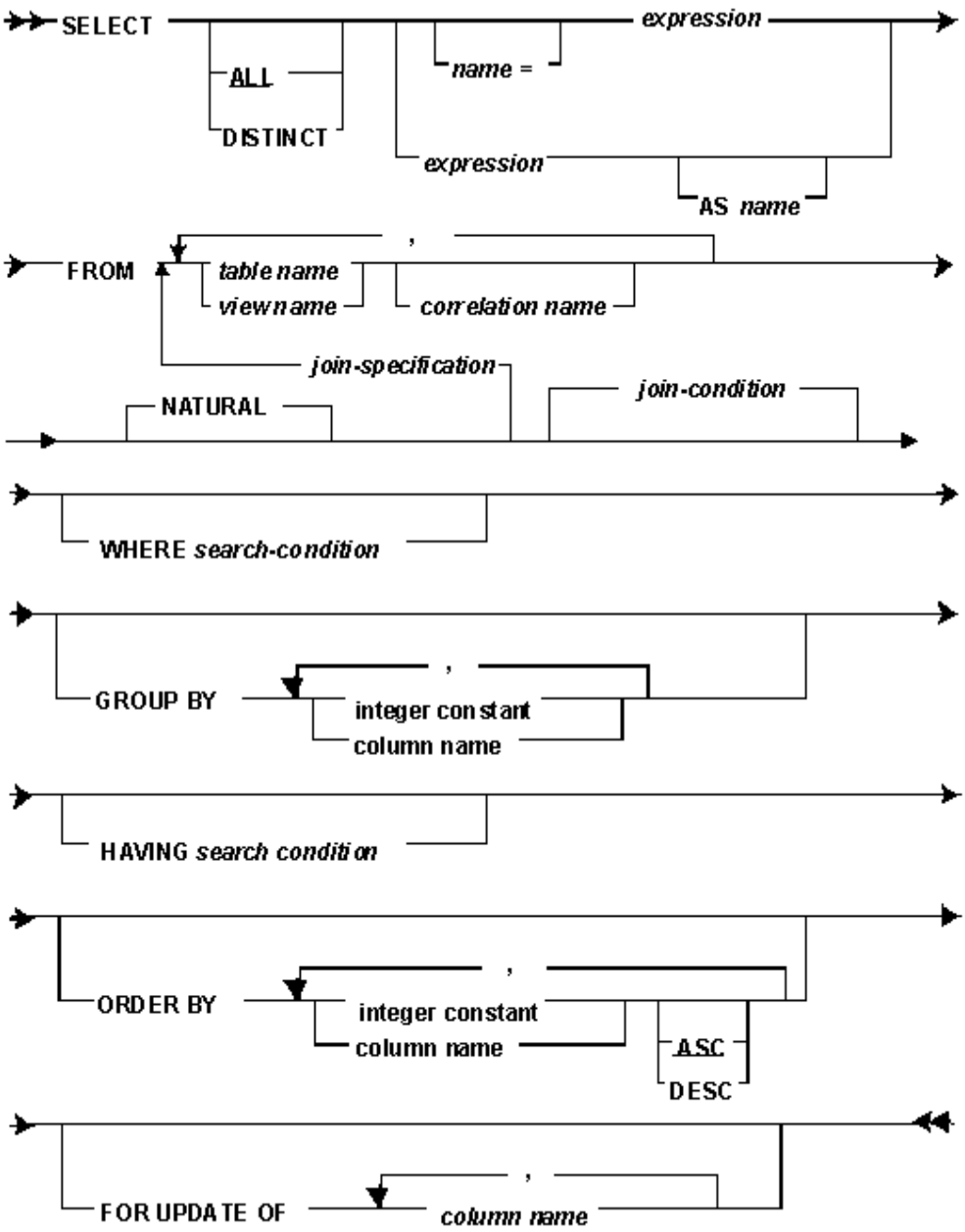
10. SQL – relatsioonilise andmebaasi käsitluskeel: päringukeel ja päringute koostamine

päringukeel - SELECT-lause

SQL-keele päringukeel (alamhulk SQL-keelest) koosneb ühest SQL-konstruktsioonist, SELECT-lausest. Kõik andmete päringud andmebaasi tehakse kasutades SELECT-korraldust. See tingib SELECT-lause väga nüansika ja keerulise struktuuri. Vagu kõik teisedki SQL-keele laused, on a SELECT-lausetel erinevates andmebaasisüsteemides erinev süntaks, mõnel lihtsam mõnel keerulisem. Kõige keerulisem SELECT-lause süntaks on andmebaasisüsteemil ORACLE. Andmebaasisüsteemi ORACLE SELECT-lause süntaksi graaf võtab enda alla üle kümne A4-mõõdus lehekülge. Samas mahub enamiku andmebaasisüsteemide SELCT-lause süntaksi kirjeldus ühe A4-mõõdus lehekülje peale seda täielikult täitmatagi.

andmebaasisüsteemi SQLBase SELECT-lause süntaks

Nii on näiteks andmebaasisüsteemi SQLBase kogu SELECT-lause süntaksi kirjeldav graaf selline:



erinevate andmebaasisüsteemide SQL-keele erinevus

See SELECT-lause süntaksit kirjeldav graaf on võetud andmebaasisüsteemi SQLBase programmeerija juhendist. Nagu näete on siis kasutatud hoopis teist SQL-keele süntaksi graafilise esituse notatsiooni kui seda tegime meie eelmise jaotises. Samas on see, kui tunda juba mõnda teist analoogilist notatsiooni, suhteliselt lihtsalt loetav. Minu eesmärk ei ole siiski tuua loengutesse sisse veel ühte SQL-keele kirjeldamise metakeelt. Minu eesmärk oli näidata, et on olemas erinevaid SQL-keele kirjeldamise graafilisi meetodeid (mis on oma olemuselt siiski väga sarnased) ja ka seda, et erinevates andmebaasisüsteemides on SELECT- Lause süntaks kohati üsna erinev aga samas ka üsna sarnane. Sellele kõigele saate pöörata tähelepanu, kui õppematerjali järgnevates osades uurime andmebaasisüsteemi ORACLE SELECT-lause süntaksit.

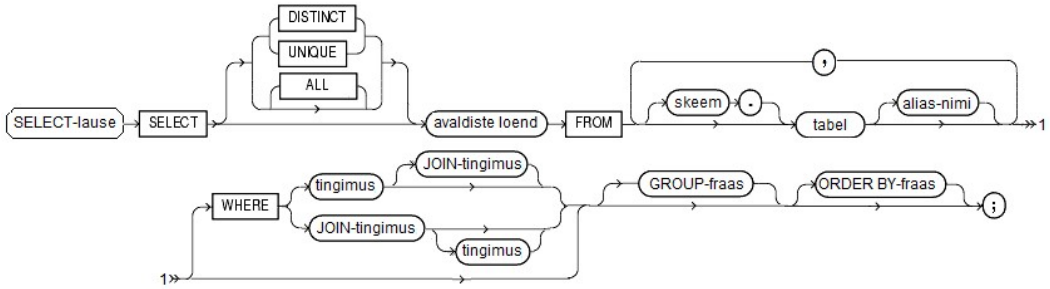
Järgnevates jaotistes ei hakka küll lahkama ORACLE SELECT-lause süntaksit tervikuna. See on liiga mahukas. Seepärast olen ma valinud välja enam kasutatava alamhulga SELECT-lause süntaksist.

vaatleme andmebaasisüsteemi ORACLE kärbitud SELECT-lause süntaksit

10.1. Andmete pärimine ühest tabelist

Nagu juba varem mitu korda öeldud, kasutatakse andmete andmebaasist pärimiseks SELECT-lauset. SELECT-lause süntaks enamikes andmebaasisüsteemides on järgmine:

SELECT-lause süntaks

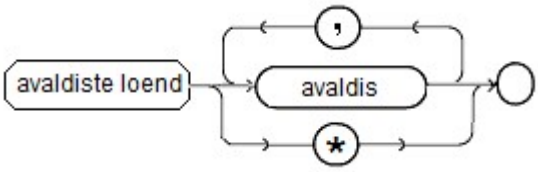


Jätame esialgu vahele fraasid DISTINCT, UNIQUE ja ALL ja vaatame neid natuke hiljem. Kõige pealt vaatame SELECT-lause kõige lihtsamat kuju - jätame ära ka tingimuse (WHERE-fraas), GOUP-fraasi ja ORDER BY-fraasi:

jätame mõned asjad praegu vahele

SELECT avaldiste loend FROM tabel;

Avaldiste loend koosneb avaldistest, mis on ükseisest eraldatud komadega:



Avaldise mõiste on sama, mis eelmistes jaotisteski. Samas on siin avaldiseks kõige suurema tõenäosusega tabeli veeru nimi või siis konstant. Tabeli nimi koosneb kõige lihtsamal SELECT-lause vormingus nagu eelmisteski jaotistes kas skeemi nimest ja tabeli nimest või ainult tabeli nimest:

NÄIDE

SELECT isiku_ID, eesnimi, perekonnanimi, isikukood, synniaeg FROM ISIK;

Mida teeb see SELECT- lause? See SELECT lause väljastab tabelist ISIK kõik kirjed. Igast kirjest võetakse viie veeru väärtused: isiku ID, isiku eesnimi, isiku

perekonnanimi, isikukood ja isiku sünniaeg. Kõik see esitatakse tabeli kujul - igas reas ühe isiku andmed.

fraas *

Avaldiste loendi süntaksi skeemilt on aga näha, et tervet avaldiste loendit võib asendada ühe sümboliga, milleks on "tärn" (" * "). "Tärn" määrab, et tuleb SELECT-lause täitmise väljundisse tuleb panna kõik FROM-fraasis oleva tabeli väljad selles järjekorras nagu need olid CREATE TABLE korralduses, millega tabel loodi:

NÄIDE

```
SELECT * FROM ISIK;
```

fraasid UNIQUE ja DISTINCT

Selliselt kirjutatud SELECT-lause väljastab tabelist kõik kirjed. Kui me nüüd lisame SELECT-fraasi järele kas UNIQUE või DISTINCT (need on samaväärsed), siis väljastatakse andmebaasist ainult unikaalsed read:

DISTINCT-fraasi kasutamise kohatine mõttetus

Kui andmebaas on projekteeritud õigesti, siis avaldiste loendi * (tärn) puhul ei oma DISTINCT-fraas mõtet, kuna õigesti projekteeritud andmebaasi tabelis ei tohi leiduda kirjeid, mille kõikide veergude väärtused on samad. Samuti ei oma mõtet DISTINCT-fraas mõtet siis, kui SELECT-lausega baasist väljastatavate veergude loendis on tabeli primaarvõtme kõik veerud. Seda selle pärast, et primaarvõti on unikaalne üle tabeli kõigi kirjete. Sellest lähtuvalt on toodud näites mõttetu kasutada DISTINCT-fraasi näite esimesel kahel real:

NÄIDE

```
SELECT DISTINCT * FROM ISIK;
SELECT DISTINCT isiku_ID, eesnimi, perekonnanimi, isikukood, synniaeg
FROM ISIK;
SELECT DISTINCT eesnimi, perekonnanimi, isikukood, synniaeg FROM ISIK;
```

SELECT-lause täitmise väljundi unikaalsus

Seejuures kontrollitakse ridade unikaalsust ainult nende veergude väärtuste järgi, mis on SELECT-lause avaldiste loendis. Andmebaasis võivad kaks kirjet olla täiesti erinevad (kui vaadata nende unikaalsuse tuvastamiseks kõiki nende väärtusi) aga kui kahe või enama erineva kirje nende veergude väärtused, mis on kirjeldatud SELECT-lause avaldiste loendis, on samad, siis tulevad nad SELECT-lause täitmise tulemusse ühe reana.

vaikimisi on ALL

SELECT ALL on sama väärne lausega SELECT, sest kui SELEC-fraasi järel puudub üks fraasidest ALL, DISTINCT või UNIQUE, siis võetakse vaikimisi kasutusele fraas ALL.

DISTINCT vs, UNIQUE

Fraas DISTINCT on pärit SQL-keele standardist, fraas UNIQUE aga ORACLE "oma leiutis". Kõigis andmebaasisüsteemides (kus loomulikult on kasutusel SQL-keel) on olemas fraas DISTINCT.

SELECT-lause ja tingimus

Lisame nüüd SELECT-lausele tingimuse. Tingimus on täpselt sama tähendusega nagu eelmises loengus kirjeldatud UPDATE- ja DELETE-lausetel. Kui UPDATE lauses valiti tingimuse järgi tabelist neid kirjeid, mille veergude väärtusi uuendada ja DELETE lauses valiti tingimuse järgi välja neid lauseid mida kustutada, siis SELECT-lauses valitakse tingimuse järgi välja neid kirjeid, mis kuuluvad SELECT-lause täitmise tulemuse kirjete loendisse:

NÄIDE

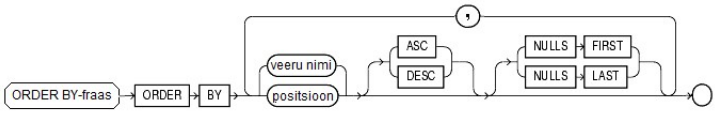
```
SELECT *
FROM ISIK
WHERE SYNNIAEG > 02/15/1960 AND SUGU='M';
```

```
SELECT DISTINCT isiku_ID, eesnimi, perekonnanimi, isikukood, synniaeg
FROM ISIK
WHERE ISIKUKOOD=:sIsikukood;      (:sIsikukood on muutuja nimi selles programmis, mis SQL-lause käivitas)
```

päringu tulemuse järjestamine

ORDER BY-fraas

Sellisel kujul, nagu me praegu SELECT-lauset kasutanud oleme antakse päringu tulemuse read täiesti suvalises järjekorras - sellises, millises andmebaasisüsteem "parasjagu just otsustab" millises järjekorras neid meile näidata. Selleks, et saada päringu tulemus mingis ettemääratud ja sama päringu korral ka alati samas järjestuses tuleb SELECT-lausele liita ORDER BY-fraas. ORDER BY võimaldab anda tabeli veergude järjestuse, mille järgi päringu tulemus sorteeritakse kas kasvavas või kahanevas järjestuses. ORDER BY fraasi süntaks on järgmine:



sorteerimise järjestus ASC ja DESC

ORDER BY- fraasis loetletakse üle koma nende veergude nimed, mille järgi väljundit soovitakse sorteerida. Seejuures kõige pealt sorteeritakse loendis oleva esimese veeru järgi, selle sees teise veeru järgi, selle sees kolmanda veeru järgi jne. nii nagu astmeline sorteerimine ikka toimub. Seejuures saab iga veeru juures öelda, kas selle veeru järgi sorteeritakse kas kasvavas (ASC- ascending) või kahanevas (DESC - descending) järjestuses. Kui veeru kirjelduse juures

ASC/DESC-fraas puudub, siis sorteeritakse selle veeru järgi vaikimisi kasvavas (ASC) järjestuses.

tühjade väljade sorteerimine NULLS FIRST ja NULLS LAST

Sorteerimise järjestuse kirjeldust on veel võimalik täiendada fraasidega NULLS FIRST või NULLS LAST. See määrab selle, kas tühjad väärtused sorteeritakse (olenemata sorteerimisjärjestusest ASC või DESC) kas loendi algusesse või lõppu. Kui fraas NULLS FIRST / NULS LAST on puudu võetakse vaikimisi kasutusele fraas NULLS FIRST, nii nagu normaalselt asi käib - väiksemad väärtused (ja tühjad väärtused on kõige väiksemad väärtused) sorteeritaks ette poole. Konstruktsioonid NULLS FIRST / NULLS LAST on omased ainult ORACLE andmebaasisüsteemile.

NÄIDE

```
SELECT DISTINCT isiku_ID, eesnimi, perekonnanimi, isikukood, synniaeg
FROM ISIK
WHERE ISIKUKOOD=:sIsikukood
ORDER BY perekonnanimi ASC, eesnimi DESC;
```

Nagu ORDER BY-fraasi süntaksist näha, võib veergude nimede asemel kasutada ka nende veergude järjenumbreid SELECT-lause avaldiste loendis:

NÄIDE

```
SELECT DISTINCT isiku_ID, eesnimi, perekonnanimi, isikukood, synniaeg
FROM ISIK
WHERE ISIKUKOOD=:sIsikukood
ORDER BY 3 ASC, 2 DESC;
```

ORDER BY-fraasis saab kasutada ka tabeli nende veergude nimesid, mis antud SELECT-lause avaldiste loendis pole üldse kirjeldatud, kuid tabelis eneses on olemas. Nendele veergudele saab küll viidata ainult nende nime järgi - järjenumbrit neil ju pole:

NÄIDE

```
SELECT DISTINCT isiku_ID, eesnimi, perekonnanimi, isikukood, sugu
FROM ISIK
WHERE ISIKUKOOD=:sIsikukood
ORDER BY  3 ASC, 2 DESC, synniaeg ASC NULLS  LAST;
```

NÄITED

Siia lõppu tahaks lisada veel mõned täiesti kontekstist välja rebitud SELECT-lausete näited:

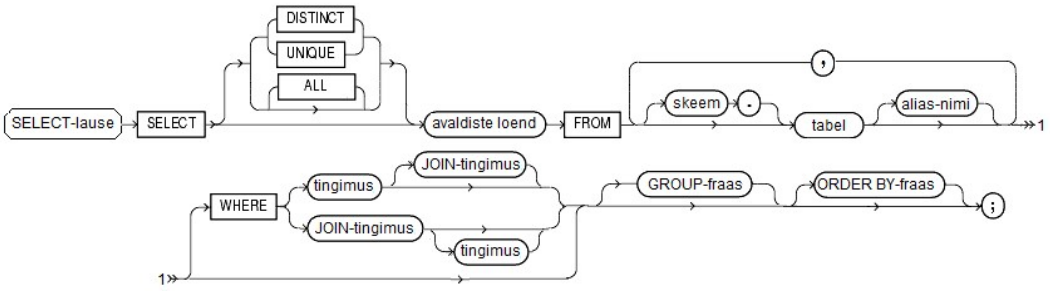
```
SELECT *
   FROM employees
   WHERE department_id = 30;
SELECT last_name, job_id, salary department_id
   FROM employees
   WHERE NOT (job_id = 'PU_CLERK' AND department_id = 30);
```

```
SELECT *
   FROM employees
   WHERE job_id = 'PU_CLERK'
   ORDER BY commission_pct DESC;

SELECT last_name, department_id, salary
   FROM employees
   ORDER BY department_id ASC, salary DESC;

SELECT last_name, department_id, salary
   FROM employees
   ORDER BY 2 ASC, 3 DESC;
```

10.2. Andmete pärimine mitmest tabelist

SELECT-lause süntaks

Selleks, et hakata konstrueerima SELECT-lauset sellise päringu koostamiseks, kus andmed tulevad mitmest tabelist, vaatame veel kord eelmises jaotises toodud SELECT-lause süntaksi graafi:



JOIN-tingimus !

Siin on näha, et SELECT-lause süntaks lubab FROM-fraasi järel kirjeldada ka mitme tabeli nimed. Vaatame, kuidas see mõjutab SELECT-lause kuju. Oluline on veel tähele panna WHERE-fraasis JOIN-tingimust. Kui Ühest andmebaasi tabelist pärimise puhul võisime WHERE-fraasi tervikuna ära jätta, siis mitmest tabelist korraga pärimisel peab SELECT-lause igal juhul sisaldama WHERE-fraasi ja seal sees JOIN-tingimust. JOIN-tingimus kirjeldab seosed tabelite vahel. Loomulikult ei sunni meid miski JOIN-tingimusi kirjutame ja me võime nende kirjeldamisest loobuda. Sellisel juhul on aga päringu tulemuseks Caretsiuse loend (Cartesian product) ja sellest tuleb palju pahandust. Cartesuiuse loendist räägin täpsemalt natuke taga pool.

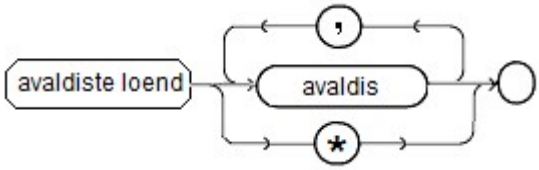
Kahest või enamast tabelist andmete pärimisel on SELECT-lause kõige lihtsam kuju selline:

SELECT avaldiste loend (siin võib nüüd olla veergude nimesid kõikidest nendest tabelitest, mis on loetletud FROM-fraasis)
FROM tabel 1, tabel 2, tabel 3, ... , tabel n

WHERE JOIN-tingimus1 AND JOIN-tingimus2 AND ... AND JOIN-tingimus n-1;

Avaldiste loend koosneb avaldistest, mis on ükstesisest eraldatud komadega nii nagu ühest tabelist pärimiselgi selle vahega et avaldised võivad nüüd olla seotud miteme tabeliga:

SELECT-lause
avaldiste loend



avaldis

Avaldise mõiste on sama, mis eelmistes jaotisteski. Samuti on siin avaldiseks kõige suurema tõenäosusega tabeli veeru nimi või siis konstant. Kui kahes tabelis on sama nimega veerg, tuleb nende eristamiseks kirjutada veeru nime ette üle punkti selle tabeli nimi, kus veerg asub.

tabeli nimi

Tabeli nimi koosneb ka selles SELECT-lause vormingus nagu eelmisteski jaotistes kas skeemi nimest ja tabeli nimest või ainult tabeli nimest.

Iga JOIN-tingimus kirjeldab seose kahe tabeli vahel. Seepärast on JOIN-tingimusi alati ühe võrra vähem kui on kirjeldatud tabeleid SLECT-lause FROM-fraasis. Seos kahe tabeli vahel kirjeldatakse ühe tabeli primaarvõtme kõigi veergude "võrdus seostega" teise tabeli välisvõtme kõigi vastavate veergudega.

NÄIDE

Olgu meil kaks tabelit KAUP ja KAUBA_TYYP:

tabel KAUBA_TYYP

CRAETE TABLE KAUBA_TYYP(
KAUBA_TYYP_ID INTEGER NOT NULL, (see on primaarvõti)
NIMETUS VARCHAR2(60) NOT NULL,
KOMMENTAAR LONG VARCHAR
);

CRAETE TABLE KAUP(
KAUP_ID INTEGER NOT NULL, (see on primaarvõti)
KAUBA_TYYP_ID INTEGER NOT NULL, (see on välisvõti, mis seob seda tabelit tabeliga KAUBA_TYYP)
KOOD CHAR(10) NOT NULL,
NIMETUS VARCHAR2(60) NOT NULL,
YHIK CHAR(10) NOT NULL
YHIKU_HIND DECIMAL(8,2) NOT NULL,

```
KOMMENTAAR LONG VARCHAR
);
```

Kirjutame nüüd SLECT-lause, mis pärib andmebaasist välja kauba tüübi nime, kauba nime, kauba arvestusühiku ja ühiku hinna. Sorteerime päringu väljundi read kauba tüübi nime ja kauba nime järgi:

```
SELECT KAUBA_TYYP.NIMETUS, KAUP.NIMETUS, YHIK, HIND
FROM KAUBA_TYYP, KAUP
WHERE KAUBA_TYYP.KAUBA_TYYP_ID=KAUP.KAUBA_TYYP_ID
ORDER BY 1, 2;
```

fraas *

Ka siin võime avaldiste loendi asemel kasutada sümbolit " * " (tärn). Siis ilmuvad SELECT-lause täitmise tulemusse mõlema tabeli kõik veerud selles järjekorras, millises need veerud olid tabelite loomise CREATE-lausetes. FROM-fraasis oleva esimese tabeli veerud on enne teise tabeli veerge. Kui me asendame avaldiste loendi sümboliga tärn, siis ei saa me enam ORDER BY-fraasis kasutada veergude järjenumbreid, vaid peame asendama need veergude reaalsete nimedega.

```
SELECT *
FROM KAUBA_TYYP, KAUP
WHERE KAUBA_TYYP.KAUBA_TYYP_ID=KAUP.KAUBA_TYYP_ID
ORDER BY KAUBA_TYYP.NIMETUS, KAUP.NIMETUS;
```

fraasid DISTINCT (UNIQUE) ja ALL

DISTINCT- ja ALL-fraasid toimivad mitmest tabelist pärimisel analoogiliselt sellele, nagu need toimisid ühest tabelist pärimisel.

SELECT-lause ja tingimus

Nüüd jääb veel SELECT-lausele lisada tavaline filtreeriv tingimus. Selle mõte on sama mis ühest tabelist pärimiselgi - piirata ridade arvu SELECT-lause täitmise tulemis.

NÄIDE

Piirame näiteks viimases näites kaupade loendit ühe tingimusega mõlemas tabelis. Vaatleme ainult selliseid kaupu, mis kuuluvad tüübi "majapidamisvahendid" alla ja mille ühiku hind on suurem kui 155 krooni:

```
SELECT *
FROM KAUBA_TYYP, KAUP
WHERE KAUBA_TYYP.KAUBA_TYYP_ID=KAUP.KAUBA_TYYP_ID AND
        KAUBA_TYYP.NIMETUS='majapidamistarbed' AND HIND>155
ORDER BY KAUBA_TYYP.NIMETUS, KAUP.NIMETUS;
```

tabeli alias-nimi

Jääb veel selgitada üks fraas ja siis võime mitut tabelit hõlmava SELECT-lause kirjeldamise lõpetada. See fraas on ALIAS-fraas. Kui SELECT-päringu erinevates tabelites on sama nimega väljad, siis tuleb iga kord kui sellise veeru nime kasutada, lisada veeru nime ette tabeli nimi. Tabeli nimed võivad aga olla väga pikad ja neid on ebamugav kasutada. Seepärast on SQL-keeles võimalus kirjeldada tabelile alias-nimi. Tabelile alias-nime kirjeldamiseks kirjutatakse see FROM-fraasi üle tühiku tabeli nime järele. Nüüd võib igal pool veeru nimede ees asendada tabeli nime alias-nimega. Alias nimedena kasutatakse tavaliselt üksikuid tähti, kuid alais nimi võib olla ka pikem. Viimati kirjeldatud SELECT-lause näeb tabelitele alias-nimede kirjeldamise järel välja järgmine :

NÄIDE

```
SELECT A.NIMETUS, B.NIMETUS, YHIK, HIND
FROM KAUBA_TYYP  A, KAUP  B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
        A.NIMETUS='majapidamistarbed' AND HIND>155
ORDER BY A.NIMETUS, B.NIMETUS;
```

pane kõikide veergude nime ette alias-nimi

Tegelikult on soovitav kõikide veergude nimede ette panna alias-nimi. See väldib seda, et andmebaasi struktuuri muutumisel, kui tabelitesse lisatakse uusi veerge, ei muutuks SELECT-lause valeks, juhul kui ühte tabelitest pannakse sellise nimega veerg, mis teises on juba olemas:

NÄIDE

```
SELECT A.NIMETUS, B.NIMETUS, B.YHIK, B.HIND
FROM KAUBA_TYYP  A, KAUP  B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
        A.NIMETUS='majapidamistarbed' AND B.HIND>155
ORDER BY A.NIMETUS, B.NIMETUS;
```

NÄITED

Siia lõppu tahaks lisada veel mõned täiesti kontekstist välja rebitud SELECT-lausete näited:

```
SELECT last_name, job_id, departments.department_id, department_name

  FROM employees, departments

  WHERE employees.department_id = departments.department_id;
```

```
SELECT e1.last_name ||' works for ' || e2.last_name || " Employees and Their
Managers"

FROM employees e1, employees e2

  WHERE e1.manager_id = e2.employee_id

    AND e1.last_name LIKE 'R%';



SELECT d.department_id, e.last_name

  FROM departments d, employees e

  WHERE d.department_id(+) = e.department_id

  ORDER BY d.department_id;
```

10.3. Statistilised funktsioonid ja andmete grupeerimine

statistilised funktsioonid SQL-keeles

SQL keeles on kuus statistilist funktsiooni, mis töötavad üle kirjete grupi. Nendeks funktsioonideks on:

SUM(avaldis)	avaldise väärtuse summa üle kõikide päringu ridade või üle ridade grupi
MAX(avaldis)	maksimaalne avalavaldise väärtus üle kõikide päringu ridade või üle ridade grupi
MIN(avaldis)	minimaalne avalavaldise väärtus üle kõikide päringu ridade või üle ridade grupi
COUNT(*)	ridade arv päringus või päringu grupis
AVG(avaldis)	avalavaldise väärtuste keskmine üle kõikide päringu ridade või üle ridade grupi
STD(avaldis)	avalavaldise väärtuste standardhälve üle kõikide päringu ridade või üle ridade grupi

NÄITED

Kuidas neid funktsioone siis kasutada. Teeme mõne SELECT-lause ja kasutame neid funktsioone:

```
SELECT SUM(HIND) FROM KAUP;   (leiab kõigi kaupade ühiku hindade summa
- absurd, aga näitlik absurd)
SELECT MAX(HIND) FROM KAUP;   (leiab maksimaalse ühiku hinna)
```

SELECT MIN(HIND) FROM KAUP; (leiab minimaalse ühiku hinna)
SELECT COUNT(*) FROM KAUP: (leiab kaupade arvu tabelis)

Need on äärmiselt lihtsad näited. Kordame sama küsides välja ainult ühe kauba tüübi statistilised andmed

```
SELECT SUM(A.HIND)                    (sporditarvete ühiku hindade summa)
FROM KAUP A, KAUBA_TYYP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
B.NIMETUS='sporditarbed';
```

```
SELECT MAX(A.HIND)          (maksimaalne ühiku hind sporditarvete hulgas)
FROM KAUP A, KAUBA_TYYP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
B.NIMETUS='sporditarbed';
```

```
SELECT MIN(A.HIND)          (minimaalne ühiku hind sporditarvete hulgas)
FROM KAUP A, KAUBA_TYYP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
B.NIMETUS='sporditarbed';
```

```
SELECT COUNT(*)                    (sporditarvete arv tabelis)
FRO
SM KAUP A, KAUBA_TYYP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
B.NIMETUS='sporditarbed';
```

Seda kõike saab teha ka ühe lausega:

ühes SELECT-lauses mitu statistilist funktsiooni

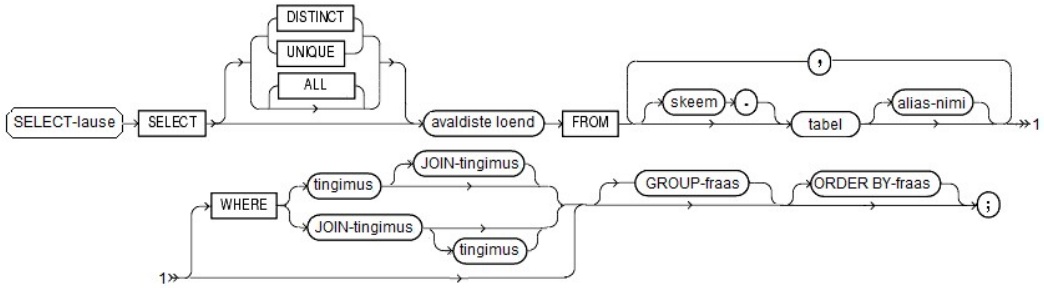
```
SELECT SUM(A.HIND), MAX(A.HIND), MIN(A.HIND), COUNT(*), AVG(A.HIND),
SUM(A-HIND) / COUNT(*)
FROM KAUP A, KAUBA_TYYP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID AND
B.NIMETUS='sporditarbed';
```

Viimase kahe SELECT-lause avaldise loendis olevad väärtuseks on arvutatud keskmine ühiku hind.

päringu tulemuse grupeerimine

Kas on võimalik leida statistiliste funktsioonide väärtused ühe SELECT-lausega mitmetes lõigetes. Eelmises näites leidsime me statistilised väärtused ühe kaubagrupi kohta. Kas oleks võimalik leida statistilised väärtused kõikide kauba

gruppide kohta korraga? On küll. Selleks vaatame veel korra SELECT-lause süntaksi graafi:

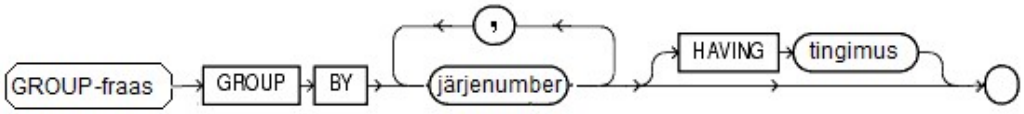


GROUP-fraasi täiendus

Ainuke komponent, mis on sellest graafist veel kirjeldamata on GROUP-fraas. GROUP-fraas võimaldab kirjeldada need veerud SELECT-lause avaldiste loendist, mille alusel päringu tulemus grupeeritakse. Seejuures on üks väike piirang - grupeerida saab ainult selliseid SELECT-lauseid, mille avaldiste loendis on statistilisi funktsioone. Ja veel üks piirang. Grupeerimisloendisse ei saa panna neid avaldisi, kus on kasutatud statistilisi funktsioone ja sinna peab panema kõik avaldised, milles ei ole statistilisi funktsioone. Grupeerimiseks saab kasutada ainult neid avaldisi, mis on toodud SELECT-lause avaldiste loendis.

GROUP-fraasi süntaks

GROUP-fraasi süntaks on järgmine:



GRPUP-fraas algab tekstiga GROUP BY, millele järgnevad järjenumbrid, mis tähistavad grupeerivate avaldiste positsiooni SELECT-lause avaldiste loendis. Sellel võib järgneda tingimus, mida saab rakendada SELECT-lause avaldiste loendis olevate statistiliste funktsioonide väärtusele. Kui WHERE tingimusega valitakse välja tabelist ridu, mis kuuluvad päringusse, suus HAVING tingimusse saab kirjutada tingimusi juba välja valitud kirjete grupeeritud väärtustele.

NÄIDE

Vaatame kõigepealt lihtsat grupeerimist, kus leitakse kõigis kaubatüüpides olevate toodete arv ja nende gruppide kaupade minimaalne ja maksimaalne ühiku hind:

```
SELECT A.NIMETUS, COUNT(*), MIN(B.HIND), MAX(B.HIND)
FROM KAUBA_TYYP A, KAUP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID
```

```
GROUP BY 1
ORDER BY 1;
```

Kui nüüd soovida kaubagruppide sees grupeerida hinnad mõõtühiku järgi, siis saame sellise SELECT-lause:

```
SELECT A.NIMETUS, B.YHIK, COUNT(*), MIN(B.HIND), MAX(B.HIND)
FROM KAUBA_TYYP A, KAUP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID
GROUP BY 1, 2
ORDER BY 1, 2;
```

Kui nüüd soovime statistikat ainult selliste kaupade kohta, mille minimaalne hind on suurem kui 100 krooni, peame kasutusele võtma HAVING-tingimuse:

```
SELECT A.NIMETUS, B.YHIK, COUNT(*), MIN(B.HIND), MAX(B.HIND)
FROM KAUBA_TYYP A, KAUP B
WHERE A.KAUBA_TYYP_ID=B.KAUBA_TYYP_ID
GROUP BY 1, 2
HAVING MIN(B.HIND)>100
ORDER BY 1, 2;
```

NÄITED

Siia lõppu tahaks lisada veel mõned täiesti kontekstist välja rebitud SELECT-lausete näited, mis sisaldavad GROUP-fraasi.

```
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
GROUP BY department_id;
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
WHERE job_id = 'PU_CLERK'
GROUP BY department_id;
```

```
SELECT department_id, MIN(salary), MAX (salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) < 5000;
```

```
SELECT d.department_id, e.last_name
FROM departments d LEFT OUTER JOIN employees e
ON d.department_id = e.department_id
ORDER BY d.department_id;
```

```
SELECT channel_desc, calendar_month_desc, co.country_id,
TO_CHAR(sum(amount_sold) , '9,999,999,999') SALES$
FROM sales, customers, times, channels, countries co
WHERE sales.time_id=times.time_id
AND sales.cust_id=customers.cust_id
AND sales.channel_id= channels.channel_id
AND customers.country_id = co.country_id
AND channels.channel_desc IN ('Direct Sales', 'Internet')
AND times.calendar_month_desc IN ('2000-09', '2000-10')
AND co.country_id IN ('UK', 'US')
GROUP BY GROUPING SETS(
(channel_desc, calendar_month_desc, co.country_id),
(channel_desc, co.country_id),
( calendar_month_desc, co.country_id) );
```

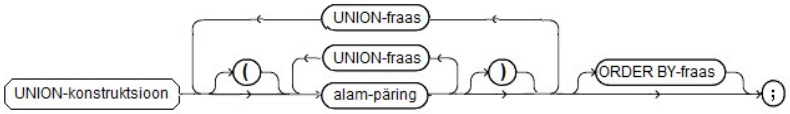
10.4. Päring mitme SELECT-lause järgi. UNION ja SELECT-lausete ühendamine

meid huvitavad andmed on eraldi tabelites

Alati ei piisa soovitud andmete pärimiseks andmebaasist ühest SELECT-lausest. Andmed, mida vajatakse ühe loendina asuvad andmebaasis laiali erinevates tabelites ja selleks et neid kätte saada, on vaja teha mitu erinevat päringut. Ainult probleem on nüüd selles, et erinevate SELECT-lausetega päritud kirjete loendeid ei saa ühes koos sorteerida. Me saame küll mitu erinevat, sisemiselt sorteeritud, kirjete loendit, kuid ühtse loendi saamiseks peaksime need programmis kokku liitma ja siis seal kirjutama programmi selle koond-loendi sorteerimiseks.

UNION-konstruktsioon

Kuna vajadus selliste mitme SELECT-lause järgi moodustatud ühtse päringu tulemuste järgi on üsna suur, siis on SQL-keeles selleks olemas eraldi keele konstruktsioon, mida nimetatakse UNION-iks. Union võimaldab liita kokku mitu sellist SELECT-lauset, mille avaldiste loendis on võrdne arv liikmeid ja samas positsioonis olevad avaldised on kõikides SQL-lausetes sama tüüpi. UNION-kostruktsiooni süntaks on järgmine:

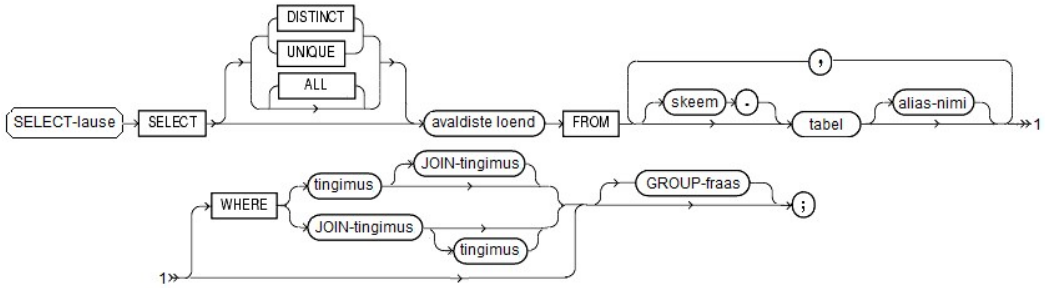


mitme SELECT-lause ühendamine üheks

Selle konstruktsiooniga saab ühendada mitu SELECT-lauset, tekitada neile ühine väljund ja rakendada sellel ühisele väljundile ühtset sorteerimist. Selles skeemis on SELECT laused tähistatud fraasiga "alam-päring"- Miks ei ole selle

koha peale kirjutatud "SELECT-lause"? Seda selle pärast, et selle koha peal on SELECT-lause süntaks eraldi seisva SELECT-lause süntaksist veidi erinev. Seal puudub ORDER-BY fraas, sest see on toodud UNION-konstruktisioni koosseisu ja alam-päringud ei vaja iga üks oma ORDER BY kirjeldust. Nagu öeldud erineb alam-päring tavalisest SELECT-lausest selle poolest, et tal puudub ORDER BY-grupp:

lauseks (üheks päringuks)

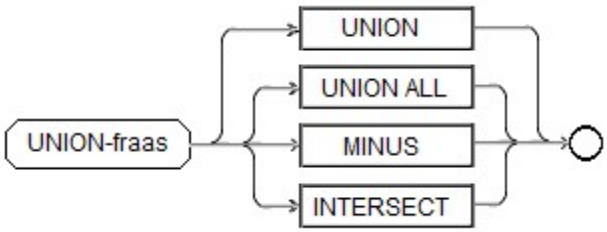


alam-päringu select lause erineb iseseisvast SELECT-lausest - puudub ORDER BY-grupp

Muu kõik jääb täpselt samaks.

Olekski kõik UNION-konstruktsiooni kohta, kui UNION-fraase erinevate SELECT-lausete liitmiseks poleks mitu:

erinevad UNION-"tehted"



UNION liidab erinevate SELECT-lausete päringud selliselt, et kõik read päringutulemuses on unikaalsed.

UNION ALL liidab erinevate SELECT-lausete päringu tulemuste kõik read.

INTERSECT väljastab ainult need read, mis on olemas mõlemas päringutulemustes

MINUS väljastab need read, mis on esimese SELECT-lause tulemuses kuid puuduvad teises.

sulgude kasutamine SELECT-lausete ühendamise järjekorra muutmiseks

Kui SELECT-lauseid on seotud rohkem kui kaks ja neid on ühendatud üle erinevate meetodite (UNION, UNION ALL, INTERSECT ja MINUS), siis õige kombinatsiooni ja “liitumistehete” rakendumise tagamiseks on mõttekas kasutada grupeerimiseks sulgusid (nii nagu kasutatakse sulgusid aritmeetika tehete järjekorra muutmiseks). Kui sulge mitte kasutada, siis tuleb teada et INTERSECT on kõrgema prioriteediga operaator kui ülejäänud. Ülejäänud operaatorite prioriteedid on võrdsed ja neid rakendatakse esinemise järjekorras.

Veel üks piirang. UNION-konstruktsiooni abil ühendatavates SELECT-lausetes ei saa avaldiste loendis kasutada LONG-tüüpi veergusid (LONG VARCHAR, BLOB, BINARY, PICTURE jms.).

ei saa kasutada LONG-tüüpi muutujaid

10.5. Outer Join

"katkenud seosed" tabelite vahel

Kui kirjeldada SELECT-lauses seos kahe tabeli vahele, ilmuvad päringu tulemusse ainult need read, mille kohta on olemas vähemalt üks kirje mõlemas tabelis. On ju JOIN-tingimus täidetud ainult siis kui mõlemas seotud tabelis on vastava võtmeväärtusega kirje olemas (ühelt poolt primaarvõtmes ja teiselt poolt välisvõtmed). Tihti on aga vaja, et päringusse kuuluksid ka sellised read, kus seose ühel pool kirjet ei ole.

NÄIDE

Olgu meil kaks tabelit (ISIK ja AADRESS), millest ühes on isikute andmed ja teises isikute aadresside andmed. Samas leidub tabelis ISIK isikuid, kelle kohta pole tabelis AADRESS ühtegi aadressi. Kui me nüüd teeme päringu kahest tabelist, nägemaks inimeste nimesid ja aadresse, siis ilmuvad päringu tulemusse ainult need isikud, kelle on aadress olemas ja need, kellel seda pole päringu tulemusse ei ilmu. Aga meie soovime näha kõiki inimesi olenemata sellest, kas neil on andmebaasis aadress registreeritud või mitte. Neil isikutel võiks lihtsalt aadressi asemel olla "tühi koht". Sellisel juhul tuleb andmebaasimootorit sellest "informeerida", näidates ära, millisesse tabelisse tuleb päringu ajal lisada tühje kirjeid. Antud näite puhul on selleks tabel AADRESS, millesse tuleb päringu ajal tekitada puuduvate kirjete asemele tühje kirjeid.

päringu aegsed "ajutised tühjad kirjed"

Selleks et päringu aegseid tühje kirjeid tekitada peame me andmebaasisüsteemile näitama millisesse kahest seoses olevasse tabelisse neid tühje kirjeid teha tuleb:

SELECT i.isik_id, i.eesnimi, i.perenimi, a.riik, a.linn. a.tanav, a.maja, a.korter

FROM isik i, aadress a

WHERE i.isik_id = a.isik_id (+);

Outer Join

Kombinatsioon (+) tähistab siin seose seda poolt, kuhu päringu aegsed tühjad kirjed tuleb lisada. Sellist seost nimetatakse Outer Join-ks

Juhul kui soovitakse saada isikuid kellel on aadressid ja aadresse, mis pole isikuga seotud, kuid mitte need isikud, kellel pole aadresse, siis näeb päring välja järgmine:

NÄIDE

Oracle

SELECT e.isik_id
FROM isik e, aadress d
WHERE e.isik_id (+)= d.isik_id;

erinevate andmebaasisüsteemide SQL-keeltes kasutatakse Oute Join-i tähistamiseks erinevaid keelelisi konstruktsioone

Erinevatel andmebaasisüsteemidel on Outer Join-i tähistamine erinev. Näiteks SQL Server kasutab tähistusi *= ja =*. Tärn pannakse aga võrreldes Oracle (+)-ga vastaspoolele – sellele poolele, mille kirje peab igal juhul olema esitatud. Eelnevad näited näevad SQL Serveri puhul välja järgmised:

SQL Server

SELECT i.isik_id, i.eesnimi, i.perenimi, a.riik, a.linn. a.tanav, a.maja, a.korter
FROM isik i, aadress a
*WHERE i.isik_id *= a.isik_id;*

SELECT i.isik_id, i.eesnimi, i.perenimi, a.riik, a.linn. a.tanav, a.maja, a.korter
FROM isik i, aadress a
WHERE i.isik_id = a.isik_id;*

10.6. Cartesiuse loend *(Cartesian Product)*

Cartesiuse loend

Kui samas SELECT lauses on kirjeldatud päring mitmest tabelist, kuid seose tingimus (Join-tingimus) on jäetud kirjeldamata, siis on päringu tulemuseks nn. Cartesiuse loend, kus on tehtud ühe tabeli iga rea seos teise tabeli kõikide ridadega. Näiteks olgu meil tabelid ISIK ja AADRESS kus esimeses on 3000 isikut ja teises 9000 aadressi. Kui mee teeme nüüd päringu nendest kahest tabelist, millel pole kirjeldatud tabelite vahelist Join-tingimust, saame tulemuseks 27 miljonit (27 000 000) rida (3000 * 9000 = 27 000 000):

NÄIDE

*SELECT * FROM ISIK, AADRESS*
ORDER BY ISIK.PEREK_NIMI, ISIK.EESNIMI;

Cartesiuse loendi moodustamine

Toodud näites saime "kõigest" 27 000 00 rida aga kujutage ette olukorda, kus SELECT-lause on tehtud läbi nelja tabeli, millest ühes o 500 000 kirjet, teises 10 000 000, kolmandas 400 kirjet ja viiendas 50 kirjet. Päringu tulemuseks on

500 000 * 10 000 000 * 400 * 50 kirjet, mis teeb kokku 100 000 000 000 000 000 kirjet ja kui nende kirjete pikkus on ca 100 baiti, siis vajame me selle päringu teostamiseks 10 000 000 000 000 000 000 baiti mälu. Kui nüüd andmebaasiserver seda päringut täitma läheb, siis koormab ta sellega ära kogu oma protsessori jõudluse ja lõpuks täidab ka kogu kasutada oleva vaba ketas-mälu. Asi lõppeb täieliku kaosega.

andmebaasi serveril lõpeb kaosega

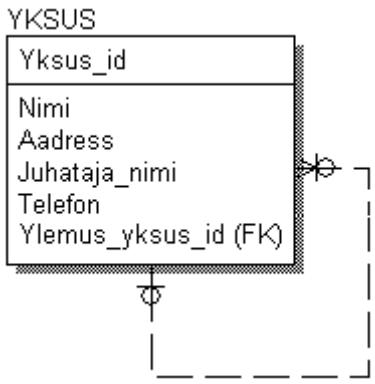
10.7. Päring sama tabeli "mitmest koopiast"

Vahel on andmebaasi tabel seoses iseendaga. Või on ta mitme seosega seotud mõne teise tabeliga. Sellisel juhul võib tekkida vajadus kasutada sama tabelit sama SELECT-lause koosseisus mitu korda. Kui vaadata SQL lause FROM-grupi kirjeldamise loogikat, siis leiamegi lahenduse sealt – sama tabel võib SELECT-lause FROM-grupis esineda mitu korda ainul et talle tuleb igal esinemisel anda erinev alias-nimi.

sama tabeli esinemine samas SELECT-lauses mitu korda

Olgu meil näiteks andmebaasis tabel YKSUS, kus sama tabelis ees on kirjeldatud üksuste alluvus:

NÄIDE



tabeli seos iseendaga

Siin on igas kirjes lisks üksuse ID-le (Yksus_id) ka selle üksuse ülemusüksuse ID (Ylemus_yksus_id). Kui me nüüd tahame saada loendit üksuste nimedest, millega samal real oleks ka ülemusüksuse nimi peame kirjutama SELECT-lause, mis kasutab üks kord tabelit YKSUS üksuste loendi moodustamiseks ja teine kord selleks, et siduda üksusega tema ülemus-üksuse:

ühe-kordne rekursioon

```
SELECT a.NIMI, B.NIMI
FROM YKSUS a, YKSUS b
WHERE a.ylemus_yksus_id=b.Yksus_id
Order by 2;
```

Sellel lausel on ainult see viga, et loendisse ei ilmu need üksused, millel ülemust pole – nad on ise hierarhia tipus. Et seda saavutada peame kasutama Outer Joini:

Outer Join

```
SELECT a.NIMI, B.NIMI
FROM YKSUS a, YKSUS b
WHERE a.ylemus_yksus_id=b.Yksus_id (+)
Order by 2;
```

Kui tahame saada iga üksuse kohta veel järgmise taseme ülemusi, siis peab lause olema järgmine:

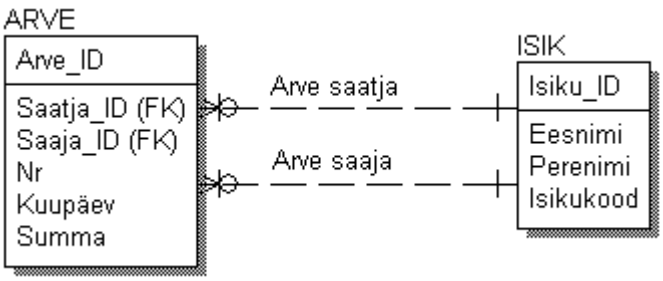
kahe kordne rekursioon

```
SELECT a.NIMI, B.NIMI, c.NIMI
FROM YKSUS a, YKSUS b, YKSUS C
WHERE a.ylemus_yksus_id=b.Yksus_id (+) AND
        b.ylemus_yksus_id=c.yksus_id (+)
Order by 2, 3;
```

Vaatame nüüd veel ühte teistsugust näidet. Olgu meil tabel ARVE, kus on arved ja tabel ISIK, kus on isikud, kes on tabelis arve olevaid arveid väljastanud ja ka need kellele need arved on saadetud:

NÄIDE

kaks seost tabelite vahel



Kui me nüüd soovime saada loendit, kus on arve number, kuupäev, summa ning saatja ja saaja nimi, siis peame kirjutama järgmise SELECT-lause:

```
SELECT a.nr, a.kuupaev, a.summa, b.eesnimi, b.perenimi, c.eesnimi, c.perenimi
FROM ARVE a, ISIK b, ISIK c
WHERE a.saatja_ID=b.isiku_ID AND a.saaja_id=c.isiku_id
ORDER BY 2, 1;
```

Siin (antud juhul) ei ole vaja Outer Joini kasutada, kuna arvel on alati igal juhul olemas nii arve saatja kui ka arve saaja. Kui aga oleks võimalus, et üks neist võib ka puududa, siis ilma Outer Joini kasutamiseta me siin ei pääseks.

10.8. Tühja väärtust tähsitav konstant NULL ja loogilised tehted sellega

konstant NULL

NULL on SQL-keele konstruktsioon, mis tähistab tühja väärtust. Stringi korral on ta võrdne tühja stringiga (''). Kuupäeva korral on ta võrdne tühja kuupäevaga (kuupäevaga, millele ei ole päeva, kuud ega aasta). Arvu puhul on ta võrdne tühja arvuga (mitte arvuga 0). Loogilise avaldise puhul väärtust NULL üldjuhul ei kasutata. Muude objektide puhul (pilt, dokument, film, heli) on ta võrdne tühja dokumendiga. Viimasel juhul on olemas sisuline analoogia tühja stringiga.

IS ja IS NOT koos konstandiga NULL

Konstandiga NULL võrdlemise WHERE- ja HAVING-tingimsues on teatavad erisused võrreldes arvude tekstide ja kuupäevade väärtuste omavahelise võrdlemisega. Kui soovitakse võrrelda konstanti NULL mõne veeru väärtusega tuleb kasutada keelelist konstruktsiooni IS NULL või IS NOT NULL:

NÄIDE

NIMI IS NULL
NIMI IS NOT NULL

= ja <> ei tohi kasutada koos konstandiga NULL

Keelekonstruktsioonid NIMI = NULL ja NIMI != NULL ei toimi. Otseselt viga välja ei anta aga täpselt pole võimalik aru saada mis tegelikult toimub. See sõltub muidugi ka konkreetsest andmebaasisüsteemist, kuidas seda situatsiooni käsitletakse.

Sisuliselt on **IS** samane märgiga **=** (samane) ja **NOT IS** märgiga **<>** (ei ole samane). Igal pool mujal võib kasutada kas ühte või teist täiesti meelevaldselt. Näiteks: **ISIKU_ID IS NOT 5** on samane **ISIKU_ID=5 ja nende** kasutamine on lubatudparalleelselt. Konstandiga NULL saab kasutada siiski ainult kas **IS** või **IS NOT** keele-konstruktsiooniga.

Oracle: NULL ei võrdu NULL'ga

ORACLE's ei tekita kirjete vahelist seost võrdus ISIK.ISIK_ID=AADRESS.ISIK_ID, kui mõlema tabeli veergude ISIK_ID väärtused on NULL'd. Kui on olemas selline võimalus, tuleb seose kirjeldamiseks kirjutada tingimus

NÄIDE

WHERE (ISIK. ISIK_ID IS NULL AND AADRESS. ISIK_ID IS NULL)
OR ISIK.ISIK_ID=AADRESS.ISIK_ID

NULL võrdlemine NULL'ga annab tulemuseks määramatuse (undefined)

NULL võrdlemine NULL-ga annab tegelikult tulemuseks "määramatuse", mis on ORACLE's tõeväärtuse kolmas olek. Seda interpreteeritakse mõningatel juhtudel kui FALSE't.

kui tingimuse mõne komponendi väärtus on määramatus on kogu tingimuse tulemus määramatus, mida interpreteeritakse kui FALSE't

Kui tingimuse kas või ühe komponendi tulemuseks on "määramatus", siis on kogu lause tulemus määramatus. Järelikult ei ole sellisel hetkel enam oluline mis on lause teiste komponentide väärtused – tulemus on nii või teisiti "määramatus" (mida interpreteeritakse kui FALSE't).

kolme-valentne tõeväärtus Oracle's

ORACLE's ei ole WHERE-tingimuse komponentide võrdlustulemused mitte TRUE ja FALSE, vaid võimalik on ka tulemus "määramatus" (undefined).

10.9. Tingimuste eri-konstruktsioonid

SQL-keeles on olemas tingimuste esitamiseks erikonstruktsioonid

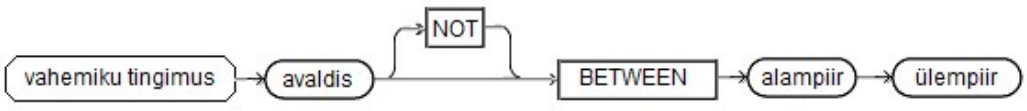
Lisaks tavalistele tingimus-avaldistele, mida tehakse üle võrdlus-tehtemärkide =, < , > >= jne. On SQL-keeles veel mitu tingimus-konstruktsiooni, mida peab eraldi kirjeldama.

10.9.1. Vahemikku kuulumine või mitte kuulumine

vahemikku kuulumine

SQL-keeles on spetsiaalne keeleline konstruktsioon, et lahendada "inimkeele" lähedasel viisil avaldis

n < =X <= m või **NOT (n <= X <= m)**



NÄIDE

palk BETWEEN 10000 20000 (so. palk>=10000 AND palk<=20000)

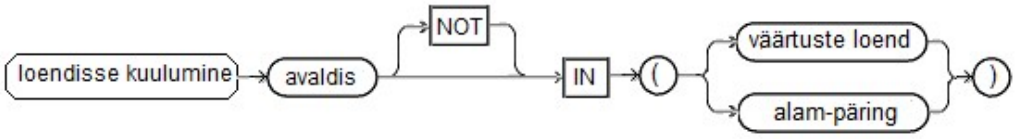
nimi NOT BETWEEN 'Ben' 'Ken' (so. NOT (nimi>='Ben' AND nimi<='Ken')

Kontrollitakse avaldise langemist (või mittelangemist) vahemikku, mis on määratud <alampiiri> ja <ülempiiriga> (kaasaarvatud). Kõik kolm avaldist peavad olema sama andmetüüpi.

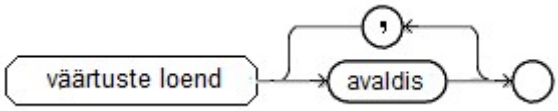
10.9.2. Loendisse kuulumine või mitte kuulumine

loendisse kuulumine

Loendisse kuulumise tingimuses kontrollitakse, kas etteantud avaldise väärtus kuulub etteantud väärtuste loendisse. Loend võib olla ette antud nii fikseeritud väärtuste (konstantide/avaldiste) jadana kui SELECT-lause päringuna:



Kontrollitakse, kas avaldis kuulub (või ei kuulu) etteantud väärtuste loendisse. Kontrollitav avaldis ja kõik loendi väärtusi määravad avaldised peavad olema sama andmetüüpi:



kontrollitav avaldis ja loendi liikmed peavad olema sama andmetüüpi

Loend millesse avaldise kuulumist kontrollitakse võib olla määratud ka SELECT-lausega. Sellise SELECT-lause väärtuste loendis on üks ainus avaldis, mille andmetüüp peab kokku langema kontrollitava avaldise andmetüübiga.

NÄIDE

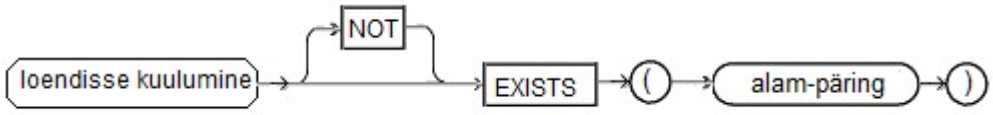
Teeme nüüd paar näidet ka:

```
NADALAPAEV IN ('L', 'P')              (on puhkepäev)
ISIKU_TYYP_KD IN (SELECT ISIKU_TYYP_KD FROM ISIKU_TYYP WHERE LIIK=1)
NADALAPAEV   NOT  IN ('L', 'P')    (pole puhkepäev)
```

10.9.2. Eksistentsi või mitte-eksistentsi kontrollimine

eksistentsi kontroll

kontrollitakse kas alam-päringu tulemus on tühi või mitte. Kui alampäringu tulemuses on kas või üks rida on tingimus tõene (TRUE). Kui alam-päring ei anna tulemuseks ühtegi rida, siis on tingimuse tulemus väär (FALSE):



NOT-fraasiga saab tingimuse tulemi pöörata vastupidiseks. alam-päring on WHERE-tingimuse kaudu seotud selle päringuga, mille komponent ta on. Võrreldes IN-konstruktsiooniga õnnestub teha hoopis keerukamaid lahendusi.

NÄIDE

```
SELECT EESNIMI, PERENIMI
FROM ISIK I
WHERE EXISTS (SELECT 1
                    FROM TOOLEPING T
                    WHERE T.ISIK_ID=I.ISIK_ID AND (T.LOPU_KUUP IS NULL
                              OR T.LOPU_KUUP >=SYSDATE));
```

Väljastatakse kõikide töötajate eesnimed ja perekonnanimed, kellel on kehtiv (lõppemata) tööleping. Lõpetamata on need töölepingud, mille lõpetamise kuupäev on kas tühi või on see tuleviku kuupäev.