9. SQL – relatsioonilise andmebaasi käsitluskeel: andmekirjelduskeel (DDL) ja andmetega manipuleerimise keel (DML)

SQL-keel

SQL (Structured Query Language) see on keel andmestruktuuride loomiseks ja kirjeldamiseks, andmete käsitlemiseks, andmete pärimiseks, andmete haldamiseks ja kasutajate õiguste haldamiseks relatsioonilistes andmebaasisüsteemides *(relational database system)*:

SQL keele funktsioonid

Kirjutame selle kõik nüüd natuke täpsemalt lahti:

andmestruktuuride loomine ja kirjeldamine	andmebaaside loomine ja hävitamine; tabelite kirjeldamine, tabeli kirjelduste muutmine ja tabelite hävitamine, tabelite vaheliste seoste loomine ja katkestamine; andmeid käsitlevate protseduuride loomine, muutmine ja hävitamine jms.
andmete käsitlemine	andemete lisamine, andmete uuendamine ja kustutamine andmebaasi tabelites
andmete pärimine	erineva raskusastmega andmete otsimine andmebaasist ja otsingu tulemuste esitamine
andete haldamine	andmebaaside varunduskoopiate tegemine, andmete taastamine varunduskoopiatest jms.
kasutajate ja kasutajaõiguste haldamine	uute kasutajate loomine, kasutajatele õiguste andmine ja neilt õiguste võtmine, kasutajate blokeerimine ja aktiveerimine, kasutajate kustutamine jms.

SQL on üks andmebaasi käsitlemise keeltest

SQL keel on üks andmebaaside käsitlemise keeltest. Kuid samas on see kõige levinum andmebaaside käsitlemise keeltest. SQL keel on kõikide tuntumate andmebaasisüsteemide koostisosaks. Kui mingil andmebaasisüsteemil on ka mingi SQL-keelest erineva käsitluskeel või -liides siis tavaliselt on selle kõrval olemas ka SQL-keel.

igal andmebaasisüsteemil on oma SQL-keel

SQL ei ole siiski midagi sellist, mis oleks kõikide andmebaasisüsteemide puhul sama sugune - igal andmebaasisüsteemil on oma spetsiifiline SQL-keel. On olemas küll erinevad SQL keele standardid, kuid iga andmebaasi juhtimissüsteemi SQL-keel on siiski erinev, kui võrrelda seda teiste andmebaasisüsteemidega. Milleks siis standardid? Asi on selles, et iga andmebaasisüsteemi SQL keel sisaldab endas ka standardit, kuid lisaks sellele omavad hulga laiendusi, mis on ainuomane just

sellele konkreetsele andmebaasisüsteemile. Nii siis kõigi andmebaasisüsteemide SQL-keeled vastavad mõnele standardile, kuid omavad suure hulga laiendusi, mis teevad nad üksteisest erinevaks.

standardne SQL-keel vs. konkreetse andmebaasisüsteemi SQL-keel

Kui nüüd kirjutada programmi mingi konkreetse andmebaasi süsteemi SQL-keelt arvestades, siis see programm ongi suuteline töötama ainult just selles andmebaasisüsteemis loodud andmebaasidega. Samas võime me loobuda nende laienduste kasutamisest ja kirjutada programmi kasutades ainult standardile vastavaid keelekonstruktsioone. Sellisel juhul saavutame me andmebaasi sõltumatuse kasutatavast andmebaasisüsteemist. Samas kaotame me võimaluse kasutada selle andmebaasisüsteemi spetsiifilise ja tavaliselt väga mugavaid ning otstarbekaid võimalusi.

konkreetse andmebaasisüsteemi SQL-keel on tihedalt seotus selle andmebaasisüsteemi omadustega

Kuidas siis käituda? Mille järgi valida andmebaasi süsteemi? Milliseid SQL-keele konstruktsioone kasutada programmeerimisel - kas standardset SQL-keelt või konkreetse andmebaasisüsteemi poolt pakutud laiendusega SQL-keelt? Erinevate andmebaasisüsteemide tootjate andmebaasisüsteemidel on mitmeid omadusi, millega nad püüavad erineda oma konkurentidest. Need erinevused on just need, mille abil püütakse saavutada konkurentsieelist. Üheks selliseks konkurentsieelise taotlemise erisuseks ongi ka SQL-keel. SQL-keel on väga tihedasti seotud andmebaasisüsteemi muude omadustega. Kui andmebaasisüsteemil on mingeid erilisi omadusi, siis peavad sisalduma tema SQL-keeles ka konstruktsioonid nende omaduste efektiivseks kasutamiseks. Lisaks sellele on erinevad tootjad teinud arendusi ka SQL-keele sellistele konstruktsioonidele. See millised on andmebaasi omadused ja seega ka SQL-keel määravad tihti ka andmebaasisüsteemi kasutusvaldkonna ja turusegmendi.

"millist" SQL-keelt kasutada

Kui hakata lahendatava ülesande jaoks valima andmebaasisüsteemi, siis tuleb seda teha mitte SQL-keele järgi vaid selle järgi, kas selle andmebaasisüsteemi omadused rahuldavad lahendatava ülesande vajadust. Kas töödeldavate andmemahtude suurus on piisav? Kas andmebaasi poole pöördumise meetodid on infrastruktuuri jaoks sobivad? Kas võimalik suurim kasutajate arv on piisav? Kas on võimalik luua selliseid andmestruktuure magu meil on vaja? Kas andmekaitse tase on piisav? Kas kasutajate õiguste kirjeldamise reeglid on piisavad? jne. jne. Kui mingi konkreetse andmebaasisüsteemi puhul suudate vastata nendest küsimustest rohkemale jaatavalt, kui teiste andmebaasisüsteemide jaoks, siis see andmebaas on teile sobivaim. Nüüd ei ole mõtet enam diskuteerida, milliseid SQL-keele konstruktsioone kasutada - et kasutada neid omadusi, mille pärast te konkreetse

andmebaasisüsteemi valisite, siis ei jää teil enam midagi muud üle. kui kasutada kogu pakutavat SQL-keelt. Kuidas siis muidu kasutada kõiki neid omadusi, mille pärast te valisite just selle andmebaasisüsteemi.

ORACLE lihtsustatud SQL-keel

Käesolevas loengu materjalis on kasutatud näiteid ORACLE SQL-keelest. Loomulikult vaadeldakse siin ainult väga väikest alamhulka ORACLE 9i SQL-keele käskudest, sest rohkemaks ei piisaks meil lihtsalt meie käsutuses olevast ajast. Lisaks sellel olen lihtsustanud mitmeid käske, kuna eesmärk ei ole siin õpetada ORACLE SQL-keelt vaid anda ülevaade põhilistest SQL-käskudest. Lihtsustamise käigus on üritatud viia ORACLE käsud lähemale SQL89 standardile.

9.1. SQL-keele standardid

IBM ja SEQUEL-keel (1970)

SQL-keele esimene versioon töötati välja firma IBM ja Donald D. Chamberlin' and Raymond F. Boyce' poolt 1970 alguses. Selle keele nimi oli alguses SEQUEL. See keel töötati välja IBM andmebaasisüsteemi System R jaoks.

Esimene standard ANSI 1986 ja ISO 1987

Aastal 1986 standardiseeriti SQL keel Ameerika Rahvusliku Standardiseerimise Instituudi (ANSI) poolt. Sama tegi ka Rahvusvaheline Standardiseerimise Organisatsioon (ISO) aasta hiljem - aastal 1987.

Järgnevalt on toodud kokkuvõtlikult SQL-keele standardite järgnevuse ajalugu:

Aasta	Nimi	Sünonüümid	Kommentaar
1986	SQL-86	SQL-87	Esmaselt registreeritud/avaldatud ANSI poolt. 1987 ratifitseeritud ISO poolt.
1989	SQL-89	FIPS 127-1	Mõnede väikeste muudatustega, adopteeritud kui FIPS 127-1 (FIPS - Federal Information Processing Standards, mis on loodud USA Föderaalvalitsuse poolt kasutamiseks mitte-sõjalistes organisatsioonides ja nende lepingupartnerite juures)

SQL-86

SQL-89

1992	SQL-92	SQL2, FIPS 127-2	Põhjalike muudatuse järel; (ISO 9075), *Entry Level* SQL-92, adopteeritud kui FIPS 127-2.	**SQL-92**
1999	SQL:1999	SQL3, SQL-99	Lisatud regulaaravaldised, rekursiivsed päringud, trigerid, toetus protseduuridele, mitte-skalaarsed andmetüübid ja mõningad objektorienteeritud omadused.	**SQL:1999**
2003	SQL:2003		Lisatud XML-omadused, sequendce-d, automaatselt genereeritud väärtustega tabelite veerud (ka vaikimisi väärtused ja ID-veerud).	**SQL:2003**
2006	SQL:2006		ISO/IEC 9075-14:2006 kirjeldab reeglid, kuidas SQL-keelt kasutatakse koos XML-ga. Kirjeldab reeglid XML struktuuride importimiseks ja eksportimiseks SQL-baasides, XML- andmete andmebaasides manipuleerimiseks ning XML ja tavaliste SQL-formaadis andmete esitamiseks XML-kujul. Lisaks sellele kirjeldab vahendid mis võimaldavad aplikatsioonides integreerida SQL koodi ja XQuery kasutamise (XQuery on XML päringu keel mis töötati välja World Wide Web Consortium (W3C) poolt, et üheaegselt pöörduda tavaliste SQL-andmete ja XML dokumentide poole.	**SQL:2006**

9.2. Graafiline metakeel SQL-keele kirjeldamiseks

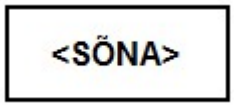
metakeele esitus graafilise regulaaravaldisena

Enne, kui alustada SQL-keele kirjeldamist, tuleb teha üks pisike kõrvale hüpe ja kirjeldada ära metakeel SQL-keelsete lausete süntaksi kirjeldamiseks. Tegemist on regulaaravaldistega, mis on kirjeldatud orienteeritud graafiga. Liikudes mööda graafi nooltega osutatud suundades ja kirjutades üles regulaaravaldisega määratud fraase saame luua vajalikud SQL-laused. Teiselt poolt saame me graafi järgi kontroolida, kas kirja pandud SQL-lause vastab teda kirjeldavale regulaaravaldisele.

Regulaaravaldiste kirjeldamise graafilise metakeele konstruktsioonid on järgmised:

literaal e. fikseeritud sõna

Igal SQL-keelt kirjeldaval avaldisel peab olema nimi. Sellega algavad kõik graafid (regulaaravaldised)



literaal e. fikseeritud sõna

fikseeritud sõna (literaal); kohtades, kus SQL-lause süntaksis kasutatakse sellist tähist, kirjutatakse raami sees olev sõna selle koha peale.

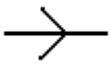
sümbol

fikseeritud sümbol - ainult üks märk, tavaliselt mitte täht või number vaid eraldussümbol (näiteks: ; , .), tehtemärk (näiteks: + - * /), erimärk (näiteks: @ $) või sulg; kohtades, kus SQL-lause süntaksis kasutatakse sellist tähist, kirjutatakse sõõri sees olev märk selle koha peale.

avaldis

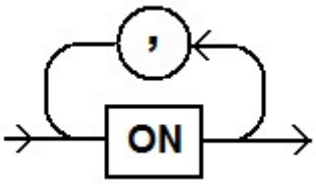
regulaaravaldis - ovaali sees on teise regulaaravaldise nimi, Selle koha peale asendatakse ovaal ovaali sees toodud nimelise regulaaravaldisega. Sisuliselt on siia ovaali sisse kirjutatud alam-avaldise nimi. Sellist konstruktsiooni kasutatakse kahel juhul. Esiteks siis, kui üks regulaaravaldis läheb liiga keeruliseks ja lihtsam on see jagada mõtestatud osadeks - erinevate nimedega alam-regulaaravaldisteks. Teisel juhul kasutatakse alam-avaldist siis, kui tegemist on palju kasutatava mõistega. Sellisel juhul

defineeritakse see mõiste regulaaravaldisena ja seal kus see mõiste esineb kasutatakse selle koha peal selle mõiste regulaar-avaldise nime. Vahel esitatakse alam-avaldise kirjeldus lihtsuse mõttes mitte regulaaravaldisena vaid semantilise kirjeldusena. Näiteks: "Eesti maakondlike linnade nimed", "Tallinna tänava nimed", "Eesti linnade tänavate nimed", "andmebaasi tabelite nimed", "andmebaasi tabeli veergude nimed" jne. Sellisel juhul esitatakse see kujul **<avaldise nimi> ::= <kirjeldus>**



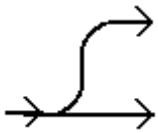
suunaja

suunaja - nool, mis seob eelnevat ja järgnevat regulaaravaldise komponenti kas "sõna", "sümbolit" või "avaldist" ja näitab ära järgnevuse suuna.



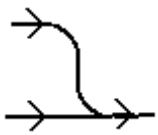
rekursioon

rekursioon - sellise konstruktsiooni puhul on võimalik tsükli (rekusiooni) lõpmatu kordumine; tegelikult kasutatakse kordumist "vastavalt vajadusele"; kui rekursioonide "vajalik" kordade arv "saab täis", siis lahkutakse rekursioonist väljuvat noolt pidi. Näiteks viie rekursiooni korral oleks joonisel toodud skeemi jaoks tulemus järgmine: ON,ON,ON,ON,ON (meil on tsükkel, mis algab sõnaga "ON", millele järgneb kas tsüklist väljumine või sümbol "," (koma) ja järgmine sõna "ON").



lahknemine

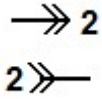
lahknemine - liikudes mööda graafi võime valida üks kõik millise tee (vastavalt vajadusele). Lahknemisi võib olla kaks või rohkem.



koondumine

koondumine - kaks või enam graafi haru koonduvad kokku ja jätkavad üheskoos sama teed pidi.

graafi jätkamine



graafi jätkaja - kui graaf ei mahu vasakult-paremale või ülalt-alla kasutatavale tööpinnale ära, siis on võimalik graafi jätkata katkestades graafi väljuva topelt-noolega ja jätkates suubuva topelt-noolega. Kuna jätkamisi võib olla palju, siis tähistatakse sama jätku topelt-nooled samade numbritega. Igal jätku paaril peab olema unikaalne number.

tühi sõlm

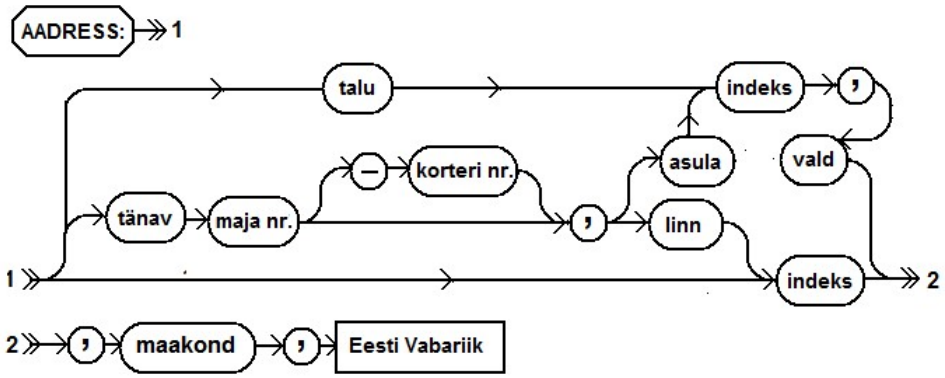
tühi sõlm, mida kasutatakse tavaliselt süntaksi graafi viimase komponendina selleks, et koondada kokku koonduvad teed. Seda kasutatakse juhul, kui graafi "lõpus" ei ole ühtegi teist sõlme, millesse koondada kokku koonduvad teed.

NÄIDE:

NÄIDE

Eesti vabariigi aadressi struktuuri kirjeldav graaf

Esitame nüüd Eesti aadressi struktuuri graafilise regulaaravaldisena:



talu ::= talu nimi

tänav ::= tänava nimi

maja ::= maja number

korteri ::= korteri number

linn ::= maakondliku linna nimi

asula ::= küla või aleviku nimi

indeks ::= postiindeks

maakond ::= maakonna nimi

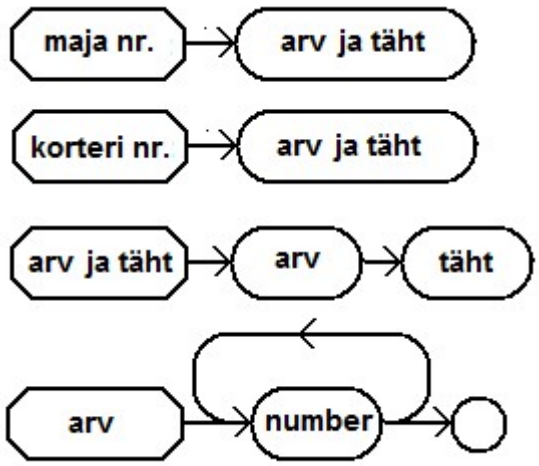
Konstrueerime nüüd selle skeemi järgi mõned aadressid:

Aadu talu 32443, Salme vald, Saaremaa, Eesti Vabariik

Õie 15-2, Tallinn 11615, Harjumaa, Eesti Vabariik

Tatra 25, Tartu 22322, Tartumaa, Eesti Vabariik

Esitasime kõik viited regulaaravaldistele, mis on skeemis toodud, semantiliste kommentaaridena. Võtame siiski kaks nendest ja kirjeldame need regulaaravaldisena. Võtame kaks väga sarnast mõistet: maja number ja korteri number, kuna need on struktuurilt samad.



täht ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | R | S | T | U | V

number ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Püstkriips (" | ") on siin kasutuses tähenduses "või"

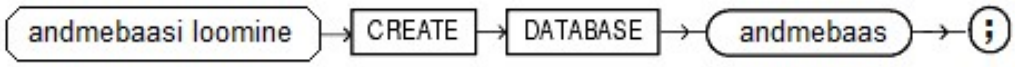
VÕI

9.3. SQL-keel

9.3.1 Andmebaasi loomine ja hävitamine

andmebaasi loomine

Andmebaasi loomise käsk võib, sõltuvalt kasutatavas andmebaasisüsteemist, olla väga keeruline. Osade andmebaaside puhul on andmebaasi loomine sedavõrd keeruline, et enamikus kasutatakse selleks spetsiaalset graafilist liidest (ORACLE, Sybase, SQLServer). Siiski on võimalik alati andmebaasi luua ka SQL keele käsuga. Kõige lihtsam ja lakoonilisem tühja andmebaasi loomise korraldus, kus kõik andmebaasi loomise parameetrid võetakse vaikimisi (vaikimisi asukoht, vaikimisi suurus jne.), on järgmine:



Nii lihtne see ongi: CRATE DATABASE TUDENG; (luua andmebaas TUDENG).

NÄIDE

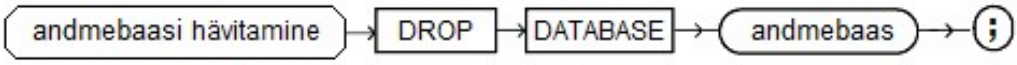
Andmebaasi nime maksimaalne lubatud pikkus sõltub sellest andmebaasisüsteemist, mille abil me seda andmebaasi loome. Samuti sõltub sellest ka nime kuju. Tavaliselt peab algama nimi tähega, koosneb ainult tähtedest ja numbritest ja ei tohi sisaldada täpitähti. Paljudel juhtudel on andmebaasi nime lubatud maksimaalseks pikkuseks 8 sümbolit. Kui nendest reeglitest lähtuda ei tohiks kunagi valesti minna.

andmebaasi kustutamine/ hävitamine

Asja korrektsuse mõttes lisame siia andmebaasi nime definitsiooni:

andmebaas ::= andmebaasi nimi; sõna, mis algab tähega, koosneb ainult numbritest ja ladina tähestiku tähtedest ja ei ole pikem kui 8 märki

Andmebaasi hävitamine on väga lihtne tegevus, mis võib tekitada probleeme pikemaks ajaks. Andmebaasi hävitamise käsu süntaks on järgmine:



Andmebaasi nime definitsioon on sama, mis andmebaasi loomise korralgi. Kuna me seal juba defineerisime selle mõiste, siis siin ei ole meil vaja seda enam korrata.

Andmebaasi hävitamine on "sama lihtne" kui tühja andmebaasi loomine: DROP DATABASE TUDENG;

NÄIDE

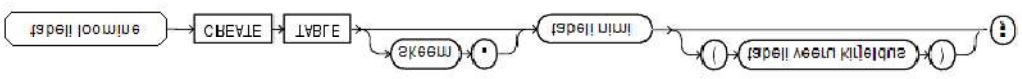
Enne selle korralduse andmist peaks hoolega mõtlema, kas andmebaasist ikka varunduskoopia on viimase seisuga tehtud, sest kellegi võib kunagi olla vaja andmeid just sellest kustutatavast baasist.

enne mõtle kui hävitad andmebaasi

9.3.2. Tabeli loomine ja hävitamine

Andmebaasi tabeli loomise lause SQL-keele süntaks on järgmine:

tabeli loomine



skeem ::= selle kasutaja nimi, kelle skeemi tabel luuakse

skeem *(schema)*

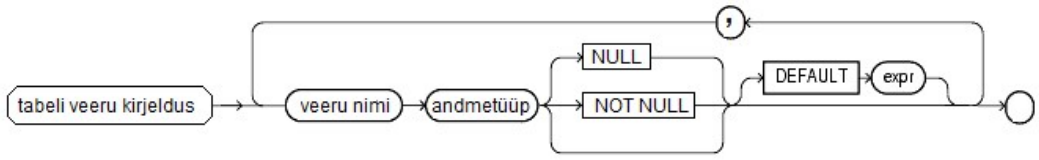
Siin skeem on selle kasutaja nimi kelle jaoks tabel luuakse - igal andmebaasi kasutajal on oma alamskeem, mille nimi langeb kokku tema kasutajanimega. Seda on vaja kasutada ainult siis, kui tabelit luuakse kellegi teise skeemi. Seda saab muidugi teha ainult vastavate õigustega administraator. Kui kasutajal on õigus luua enda jaoks ise tabeleid, siis oma kasutajanime kirjutada pole vaja. Kui see kirjutamata jätta, siis luuakse tabel vaikimisi tabeli loonud kasutaja skeemi.

tabeli nimi ::= loodava tabeli nimi, mille kuju sõltub kasutatavast andmebaasisüsteemist.

tabeli nimi

Tavaliselt peab tabeli nimi algama tähega, sisaldama ainult ladina tähestiku tähti, numbreid ja all-kriipse (" _ ") ja lõppema kas numbri või tähega. All-kriipse kasutatakse tühiku asendajatena. Tabeli nime maksimaalne lubatud pikkus sõltub konkreetset andmebaasisüsteemist ja on tavaliselt 32 sümbolit. Aga on ka andmebaasisüsteeme, kus tabeli nimi võib olla nii lühem kui ka pikem ja kus on nimes lubatud ka täpi-tähed ja tühikud.

tabeli veeru kirjeldus



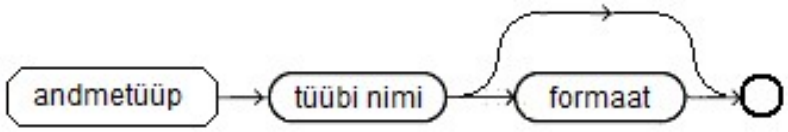
veeru nimi ::= andmebaasi tabeli veeru nimi

tabeli veeru nimi

Tavaliselt peab tabeli veeru nimi algama tähega, sisaldama ainult ladina tähestiku tähti, numbreid ja all-kriipse (" _ ") ja lõppema kas numbri või tähega. All-kriipse kasutatakse tühiku asendajatena. Tabeli veeru nime maksimaalne lubatud pikkus sõltub konkreetset andmebaasisüsteemist ja n tavaliselt 32 sümbolit. Aga on ka andmebaasisüsteeme, kus veeru nimi võib olla nii lühem kui ka pikem ja kus on nimes lubatud ka täpi-tähed ja tühikud.

Andmetüüp koosneb tüübi nimest ja formaadist:

andmetüüp



tüübi nimi ::= CHAR | VARCHAR | VARCHAR2 | LONG VARCHAR | DECIMAL | DATE | TIMESTAMP | DATETIME | INTEGER | SMALLINT | TINYINT | BLOB | PICTURE | NUMERIC | REAL | LONG

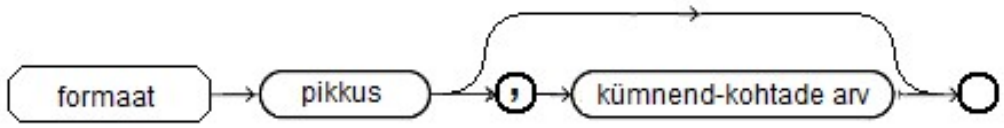
valik andmetüüpe

See andmetüüpide loend ei ole sugugi lõplik. Erinevatel andmebaasisüsteemidel on see loend vägagi erinev. Siin tõin sellise loendi, mis ei kuulu konkreetselt ühelegi andmebaasisüsteemile vaid on mix enam levinud andmetüüpidest.

NULL ja NOT NULL

Fraasid NULL ja NOT NULL määravad selle, kas antud veeru väärtus tohib olla kirjes tühi (NULL) või mitte (NOT NULL). Kui kumbagi fraasi ei kirjutata, siis on vaikimisi NULL.

pikkus ja kümnendkohtade arv



pikkus ::= arv mis on suurem kui 0 ja väiksem kui 255 (0 < n <=254). Arvude puhul määrab arvu maksimaalse pikkuse ilma + / - märgita ja kümnend kohti eraldava komata/punktita (numbrikohtade üldarvu).

kümnendkohtade arv ::= arv, mis on suurem võrdne kui 0 ja väiksem võrdne kui X (0<= n <= X). Seejuures X on konkreetse andmebaasisüsteemi poolt määratud arvu täpsus. Tavaliselt jääb see sinna 15 kuni 25 piiresse. See määrab arvu kümnendkohtade arvu pärast koma/punkti.

Igal andmetüübile vastab oma formaat. Formaadiks on kas lihtsalt andmevälja pikkus vaõi siis andmevälja pikkus ja (üle koma= kümnendkohtade arv. See viimane on ainult murdarvudel. Ülejäänud andmetüüpidel on formaadiks kas ainult pikkus või pole neil formaati üldse.

Kirjutame tabeli loomise lause tabel ISIK loomiseks:

NÄIDE

```
CREATE TABLE ISIK
   ( isik_id            NUMBER(6),
     eesnimi            VARCHAR2(20),
     perskonnanimi      VARCHAR2(25) NOT NULL,
     isikukood          CHAR(11),
     e_mail             VARCHAR2(25) NOT NULL,
     telefoni_nr        VARCHAR2(20),
     synnipaev          DATE  NOT NULL,
     aadress            VARCHAR2 (100),
     reg_kuup           DATE DEFAULT SYSDATE
   );
```

Lõpetuseks annan veel lühikesed selgitused andmetüüpide kohta:

Andmetüübid

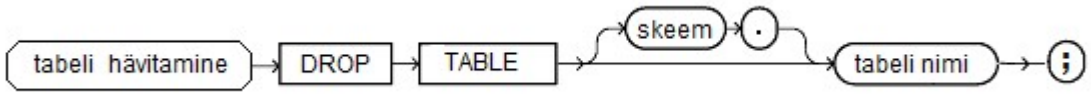
CHAR	CHAR (pikkus)	fikseeritud pikkusega tekst. Veerg reserveerib andmebaasis alati just nii palju baite, kui on märgitud tema pikkuseks, seda isegi siis, kui väli on tühi. Veergu ei saa salvestada pikemat	**andmetpüüp CHAR**

		teksti kui on määratud pikkus. N: CHAR(25)	
VARCHAR ja VARCHAR2	VARCHAR(pikkus) VARCHAR2(pikkus)	muutuva pikkusega tekst, mis ei saa olla pikem kui määratud pikkus. Veerg reserveerib nii palju ruumi, kui temasse reaalselt on sümboleid talletatud. Kui veerg on tühi, siis ei reserveeri ta mälu üldse. VARCHAR2 on puhtalt ORACLE andmetüüp. Oracle's on olemas ka andmetüüp VARCHAR. Aga see on vana andmetüüp, mis on olemas ainult tagant järele ühilduvuse tarvis. Uute andmebaaside loomisel tuleneb kasutada andmetüüpi VARCHAR. Teistes andmebaasisüsteemides on kasutusel andmetüüp VARCHAR. N: VARCHAR(254), VARCHAR2(254)	**andmetüübid VARCHAR ja VARCHAR2**
LONG VARCHAR, BLOB, PICTURE ja BINARY	LONG VARCHAR BLOB PICTURE BINARY	Erinevates andmebaasisüsteemides on kasutusel erinevad võtmesõnad. LONG VARCHAR on olemas kõigis andmebaasisüsteemides. Tüüp tähistab lõpmatu pikkusega teksti. Tegelikult on küll iga konkreetse andmebaasisüsteemi puhul olemas mingi piir aga see on väga suur. Konkreetse andmebaasisüsteemiga töötamisel tuleb see lihtsalt juhendist järele vaadata. Seda tüüpi veergu saab salvestada ka pilte, helisid , Word'i dokumente, Exceli tabeleid, filme jms.	**andmetüübid LONG VARCHAR, BLOB, PICTURE ja BINARY**
DATE	DATE	kuupäev, mis sisaldab nelja kohalist aastaarvu, kuud ja päeva. Väljastusformaat sõltub keskkonnast - näiteks Windows' all kasutatakse tavaliselt Windows'is määratud kuupäeva lühikest või pikka formaati. erinevad programmid võivad kuupäeva formaatida erinevalt.	**andmetüüp DATE**
DATETIME ja TIMESTAMP	DATETIME TIMESTAMP	Kuupäev ja kellaaeg. Kuupäev esitatakse nagu DATE formaadi puhulgi, kellaaeg esitatakse tavaliselt sekundi täpsusega.	**andmetüübid DATETIME ja TIMESTAMP**
DECIMAL	DECIMAL(pikkus, kümnendkohti)	Fikseeritud komakohtade arvuga arv. N: DECIMAL(10,3) formaati mahub maksimaalselt arv 9999999.999 N: DECIMAL(10,0) formaati mahub maksimaalselt arv 9999999999 (kümnendkohti ei ole)	**andmetüüp DECIMAL**
NUMERIC	NUMERIC(pikkus) NUMERIC(pikkus, kümnendkohti)	Sama, mis DECIMAL, aga natuke erinev süntaks aj andmebaasisisene salvestus-formaat	**andmetüüp NUMERIC**
TINYINT, SMALLINT ja INTEGER	TINYINT SMALLINT INTEGER	2-, 4-, 8-baidised või 1-, 2- ja 4-baidised täisarvud. See kui pikad tegelikult nad on sõltub konkreetsest andmebaasisüsteemist. Mõni nendest võib konkreetse andmebaasisüsteemi puhul ka puududa.	**andmetüübid TINYINT, SMALLINT ja INTEGER**

		Kõikides baasides on igal juhul olemas tüüp INTEGER .	
REAL ja LONG	REAL LONG	4- ja 8-baidise pikkusega ujupunkt-arvud. Ei kasutata majandusarvutustes vaid ainult teadusarvutustes	**andmetüübid REAL ja LONG**

tabeli hävitamine

Tabeli hävitamise lause süntaks on äärmiselt lihtne:



Kui hävitada enda skeemis olevaid tabeleid, siis pole skeemi nime lausesse vaja kirjutada. Kui hävitada tabeleid mingi teise kasutaja skeemist, siis tuleb skeemi nime kohale kirjutada selle kasutaja nimi. Teise kasutaja skeemist saab tabeleid hävitada ainult administraatori õigustes kasutaja. Pärast tabeli kustutamist ei ole seda enam võimalik taastada muidu kui ainult varunduskoopiast (kui selline peaks olemas olema). Seepärast tasub enne tabeli hävitamist tõsisemalt kaaluda, ega mingid olulised andmed koos selle tabeliga kaduma ei lähe.

NÄIDE

Näide: DROP TABLE ISIK;

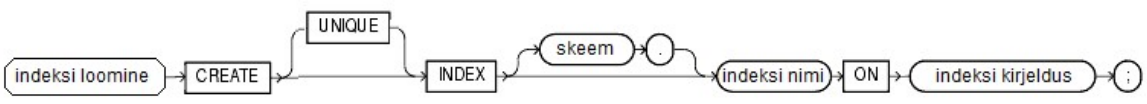
9.3.3. Indeksi loomine ja hävitamine

Indeksite täpne kirjeldus on 14. loengu materjalides

Indeks on tabelitest andmete otsimise kiirendi, mille toimimisest räägitakse täpsemalt loengus 14. Siin toome vaid süsteemi terviklikkuse mõttes indeksite loomise ja hävitamise SQL-korralduste süntaksi. Selle jaotise võib praegu vahele jätta või siis lugeda enne käesoleva jaotise lugemist ettehaaravalt läbi 14. loengu materjalid. Pärast seda on siin jaotises kirjutatu igati mõistetav.

indeksi loomise korralduse süntaks

Indeksi loomise SQL-keelse korralduse süntaksi graaf on järgmine:



indeksi nimi

Kus skeemi tähendus on sama mis jaotises 9.3.1. tabeli loomise ja hävitamise korralduses. Indeksi nimi on nimi, mille all loodav indeks andmebaasi salvestatakse ja mille alusel on indeksit võimalik andmebaasist hävitada.

indeksi nimi ::= loodava indeksi nimi, mille kuju sõltub kasutatavast andmebaasisüsteemist.

Tavaliselt peab indeksi nimi algama tähega, sisaldama ainult ladina tähestiku tähti, numbreid ja all-kriipse (" _ ") ja lõppema kas numbri või tähega. All-kriipse kasutatakse tühiku asendajatena. Indeksi nime maksimaalne lubatud pikkus sõltub konkreetset andmebaasisüsteemist ja on tavaliselt 32 sümbolit. Aga on ka andmebaasisüsteeme, kus indeksi nimi võib olla nii lühem kui ka pikem ja kus on nimes lubatud ka täpi-tähed ja tühikud.

unikaalsed indeksid

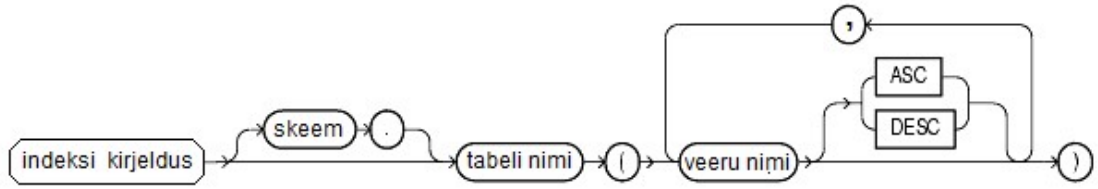
Kui indeksi loomise lausesse on lisatud fraas UNIQUE, siis pärast indeksi loomist ei ole võimalik indeksis kirjeldatud tabeli väljade komplekti sisestada selliseid väärtusi, et see väljade komplekt omaks mitmes sama tabeli kirjes samu väärtusi.

Sisuliselt tähendab skeemil toodud fraas:

LUUA (UNIKAALNE) INDEKS indeksi nimi (näiteks: ISIKU_NIMI) VÕTTES ALUSEKS indeksi kirjelduse.

indeksi kirjeldus

See indeksi kirjeldus, mis aluseks tuleb võtta, kirjeldatakse järgmise süntaksiga:



Indeks luuakse alati tabelile. See tabel kirjeldatakse nagu ikka määrates skeemi nime (kus tabel asub / kellele tabel kuulub) ja tabeli nime. Kui luuakse indeks tabelile, mis asub selle indeksi looja skeemis, siis skeemi nime pole vaja lisada. Indeksi kirjeldus on tabeli veeru nimede loend üle koma. Iga nime järele saab veel kirjutada fraasi ASC või DESC (ascending või descending), mis määrab, kas veeru väärtused indeksis sorteeritakse kasvavas (ASC)

või kahanevas (DESC) järjestuses. Kui seda fraasi pole määratud on vaikimisi määratud ASC.

veeru nimed samast tabelist

Veeru nimi on juba eelnevalt defineeritud. Siin tuleb lisada vaid seda, et veeru nimedena saavad indeksis olla kasutusel selle sama tabeli veergude nimed, millele me indeksit loome (mille skeemi ja nime me kirjeldasime selles samas lauses).

NÄIDE

Näide: CREATE INDEX .REIN.ISIK_NIMED ON ISIK (PEREKONNANIMI ASC, EESNIMI ASC);

indeksi hävitamine

Indeksi hävitamine ei too mingit pöördumatut kahju - indeks on võimalik ühe SQL-lause serverisse saatmisega uuesti luua. Tõsi küll, kui tabelis, millele indeksit luuakse on väga palju kirjeid võib indeksi loomine oluliselt serverit koormata. Samuti võivad mõned päringud oluliselt aeglasemaks muutuda. Indeksi hävitamise korralduse süntaks on järgmine.

Ega siin midagi pikemalt vist seletada pole - kõik peaks ise enesest selge olema. Kindluse mõttes toon siin ka näite.

Näide:

NÄIDE

DROP INDEX REIN.ISIK_NIMED;

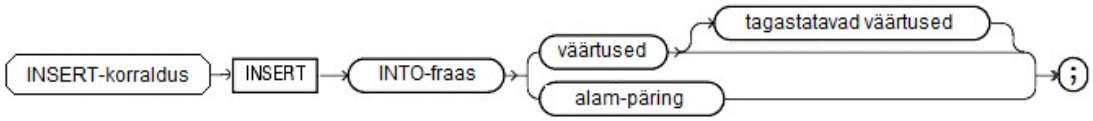
9.3.4. Andmete lisamine tabelisse

kirjete lisamine tabelisse ühe ja mitme kaupa

Andmete lisamine (kirjete lisamine) andmebaasi toimub tabelite kaupa ja üldjuhul kirje haaval. On võimalikud ka sellised keele konstruktsioonid, kus kirjeid lisatakse mitme kaupa korraga, kuid sellise korralduse süntaks on erinevates andmebaasisüsteemides erinev ja seepärast sellel siin ei peatuta.

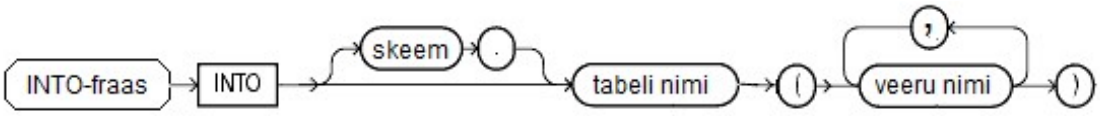
INSERT - kirje lisamine tabelisse

Kirje tabelisse lisamise korralduse SQL-keelne süntaks on järgmine:



INTO-fraas

Into fraas kirjeldab, millisesse tabelisse kirje lisatakse ja millistesse tabeli veergudesse väärtused kirjutatakse:

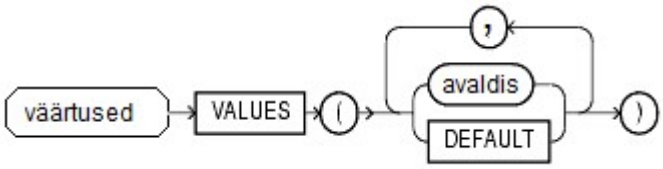


INSERT-korraldus ja NOT NULL veerud

Skeemi ja tabeli nime järgi määratakse tabel kuhu kirje lisatakse (ei pea vist enam mainima, et skeemi nimi võib ka puududa - see paistab skeemilt ilusasti välja kah) ja seejärel tulevad sulgudes üle koma nende tabeli veergude nimed, kuhu väärtused talletatakse. Siin tuleb silmas pidada seda, et INSERT lause INTO-fraasis peavad olema üles loetletud vähemalt kõik NOT NULL veerud - ilma nendesse veergudesse väärtust kirjutamata kirjet salvestada ei saa. Andmebaasisüsteem lihtsalt takistab seda "jõuga", milleks on veateade "Not enough NOT NULL values" või midagi sellist.

VALUES - fraas (väärtused)

Väärtused grupis kirjeldatakse samas järjekorras veergudega ära nendesse veergudesse talletatavad väärtused:



avaldis

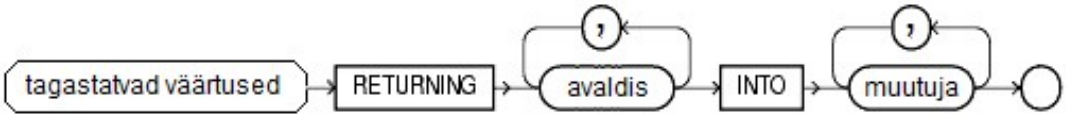
avaldis ::= avaldis on kas programmi muutuja nimi, millele eelneb " : " (koolon) või konstant (tekst veeru korral on tekst-konstant apostroofide vahel; kuupäev-tüüpi veeru korral on konstandi formaat tavaliselt KK/PP/AAAA; kuupäev-kellaaeg tüüpi veeru korral on konstandi formaat tavaliselt KK7PP/AAAA hh:mm:ss)

DEFAULT-väärtus

Kui avaldise asemel seisab väärtuse koha peal võtmesõna DEFAULT, siis pannakse veeru väärtuseks tabeli loomisel sellele veerule vaikimisi väärtuseks kirjeldatud väärtus.

tagastatavad väärtused

Pärast INSERT-korralduse täitmist on vahel vaja saada tagasi programmi, kust INSERT-korraldus käivitati, väärtusi, mis salvestati tabeli vast loodud kirjesse. Tekkib küsimus, kas me siis ei tea, mida me sinna salvestasime? Vahel on tõe poolest nii. Seda selle pärast, et andmebaasi kirjete kalla toimetavad ka andmebaasi protseduurid, mis kirje lisamise hetkel võivad sinna kirjutada väärtusi. JA neid väärtusi me tõe poolest ei tea. Näiteks võib olla andmebaasi kirjutatud protseduur, mis automaatselt väärtustab kirje ID- Kirje ID väärtuse teadmine pärast kirje lisamist baasi on väga oluline. Selle ID kaudu saame me lisatud kirjega siduda teistes tabelite sinna lisatavaid kirjeid. Kirje väärtuste tagastamist pärast kirje lisamist võimaldabki tagastatavate väärtuse kirjeldus:



Siin avaldiseks on tavaliselt andmebaasi tabeli veeru nimi (loomulikult sellest samast tabelist kuhu kirje lisati) ja muutuja koha peal seisab selle programmi muutuja nimi (mille ees on koolon), kust SQL-korraldus andmebaasi saadeti. Pärast INSERT-korralduse edukat täitmist saadetakse avaldiste väärtused samas positsioonis asuvasse muutujatesse. Ei tasu vist mainimist, et avaldiste loendis ja programmi muutujate loendis peab olema (üle koma) võrdne arv liikmeid. Seejuures peavad paari kaupa langema kokku ka andmetüübid.

alam-päringuga INSERT-koraldust vaatame hiljem - jaotises 10

Alam-päringu tähendust me siinkohal ei vaata, kuna meil ei ole läbi vaadatud SELECT-korralduse tähendus ja süntaks. Vaatame seda eraldi pärast SELECT-korralduse kirjeldamist jaotises 10.

NÄITED

Nüüd lõpetuseks mõned INSERT-korralduste paar näidet ka:

INSERT INTO employees (employee_id, last_name, email, hire_date, job_id, salary, commission_pct)
VALUES (207, 'Gregory', 'pgregory@oracle.com', sysdate, 'PU_CLERK', 1.2E3, NULL);

INSERT INTO employees (employee_id, last_name, email, hire_date, job_id, salary)
VALUES (employees_seq.nextval, 'Doe', 'john.doe@oracle.com', SYSDATE,

'SA_CLERK', 2400)
*RETURNING salary*12, job_id INTO :bnd1, :bnd2*

INSERT INTO employees (employee_id, last_name, email, hire_date, job_id, salary)
VALUES (:nEmpID, :sName, :sEmail, SYSDATE, :sJob, :nSalary)
*RETURNING salary*12, job_id INTO :bnd1, :bnd2*

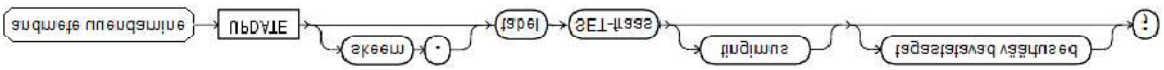
9.3.5. Andmete uuendamine

kirjete uuendamine ühe ja mitme kaupa

Andmete uuendamine (kirjete veergude väärtuste muutmine) toimub andmebaasis tabelite kaupa. Kirjeid on võimalik uuendada nii kirje haaval (ühe kirje kaupa) kui massina (mitmed kirjed sama korraldusega) aga ainult ühes tabelis korraga. See, mitu kirjet tabelis uuendatakse sõltub tingimusest, mille abil filtreeritakse kirjeid. Kui esitatud tingimusele vastab üks kirje, uuendatakse ainult selle kirje veergude väärtused, kui aga mitu kirjet, siis uuendatakse kõigi nende kirjete veergude väärtused. Kui esitatud tingimusele ei vasta ühtegi kirjet, siis korraldus küll täidetakse aga ühegi kirje veeru väärtust tabelis ei uuendata.

UPDATE - kirje(te) väärtuste muutmine

Kirje (kirjete) uuendamise korralduse SQL-keelne süntaks on järgmine:

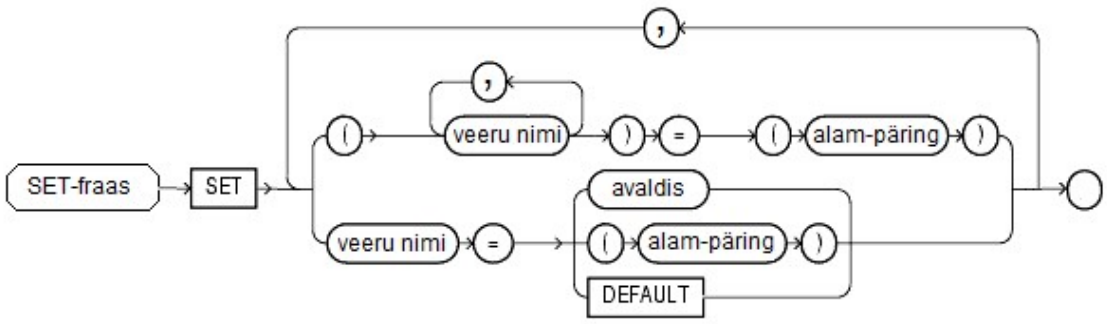


tabeli nime kirjeldamine

Tabeli, mille kirjet või kirjeid UPDATE-korraldusega muudetakse, määramiseks kasutatakse ikka seda sama skeemi ja tabeli nime kaudu tabelile viitamist. Täpselt samuti nagu eelnevalt kirjeldatud SQL-korraldustes.

SET-fraas

SET-fraas määrab tabeli veerud, mida uuendatakse ja väärtused millega seniseid veergude väärtusi uuendatakse. Tegemist on üle koma esitatud paaridega " veeru nimi = uus väärtus". Uus väärtus esitatakse avaldisena:



veergude nimed samast tabelist

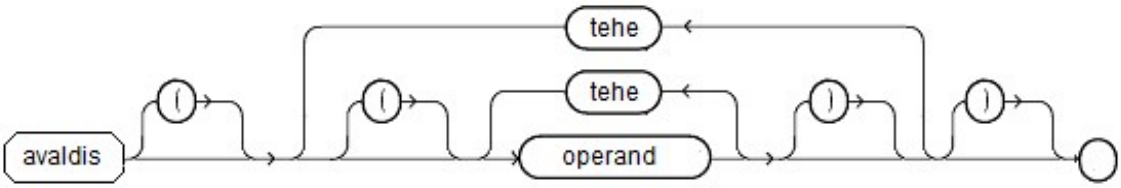
Veeru nimed valitakse selle tabeli veeru nimede hulgast, mille kirjeid UPDATE-korraldusega uuendatakse. Veeru nimi on meil varasemast defineeritud ja seega pole meil siin seda definitsiooni vaja korrata.

avaldis

Avaldise süntaksit Avaldis on "valem", mis koosneb tabeli veeru nimedest (selle sama uuendatava tabeli veeru nimedest), konstantidest, programmi muutujatest (selle programmi, mis UPDATE-korralduse andmebaasi saatis), funktsioonidest (andmebaasisüsteemis kirjeldatud süsteemsetest ja kasutaja poolt loodud funktsioonidest) ja matemaatilise valemi korral matemaatilistest tehtemärkidest või tekstilise tulemiga valemi korral teksti sidurdus-märkidest.

avaldise süntaks

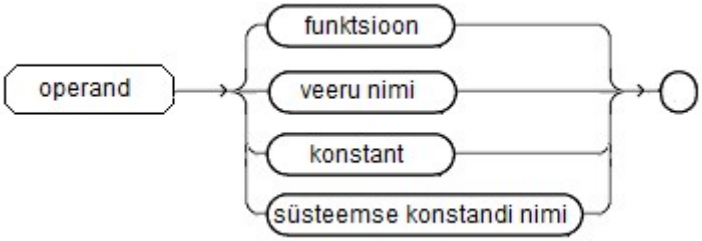
Esitame nüüd definitsiooni lõplikkuse pärast ka avaldise süntaksi graafid:



tehe

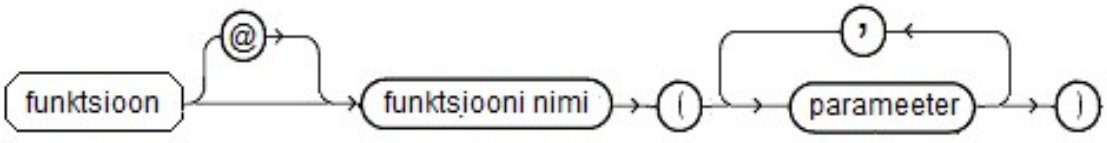
tehe ::= + | **-** | ***** | **/** | **||** , kus "**||**" on tekstide sidurdamise tehe ja "**^**" on astendamise tehe (2^3 - kaks astmes kolm)

operand



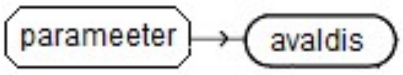
süsteemsed- ja kasutaja loodud funktsioonid

Funktsioon on siin funktsioon, mis on kirjeldatud andmebaasisüsteemis. See võib olla nii süsteemne funktsioon kui ka kasutaja poolt loodud funktsioon.



Enamiku süsteemsete funktsioonide nime ees on sümbol "@". Seda pole statistiliste funktsioonide nime ees ja kasutaja poolt kirjeldatud funktsioonide nime ees. Funktsiooni nime järel on üle koma funktsiooni parameetrid. Funktsiooni parameeter on avaldis.

funktsiooni parameetrid



Veeru nime tähendus on sama, mis eelmistes süntaksi graafideski.

konstant

Konstant on kas arv-, tekst-, tõeväärtus- (TRUE või FALSE), kuupäev või kuupäev/kellaeg-kontsant (Näiteks : 1.23 ; 'Tekst' ; TRUE ; 02/15/2008 (KK/PP/AAAA) ; 01/15/2008 12:05:23 (KK/PP/AAAA hh:mm:ss))

süsteemne konstant

Süsteemse konstandi nimi on andmebaasisüsteemis defineeritud konstandi nimi, näiteks SYSDATE (jooksev kuupäev).

alam-päringutest räägime SELECT-korralduse juures

Alam-päring on SELECT-lause ja kuna me pole veel SELECT-korralduse süntaksi kirjeldanud, siis vaatame selle osa UPDATE-lause süntaksist üle pärast SELECT-lause süntaksi käsitlemisest. Praegu jätame selle vahele. Seega jääb meil esitatud süntaksi skeemist praegu vaatluse alla ainult alumine haru ja sedagi ilma keskmise alamharuta, kus on kasutatud alam-päringut.

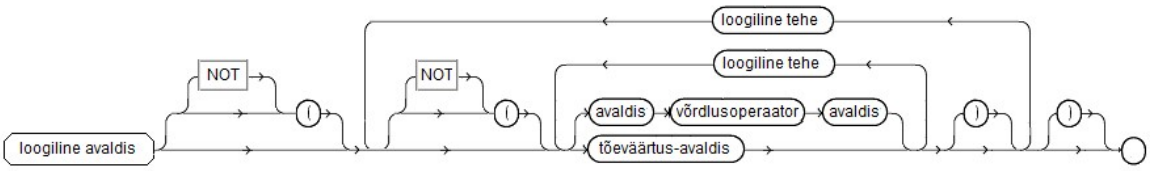
Seega on SET-fraas meie jaoks praegu jada üle koma järgnevatest alam-fraasidest, kus alam-fraasiks on kas "veeru nimi = avaldis" või "veeru nimi = DEFAULT",

Avaldiseks on enamikel juhtudel kas konstant, muutuja nimi (selle programmi muutuja, mis UPDATE-lause andmebaasisüsteemile saatis) või siis tehe selle sama tabeli veeruga. Kõige sagedamini kasutatavaks konstruktsiooniga tehtes on "veeru nimi = veeru nimi +1" või mingi muu tehe (*, / , -) ja mingi muu väärtusega konstant. Siiski on lihtsalt muutuja nimi ja lihtsalt konstant kõige sagedamini esinevad avaldised.

WHERE-tingimus

Tingimus on loogiline avaldis, mille alusel otsustatakse, milline kirje või millised kirjed tabelis uuendatakse. Loogiline avaldus koosneb alam-avaldistest, millest iga kirjeldab mingi tingimuse tabeli veergude suhtes. Tingimuse süntaks on järgmine

loogiline avaldis



loogiline tehe

l**ooghiline tehe ::= AND | OR | XOR**

Võrdlus-operaator

võrdlusoperaator ::= < | <= | = | IS | != | # | IS NOT | > | >= (!=, # ja IS NOT tähistavad kõik võrdlust "mitte võrdne")

loogiline avaldis

Loogiline avaldis on avaldis, mille tulemust on võimaik tõlgendada tõeväärtusena - TRUE või FALSE (Oracle's on kolme-valentne loogika kolmandaks tõeväärtuseks on UNDEFINED - määramata; selle käsitlemisest räägime eraldi).

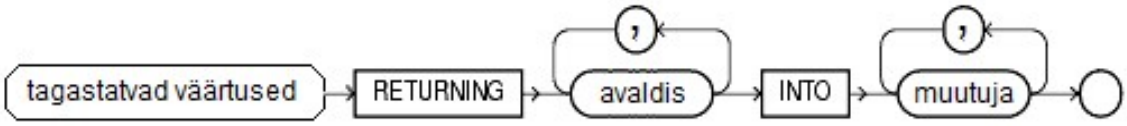
tõeväärtus-avaldis

Tõeväärtus-avaldis on tõeväärtuste (BOOLEAN-tüüpi andmebaasi väljad ja programmi muutujad) kombinatsioon üle loogiliste tehete (ei kasutata võrdlusoperaatoreid).

Avaldis on täiesti tavaline avaldis just sellises tähenduses nagu me ees pool selle defineerisime.

Tagas-tagtavad väärtused

Tagastatavate väärtuste kirjeldus on analoogiline INSERT-korralduse juures (eelmises jaotises) kirjeldatuga.



Ka tagastatavate väärtuste tähendus on sama - pärast UPDATE korralduse täitmist tagastatakse baasis olevad väärtused korralduse andmebaasi saatnud programmi muutujatesse. Seda on vaja põhjusel, et andmete uuendamise käigus võivad käivituda andmebaasi protseduurid, mis muudavad kirje veergude väärtusi ja meil on kohe vahetult pärast korralduse täitmist vaja neid väärtusi teada. Andmebaasisüsteemides, mille SQL-keeles see konstruktsioon puudub, tuleb pärast UPDATE-korralduse täitmist pärida muutunud andmed baasist välja SELECT-korraldusega, mis täidetakse vahetult pärast UPDATE-korraldust.

NÄITED

Lõpetuseks toon ka mõned näited UPDATE-lausete kohta:

```
UPDATE employees
   SET commission_pct = NULL
   WHERE job = 'SA_CLERK';

UPDATE employees
   SET  job_id = 'SA_MAN', salary = salary + 1000, department_id = 120
   WHERE first_name||' '|| last_name = 'Douglas Grant';

UPDATE employees
   SET job_id = :sTekst, salary = salary + 1000, department_id = :nDepID
   WHERE first_name||' '|| last_name = ':sFilter;


UPDATE accounts@boston
   SET balance = balance + 500
   WHERE acc_no = 5001;

UPDATE employees
  SET job_id ='SA_MAN', salary = salary + 1000, department_id = 140
  WHERE last_name = 'Jones'
```

```
RETURNING salary*0.25, last_name, department_id  INTO :bnd1, :bnd2, :bnd3
```

Vaatamata pikale ja suhteliselt keerukale süntaksi kirjeldusele on UPDATE-lause enamikus just nii lihtsad nagu siin näidetes on näha. SET-loend võib küll olla oluliselt pikem, kuid see ei saa olla kunagi pikem kui uuendatava tabeli veergude arv, kuna iga veerg võib selles loendis esineda ainult üks kord.

Ka tingimus on UPDATE lauses tavaliselt lihtne - "Primaty Key veeru nimi = :muutuja nimi või konstant" kus muutuja või konstant sisaldavad selle kirje primaarvõtme väärtust, mille veergude väärtusi soovitakse uuendada (üle kirjutada):

NÄIDE

```
UPDATE ISIK
SET PEREKONNANIMI=:sPerenimi
WHERE ISIK_ID=:nIsikID;
```

Kui jätta lausesse tingimus kirjutamata uuendatakse kõik tabeli kirjed:

NÄIDE

```
UPDATE ISIK
SET PEREKONNANIMI=:sPerenimi;
```

See lause kirjutab kõigile baasis registreeritud inimestele (kõikidesse kirjetesse) selle perekonna nime, mis on talletatud muutujasse "sPerenimi".

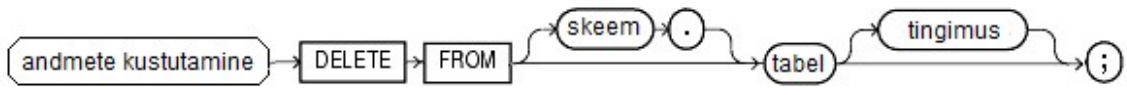
9.3.6. Andmete kustutamine

kirjete kustutamine ühe ja mitme kaupa

Andmete kustutamise (kirjete tabelist füüsilise kustutamise) andmebaasist toimub tabelite kaupa. Kirjeid on võimalik tabelist kustutada nii ükshaaval (ühe kirje kaupa) kui massina (mitmed kirjed sama korraldusega). Nii nagu andmete uuendamise korralgi sõltub see mitu kirjet kustutatakse tingimusest, mille abil filtreeritakse kirjeid. Kui esitatud tingimusele vastab üks kirje, kustutatakse ainult see kirje, kui aga mitu kirjet, siis kustutatakse mitu kirjet. Kui esitatud tingimusele ei vasta ühtegi kirjet, siis korraldus küll täidetakse aga ühtegi kirjet tabelist ei kustutata.

DELETE - kirje(te) kustutamine

Kirje (kirjete) kustutamise korralduse SQL-keelne süntaks on järgmine:



DELETE- ja UPDATE-lause sarnasus

Nagu te näete, pole siin enam midagi pikalt kirjeldada - kõik mõisted ja lause süntaksi komponendid on juba eelmistes jaotistes kirjeldatud. Tabeli nime kirjeldamine läbi skeemi nime ja tabeli nime oleme kasutanud korduvalt. Eelmises jaotises kirjeldasime ära ka tingimuse süntaksi - kirjete välja valimine kustutamiseks toimub täpselt samade reeglite alusel, nagu see toimub kirjete välja valimine kirjete uuendamiseks. Kahe laudse (UPDATE ja DELETE) on ju ainult selle, mida nende valitud kirjetega tehakse - ühel juhul muudetakse valimisse kuuluvate kirjete veergude väärtusi teisel juhul aga kustutatakse need kirjed.

NÄITED

Toome nüüd teema lõpetuseks veel mõned DELETE-lausete näited:

```
DELETE FROM employees
   WHERE job_id = 'PU_CLERK'  AND commission_pct < .1;

DELETE FROM employees
   WHERE expire_date>= :dDateBegin AND expire_date<=:dDateEnd;
```

DELETE-lause on väga lihtne. Enamikus kasutatakse kirje kustutamist kirje primaarvõtme järgi. Sellisel juhtumil on tingimus "primaarvõtme veeru nimi = :muutuja nimi või konstant" kus muutuja või konstant sisaldavad selle kirje primaarvõtme väärtust, mida tabelist kustutada soovitakse:

NÄIDE

```
DELETE FROM ISIK
WHERE ISIK_ID=:nIsikID;
```

Kui jätta lausesse tingimus kirjutamata kustutatakse kõik tabeli kirjed:

NÄIDE

```
DELETE FROM MARGUS.ISIK;
```