10. Objektinės technologijos reliacinėse DB

10.1. Įvadas

Reliacinės DB paplitę daug plačiau negu visų kitų rūšių duomenų bazės kartu. Taip yra ir Lietuvoje, ir visame pasaulyje. "Grėsmę" tokiam "viešpatavimui" pastaraisiais metais kelia tik objektinės technologijos (angl. *object oriented technology*). Pavyzdžiui, objektinės programavimo kalbos (C++ ir Java) tapo vienomis populiariausių programavimo kalbų, gal net pačiomis populiariausiomis.

Grupė didžiųjų pasaulio informacinių technologijų firmų 90-jų pradžioje ėmė diegti objektines technologijas duomenų bazių sistemose. Buvo prognozuojama, kad objektinės DBVS (ODBVS) taps tokiomis pat populiariomis, kaip ir objektinės programavimo kalbos. Tačiau prognozės, bent jau iki šiol, nepasitvirtino. Nors ODBVS įdiegimų skaičius pasaulyje santykinai auga greičiau negu RDBVS įdiegimų skaičius, tačiau absoliutūs augimo tempai yra ryškiai RDBVS naudai. Jeigu panaši tendencija išliks ir ateityje, tai RDBVS vyraus dar daugelį metų.

Tai, kad RDBVS ir toliau dominuoja, didelę įtaką padarė pačių RDBVS vystymasis. Komercinių RDBVS kūrėjai pripažįsta objektinių technologijų privalumus ir energingai diegia jas reliacinėse DB valdymo sistemose, taip išplėsdami sistemų galimybes. Dauguma šiuolaikinių RDBVS pradėta vadinti objektinėmis-reliacinėmis DBVS (ORDBVS).

Šiame skyriuje trumpai apžvelgsime objektinių DB privalumus bei trūkumus ir susipažinsime su pagrindinėmis objektinių-reliacinių DBVS savybėmis.

10.2. Objektinės duomenų bazės

Skirtingai nuo reliacinio duomenų modelio, kuris grindžiamas griežtais matematiniais apibrėžimais, objektinės duomenų bazės (ODB) tokio griežto teorinio pagrindo neturi. Kadangi pagrindinės objektinių DB sąvokos yra labai artimos objektinio programavimo sąvokoms, tai jų detaliai nenagrinėsime.

- **Objektai**. Visi ODB duomenys yra vaizduojami objektais. Reliacinėms duomenų bazėms būdingas duomenų tvarkymas lentelių eilutėmis objektinėse DB yra pakeistas objektų tvarkymu.
- **Klasės**. Paprastas reliacinių DB duomenų tipas ODB yra pakeistas hierachine klasės sąvoka. Kiekvienas objektas priklauso konkrečiai objektų klasei.
- **Paveldimumas**. Objektai paveldi savo klasės ir visų aukštesnių klasių savybes.
- **Atributai**. Atributai – tai klasės objektų charakteristikos. Atributo sąvoka yra labai artima stulpelio sąvokai reliacinėse DB.
- **Pranešimai ir metodai**. Objektai "bendrauja" tarpusavyje pranešimais. Objektas, gavęs pranešimą, įvykdo atitinkamą metodą – vidinį paprogramį, kuris realizuoja pranešimo apdorojimą. Objekto elgesys aprašomas metodų aibe.
- **Inkapsuliacija**. Vidinė objekto struktūra ir duomenys yra paslėpti. Informaciją apie objekto būseną galima gauti tik metodais. Objekto būsena keičiama taip pat metodais.
- **Objektų tapatumo požymiai**. Objektai yra atskiriami vienas nuo kito pagal tapatumo požymį (angl. *object ID - OID*). Jie naudojami sąryšiuose tarp objektų. Tapatumo požymį objektui priskiria DBVS. Paprasčiausių objektų, pvz., skaičių, tapatumas nustatomas pagal jo reikšmę. Sudėtingesnių objektų tapatumui nustatyti naudojami loginiai objektų adresai.

Šių principų dėka ODB puikiai tinka uždaviniams, kuriuose apdorojami labai sudėtingos struktūros duomenys, pvz., kompiuterinio modeliavimo ir struktūriškų dokumentų apdorojimo sistemose. Tokioms sistemoms yra būdingos hierarchinės duomenų struktūros, kurios patogiai vaizduojamos objektais ir jų klasėmis.

Pagrindinis ODB šalininkų argumentas yra taikomajai sričiai būdingos duomenų struktūros tiesioginis vaizdavimas objektais tiek duomenų bazėje, tiek ir taikomosiose

programose. Tai labai svarus argumentas prieš gana nelankstų reliacinėse DB vartojamą lentelių modelį. Realaus pasaulio esybės dažnai daug paprasčiau modeliuojamos lanksčia objektų klase, negu lentele. Be to, kai duomenys apdorojami programomis, sudarytomis objektinio programavimo kalba, nėra duomenų struktūrų suderinamumo sunkumų.

Savo ruožtu, ODB oponentai mano, kad nėra būtinybės pereiti nuo RDB prie ODB. Teigiama, pvz., kad objekto tapatumo požymio sąvoka yra artima nuorodos į duomenis sąvokai, kuri buvo būdinga senesniosioms, ikireliacinėms DB. Kadangi ODB neturi griežto matematinio pagrindo, tai yra ir standartizavimo sunkumų. Be to, reliacinio duomenų modelio ir objektinės programavimo kalbos duomenų struktūrų atitikimo sunkumus galima įveikti įvedant naujas objektines-reliacines sąsajas.

Komercinių RDBVS kūrėjai, atsižvelgdami į objektinių technologijų privalumus ir siekdami išlaikyti savo pozicijas rinkoje, įdiegė nemažai sąvokų, kurios anksčiau buvo būdingos tik ODBVS. Taip RDBVS, išsaugodamos reliacinio modelio privalumus, įgavo naujų bruožų, dėl kurių jos pradėtos vadinti objektinėmis-reliacinėmis DBVS (ORDBVS). Vartotojo sąsajos su DB vaidmenį objektinėse-reliacinėse sistemose atlieka SQL kalba, kurios galimybės labai išsiplėtė. Objektines-reliacines DB aptarsime detaliau.

10.3. Objektinės-reliacinės duomenų bazės

Kad pasinaudoti gerosiomis objektinių DB savybėmis, objektinėse-reliacinėse DB įdiegta gana daug papildymų. Tarp jų:

- **Dideli duomenų objektai.** Tradiciniai reliaciniai duomenų tipai yra nedidelės apimties - iki keleto tūkstančių baitų. Dideliuose duomenų objektuose (angl. *large objects*, *LOB*) galima talpinti didelius tekstų procesoriais paruoštus dokumentus, grafinius vaizdus, garso ir vaizdo įrašus, kitus audiovizualinius duomenis.
- **Vartotojo duomenų tipai**. Tradicinėse reliacinėse sistemose vartojamų duomenų tipų rinkinys yra ribotas. Galimybė vartotojui pačiam apibrėžti naujus duomenų tipus panaikino šį ribotumą.
- **Struktūriniai duomenų tipai**. Reliacinių duomenų tipų reikšmės yra nedalomi atomai. Naujieji struktūriniai duomenų tipai leidžia apjungti kelis vienodų ar skirtingų tipų elementus į vientisą struktūrą, kartu išsaugant ir kiekvieno jų išskirtinumą.
- **Vartotojo funkcijos**. Kad būtų tenkinami pastoviai augantys vartotojų poreikiai, reliacinėse DBVS buvo diegiamos naujos skaliarinės bei agregatinės funkcijos. Problema tapo nebeaktuali įdiegus galimybę vartotojui pačiam apibrėžti reikiamas funkcijas. ORDB sistemose funkcijas galima realizuoti įprastomis, tame tarpe objektinėmis, programavimo kalbomis.
- **Eilučių tapatumo požymiai**. Šiuolaikinės ORDBVS automatiškai suteikia lentelių eilutėms unikalius tapatumo požymius, atliekančius pirminio rakto vaidmenį.
- **Aktyvūs duomenys** – tai duomenys su apibrėžtomis elgesio taisyklėmis. Taisyklių prisilaikymu užtikrinamas duomenų vientisumas. Pagal nustatytas taisykles apdorojamos duomenų įvedimo ir keitimo klaidas ir pan. Duomenų elgesys aprašomas deklaratyviais apribojimais reikšmėms, dalykinėmis taisyklėmis (trigeriais) ir kt.

Tekstų procesoriais paruošti dokumentai, skaitmeniniai garsų įrašai ir pan. kompiuterio atmintyje gali užimti daug kilobaitų ar net megabaitų. Tokių duomenų neįmanoma įvesti į RDB tradicinėmis priemonėmis, nurodant juos betarpiškai SQL sakinyje. ORDBVS įdiegta galimybė nurodyti reikšmę bylos vardu. Daug vietos kompiuterio atmintyje reikalaujančias reikšmes, esančias duomenų bazėje, galima nuskaityti į bylą, o paskui apdoroti specialia programa, pavyzdžiui, tektų procesoriumi. Šiuolaikinėse sistemose didelės apimties reikšmę galima vartoti netgi duomenų paieškos sąlygoje. Daugelyje komercinių sistemų yra simboliniai (angl. *character large object*, `CLOB`) ir dvejetainiai dideli objektai (angl. *binary large object*, `BLOB`). `CLOB` skirti dideliems tekstams, o `BLOB` įvairių taikymų dvejetainiams kodams.

Įvedus į ORDB didelius objektus ir kitas objektines priemones (visų pirma, naujus duomenų tipus ir funkcijas), susidarė galimybė sudarinėti **reliacinius išplėtimus** (angl.

relational extender), skirtus atskiroms DB taikymo sritims. Reliacinio išplėtimo sąvoka nėra standartizuota. Skirtingose komercinėse RODBVS reliaciniai išplėtėjai vadinami skirtingai, pvz., DB2 jie vadinami "*Relational Extenders*", Oracle – "*Data Options*", Informix – "*Data Blades*". Reliacinio išplėtimo sąvoka iš dalies atitinka klasių bibliotekos sąvoką objektinio programavimo sistemose. Supaprastintai reliacinį išplėtimą galima apibūdinti kaip papildomus duomenų tipus ir funkcijas, kurie sprendžia konkrečios taikomosios srities uždavinius.

Išvardinsime gana dažnai vartojamus reliacinius išplėtimus, būdingus objektinei-realiacinei sistemai DB2:

- Grafinis išplėtimas (angl. *image extender*), kuris leidžia vartoti visus populiariausius grafinių duomenų formatus: GIF, JPEG, BMP, TIFF, bei konvertuoti duomenis iš vieno formato į kitą. Darbui su grafiniais vaizdais yra sudarytos tokios pagrindinės galimybės: pateikti nedidelį surasto vaizdo fragmentą pradinei jo peržiūrai, pavaizduoti užklausos rezultatą grafiškai, atlikti kontekstinę vaizdų paiešką pagal nurodytą vaizdo ar jo dalies spalvą ir kontrastą, ieškoti vaizdų, panašių į pateiktąjį pagal spalvą bei faktūrą, pagal nurodytą kontūrą ir pan.
- Vaizdų išplėtimas (angl. *video extender*) leidžia įsiminti ir pateikti vaizdo įrašus populiariuose formatuose: MPEG1, MPEG2, AVI, Quicktime. Sudaryta galimybė automatiškai segmentuoti įrašus pagal scenos pasikeitimus bei surasti scenos atstovą. Funkcijomis galima sužinoti įvairias vaizdo įrašo charakteristikas.
- Garsų išplėtimas (angl. *audio extender*) leidžia įsiminti ir pateikti garsų įrašus visuose populiariuose formatuose: AIFF, MIDI, WAVE. Funkcijomis galima sužinoti daugelį įrašų charakteristikų: įrašo trukmę, garso takelių skaičių ir pan.
- Tekstų išplėtimas (angl. *text extender*), kuris:
  - leidžia išsaugoti duomenų bazėje įvairiais tekstų procesoriais (*Microsoft Word*, *Word Perfect*, *AmiPro*) sudarytus dokumentus. Tokie tekstai gali tapti stulpelio reikšme konkrečioje lentelės eilutėje;
  - sukuria specializuotus indeksus kontekstinei (pagal žodžius, jų dalis ir frazes) paieškai dideliuose tekstuose;
  - atsižvelgia į tai, kuria kalba (anglų, vokiečių, prancūzų ir kt.) yra sudarytas dokumentas;
  - ieškant duomenų bazėje dokumentų, atsižvelgia į žodžių sinonimus bei formas.

Sudarysime užklausą, panaudodami joje tekstų išplėtimo funkciją. Tarkime, DB *Darbai* lentelėje *Vykdytojai* yra stulpelis *CV*, kurio reikšmės yra sudarytos tekstų procesoriumi, pvz., *Microsoft Word*. Šio stulpelio reikšmė – darbuotojų gyvenimo aprašas (autobiografija). Sąrašą darbuotojų, kurių autobiografijoje yra paminėtas žodis "stažuotė", bei bent vienas iš žodžių "Vokietija" ar "Prancūzija", ir tekste nėra žodžio "JAV", gausime įvykdę užklausą:

```
SELECT Nr, Pavardė FROM Vykdytojai
WHERE  CONTAINS( CV,
          '("stažuotė" & ("Vokietija" | "Prancūzija") & NOT "JAV" )' ) > 0 .
```

Funkcija `CONTAINS` yra dviejų argumentų: pirmasis - didelis tekstas (simbolinis didelių objektas, `CLOB`), o antrasis – paprasta simbolių eilutė, kurioje tarp dvigubų kabučių rašomi prasmingi žodžiai, kurie jungiami loginėmis operacijomis. Ši simbolių eilutė tekstų išplėtimo funkcijai `CONTAINS` yra sąlyga kontekstinei dokumentų paieškai atlikti. Apdorojant šią simbolių eilutę yra analizuojama jos struktūra.

Jau minėjome, kad objektinėse-reliacinėse DB, lyginant jas su reliacinėmis DB, yra įdiegta daug naujų sąvokų. Plačiau aptarsime tik dvi iš jų: vartotojo duomenų tipus bei funkcijos.

10.4. Naujų duomenų tipų apibrėžimas

Naudodamas jau esamus duomenų tipus vartotojas gali apibrėžti naujus (angl. *user-defined type*). Tam SQL kalba yra papildyta sakiniu `CREATE DISTINCT TYPE`.

Tarkime, DB *Darbai* lentelėje *Projektai* mums reikia saugoti ir projekto vertę pinigine išraiška. Projekto vertė – tai sutartinė suma, kuri patenka į įstaigos sąskaitą bazine užsienio valiuta ir/arba į sąskaitą litais. Kadangi suma nėra konvertuojama pinigų pervedimo ir duomenų įvedimo metu, be to, santykis tarp lito ir bazinės valiutos gali keistis, todėl patogu lentelę papildyti ne vienu, o dviem naujais stulpeliais *Vertė_Valiuta* ir *Vertė_Lt*. Abu šie stulpeliai galėtų būti `DECIMAL` tipo. Tačiau tuomet užklausose galima rašyti, pavyzdžiui, tokią sąlygą *Vertė_Valiuta* < *Vertė_Lt*, kuri nors ir yra sintaksiškai teisinga, bet joje lyginamos dvi logiškai nesulyginamos (netiesiogiai sulyginamos) reikšmės. Todėl kiekvienai valiutai apibrėžkime atskirą duomenų tipą:

```
CREATE DISTINCT TYPE Valiuta  AS DECIMAL(15, 2) WITH COMPARISONS,
CREATE DISTINCT TYPE Litai    AS DECIMAL(15, 2) WITH COMPARISONS,
```

čia *Valiuta* ir *Litai* yra naujų duomenų tipų vardai, o frazė `WITH COMPARISONS` nurodo, kad apibrėžiamo tipo reikšmes bus galima vartoti palyginimo operacijose: <, >, = ir kt. Naujojo tipo reikšmių negalima lyginti, jei ši frazė nenurodoma. Nurodant tipų vardus, kaip ir kitų DB objektų vardus, galima papildyti juos schemos vardu. Vartotojo apibrėžtus tipus galima vartoti, kaip standartinius, pvz.,

```
ALTER TABLE Projektai ADD Vertė_Valiuta Valiuta;
ALTER TABLE Projektai ADD Vertė_Litais  Litai.
```

Sukūrus naują tipą automatiškai sudaroma funkcija naujojo tipo reikšmėms sudaryti. Tos funkcijos vardas sutampa su tipo vardu, o argumentas yra bazinio tipo. Sudarykime užklausą projektams, kurių vertė užsienio valiuta (valiutinėje sąskaitoje) yra didesnė už 10000:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta > Valiuta(10000) ,
```

čia reiškinio *Valiuta*(10000) tipas - *Valiuta*, o jo reikšmė yra 10000. Sakinys:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta > 10000 ,
```

yra sintaksiškai neteisingas. Naujai apibrėžtiems tipams SQL nustato griežtą tipų kontrolę. Sintaksiškai neteisingas ir toks sakinys:

```
SELECT Pavadinimas FROM Projektai WHERE Vertė_Valiuta < Vertė_Litais .
```

Norint palyginti dvi skirtingų tipų reikšmes, reikia atlikti tipų suvienodinimą, pvz., vieną iš reikšmių paversti kito tipo reikšme. Apibrėžiant naują tipą, kartu sudaroma ir funkcija naujojo tipo reikšmei paversti bazinio tipo reikšme, pvz.,

```
SELECT Pavadinimas FROM Projektai
WHERE DECIMAL(Vertė_Valiuta) < DECIMAL(Vertė_Litais).
```

Šios užklausos sintaksė yra teisinga, tačiau logiškai sakinys nėra teisingas, kadangi lyginami skirtingų dimensijų dydžiai. Panašiam palyginimui reikalingas valiutos konvertavimas. Tam užklausoje galima įvesti daugiklį, pvz.,

```
SELECT Pavadinimas FROM Projektai
WHERE DECIMAL(Vertė_Valiuta) * 4 < DECIMAL(Vertė_Litais).
```

Tačiau, jei kursas gali kisti, tai užklausą reikia parametrizuoti. Paprasčiausias būdas įtraukti parametrą, kurį galima vartoti ir interaktyviame SQL, yra sudaryti skaliarinę funkciją.

10.5. Naujų funkcijų apibrėžimas

Sukuriant naują duomenų tipą, jokios kitos funkcijos, išskyrus reikšmių pervedimo tarp bazinio ir naujojo tipo funkcijas, automatiškai nėra sudaromos. Reiškinys *Valiuta*(10) + *Valiuta*(20) yra sintaksiškai neteisingas, jei duomenų tipui *Valiuta* nėra apibrėžta sudėties operacija.

Naujos funkcijos apibrėžiamos sakiniu `CREATE FUNCTION`. Sakinyje nurodoma apibrėžiamos funkcijos vardas, argumentų kiekis bei jų tipai, rezultato tipas ir funkcijos reikšmės apskaičiavimo būdas. Paprasčiausiai funkciją galima apibrėžti betarpiškai per atitinkamą bazinio tipo funkciją. Pvz.:

```
CREATE FUNCTION "+"(Valiuta, Valiuta) RETURNS Valiuta
   SOURCE "+"(DECIMAL(15, 2), DECIMAL(15, 2));
CREATE FUNCTION "*"(Valiuta, DECIMAL(10, 5)) RETURNS Valiuta
   SOURCE "*"(DECIMAL(15, 2), DECIMAL(10, 5)).
```

Apibrėžus šias dvi funkcijas, galima atlikti sudėties ir daugybos operacijas tarp dviejų *Valiuta* tipo reikšmių. Funkcijų apibrėžimuose nurodyta, kad jų reikšmės apskaičiuojamos panaudojant bazines operacijas, vartojamas su bazinio tipo `DECIMAL` reikšmėmis. Kadangi kvadratiniai piniginiai vienetai (pvz., litai × litai) yra neprasmingi, todėl daugybos operaciją apibrėžėme piniginių vienetų daugyba iš bedimensinio koeficiento. Naujai apibrėžtas sudėties ir daugybos operacijas, kaip ir standartines, galima rašyti tiek prefiksine, tiek ir sufiksine formomis.

Panašiai galime apibrėžti ne tik skaliarines, bet ir agregatines (stulpelių) funkcijas, pvz.:

```
CREATE FUNCTION Sum_Valiuta(Valiuta)
   RETURNS Valiuta SOURCE SUM(DECIMAL(15, 2)).
```

Naująsias funkcijas apibrėžėme per standartines funkcijas, praktiškai, tik papildydami standartinių funkcijų apibrėžimo sritį naujų tipų reikšmėmis. Naujųjų funkcijų reikšmės gali būti apskaičiuojamos SQL sakiniais. Apibrėžkime, bazinės valiutos konvertavimo į litus funkciją:

```
CREATE FUNCTION Valiuta_Litai(X Valiuta) RETURNS Litai
   LANGUAGE SQL
   CONTAINS SQL
   RETURN X * 4.0 ,
```

čia *X* yra formalus funkcijos parametras, fraze `LANGUAGE` nurodoma, kad funkcija yra realizuojama SQL reiškiniu, frazėje `RETURN` pateikiamas SQL reiškinys funkcijos reikšmei apskaičiuoti, fraze `CONTAINS SQL` nurodoma, kad SQL reiškinyje nėra vykdoma jokia užklausa. Vietoje `CONTAINS SQL` nurodžius `READS SQL DATA`, SQL reiškinyje galima naudoti ne tik konstantas ir formalius apibrėžiamos funkcijos parametrus, bet ir SQL sakinius. Apibrėžkime skaliarinę funkciją, kuria galima būtų apskaičiuoti, kiek konkretus vykdytojas skiria valandų visiems projektams vykdyti:

```
CREATE FUNCTION Sum_Valandos(Nr INTEGER) RETURNS INTEGER
   LANGUAGE SQL
   NOT DETERMINISTIC
   NO EXTERNAL ACTION
   READS SQL DATA
   RETURN (SELECT SUM(Valandos) FROM Vykdymas WHERE Vykdytojas = Nr),
```

čia fraze `NOT DETERMINISTIC` nurodoma, kad apskaičiuojant funkcijos reikšmę tai pačiai argumento reikšmei galima gauti skirtingus rezultatus, fraze `NO EXTERNAL ACTION` pranešama sistemai, kad vykdant funkciją nepasikeis jokio išorinio objekto, kurio nevaldo DBVS, pvz., failo būsena. Alternatyva frazei `NOT DETERMINISTIC` yra `DETERMINISTIC`, o frazei `NO EXTERNAL ACTION` - `EXTERNAL ACTION`. Tiek frazė

DETERMINISTIC, tiek ir NO EXTERNAL ACTION, leidžia DBVS optimizuoti pakartotinų kreipinių į funkciją atlikimą.

Apibrėžta skaliarinė funkcija *Sum_Valandos* gali labai sutrumpinti kai kuriuos SQL sakinius, pvz.,

SELECT *Pavardė*, *Sum_Valandos*(*Nr*) FROM *Tiekėjai.*

Priminsime, kad užklausos SELECT frazėje esantys reiškiniai skaičiuojami kiekvienai lentelės eilutei. Šioje užklausoje nėra dalinių užklausų. Tačiau, jei prisiminsime, kaip skaičiuojama funkcijos *Sum_Valandos* reikšmė, tai pastebėsime, kad pastaroji užklausa struktūriškai labai panaši į užklausą su koreliuota daline užklausa. Dėl panašių priežasčių negalima piktnaudžiauti tokiomis funkcijomis. Jų vartojimas gali būti paslėpta užklausų neefektyvumo priežastimi.

SQL nėra vienintelė kalba, kuria galima apibrėžti funkcijos reikšmę. Daugelis komercinių DBVS leidžia apibrėžti išorines funkcijas. **Išorine** vadinama **funkcija**, kuri yra realizuota kuria nors bazine programavimo kalba, jos vykdomasis (mašininis) kodas yra dinaminio ryšio bibliotekoje (angl. *Dynamic Link Library*), o duomenų bazėje yra saugomas tik funkcijos sąsajos apibrėžimas. Daugelis DBVS leidžia kreiptis į išorines funkcijas, sudarytas C ir JAVA programavimo kalbomis.

Tarkime, turime litų konvertavimo į valiutą funkciją, realizuotą ne SQL reiškiniu, bet C programavimo kalba. Tuomet mums reikia pranešti duomenų bazei apie funkcijos egzistavimą ir jos sąsają:

```
CREATE FUNCTION Litai_Valiuta(Litai) RETURNS Valiuta
EXTERNAL NAME 'konvertavimas ! litai_valiuta'
   LANGUAGE C
   PARAMETER STYLE DB2SQL
   DETERMINISTIC
   NO SQL
   NO EXTERNAL ACTION,
```

čia fraze EXTERNAL NAME nurodyta bibliotekos vardas (*konvertavimas*) ir joje esančios funkcijos vardas (*litai_ valiuta*), LANGUAGE - bazinė programavimo kalba (C) nuo kurios gali, pvz., priklausyti parametrų perdavimo tvarka, kreipiantis į funkcijos mašininį kodą; PARAMETER STYLE - funkcijos argumentų tipų atitikimo stilius (DB2SQL - funkcijai perduodami ir grąžinami parametrai tenkina DB2 SQL tipų atitikimo programoje reikalavimus, pvz., data vaizduojama dešimties simbolių eilute); NO SQL - funkcijos kūne nėra SQL sakinių (alternatyva – CONTAINS SQL).

DBVS, pasiruošdama vykdyti užklausą su kreipiniu į išorinę funkciją, pvz.,

```
SELECT   Pavadinimas,
         Vertė_ Valiuta + Litai_Valiuta(Vertė_Litais) AS "Bendra vertė valiuta"
FROM Projektai,
```

atlieka sintaksinę sakinio analizę pasitelkusi funkcijos sąsajos apibrėžimą. Pradedant vykdyti užklausos vykdomąjį planą, DBVS kreipiasi į operacijų sistemą "prašydama" įkelti į operatyviąją atmintį reikiamą biblioteką ir kiekvieną kartą, skaičiuodama reiškinio reikšmę, vykdo išorinės funkcijos mašininį kodą.

Išorinių funkcijų sudarymo technika yra labiau susijusi su programavimo technika nei su DB vartojimu, todėl detaliau jos nenagrinėsime.