nosaukums sastāv no tabulas saīsinājuma ar „\_UK” prefiksu.Tabulas ārējās atslēgas nosaukums ir vienāds ar saistītās tabulas primārās atslēgas nosaukumu. Tabulas ārējās atslēgas ierobežojuma sastāv no abu tabulu saīsinājumiem, ar pievienotu „\_FK” sufiksu.*Check* ierobežojumi sastāv no tabulas saīsinājuma, saīsināta lauka nosaukuma ar pievienotu sufiksu „\_CHK” | EMPLOYEES\_IDEMP\_PKEMP\_UKDEPARTMENTS\_IDEMP\_DEP\_FKEMP\_BDAY\_CHK |
| Trigeri | Trigera nosaukumā tiek izmantots tabulas nosaukums, funkcionalitāte, un trigera darbības veidu – „INS”, „UPD”. Ja trigeris izpildās pie vairākām darbībām, tad norāda visas izmantojamās darbības. | CUSTOMER\_INSCUSTOMER\_INS\_UPDCUSTOMER\_NOTIFICATION\_INS |
| Indeksi | Indeksa nosaukumā iekļauj tabulas saīsinājumu, indeksējamo lauku saīsinātus nosaukumus, kā arī sufiksu „\_IDX” | CUST\_ID\_IDXEMP\_NAME\_BDATE\_IDX |
| Sekvences (Sequences) | Sekvences parasti tiek izmantotas tabulas ID ģenerēšanai, līdz ar to tipisks sekvences nosaukums ir tabulas nosaukums ar „\_SEQ” sufiksu | CUSTOMERS\_SEQ |
| Skati | Skatu nosaukumiem izmanto kaut ko nozīmējušu nosaukumu, ar prefiksu „V\_”.Materializētajiem skatiem izmanto prefiksu „MV\_” | V\_TOP\_CUSTOMERSMV\_TOP\_CUSTOMERS |

## Oracle PL/SQL programmēšanas vadlīnijas

Kodējot PL/SQL procedūras, ir ietiecams izmantot vadlīnijas, kuras aprakstītas [5] avotā.