

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
Zavod za primijenjenu matematiku
Grupa Računarske znanosti

Slaven Zakošek

Uvod u baze podataka SQL

Materijali za auditorne vježbe

Sadržaj

1. OSNOVNI POJMOVI.....	3
1.1 SUSTAV ZA UPRAVLJANJE BAZAMA PODATAKA	3
1.2 JEZICI U SUSTAVIMA ZA UPRAVLJANJE BAZAMA PODATAKA	3
1.3 FIZIČKA I LOGIČKA ORGANIZACIJA PODATAKA	4
1.4 KLIJENT-POSLUŽITELJ ARHITEKTURA	4
2. SQL STANDARD.....	5
3. SINTAKSA I OSNOVE UPORABE SQL JEZIKA.....	6
3.1 UVOD	6
3.2 OSNOVNI OBJEKTI U SQL JEZIKU.....	7
3.3 VELIKA I MALA SLOVA U SQL-U	7
3.4 OBAVLJANJE SQL NAREDBI U INTERAKTIVNOM OKRUŽENJU	7
3.4.1 <i>Separator</i>	7
3.4.2 <i>Komentari u SQL-u</i>	8
3.5 FORMAT NAREDBI.....	8
3.6 TIPOVI PODATAKA	8
3.7 NAREDBE ZA DEFINICIJU BAZE PODATAKA I RELACIJE	10
3.7.1 <i>CREATE DATABASE</i>	10
3.7.2 <i>CREATE TABLE</i>	11
3.7.3 <i>Privremena relacija - CREATE TEMP TABLE</i>	12
3.7.4 <i>DROP TABLE</i>	12
3.7.5 <i>NULL vrijednosti</i>	12
3.8 ČETIRI OSNOVNE SQL NAREDBE ZA UPRAVLJANJE PODACIMA	13
3.9 NAREDBE ZA PRIJENOS PODATAKA U/IZ DATOTEKE OPERACIJSKOG SUSTAVA.....	13
4. SINTAKSA I NAČIN UPORABE NAREDBE SELECT	15
4.1 JEDNOSTAVNI UPITI - SELECT - FROM	15
4.2 IZRAZ (EXPRESSION)	17
4.2.1 <i>Atributi, konstante, aritmetički i znakovni operatori, funkcije</i>	17
4.2.2 <i>Datumske funkcije u SQL-u</i>	23
4.3 DEFINIRANJE UVJETA DOHVATA	24
4.3.1 <i>Uvjeti za dohvat i uvjeti usporedbe</i>	24
4.3.2 <i>Primjeri korištenja složenijih izraza (Expression) u uvjetima za usporedbu i listi za selekciju</i>	27
4.4 AGREGATNE FUNKCIJE	29
4.5 SPAJANJE RELACIJA	31
4.5.1 <i>Kartezijev produkt relacija</i>	31
4.5.2 <i>Prirodno spajanje relacija</i>	31
4.5.3 <i>Način spajanja relacija u slučaju postojanja "paralelne" veze</i>	33
4.5.4 <i>Način spajanja relacija u slučaju postojanja "refleksivne" veze</i>	33
4.5.5 <i>Spajanje relacija uz uvjet</i>	34
4.6 SQL NAREDBE KOJE SADRŽE UVJETE S PODUPITOM.....	35
4.6.1 <i>Korelirani podupiti</i>	38
4.6.2 <i>Kombiniranje uvjeta s podupitima</i>	38
4.6.3 <i>Podupiti u listi za selekciju</i>	40
4.6.4 <i>Podupite ne treba koristiti bez razloga</i>	40
4.6.5 <i>Određivanje presjeka i razlike relacija pomoću SELECT naredbe</i>	41
4.7 GRUPIRANJE REZULTATA.....	43
4.8 POSTAVLJANJE UVJETA NAD GRUPOM ZAPISA.....	45
4.8.1 <i>Ispitivanje postojanja funkcijske zavisnosti na temelju trenutnog sadržaja relacije</i>	46
4.9 POREDAK REZULTATA	47
4.10 POHRANA REZULTATA UPITA U PRIVREMENU RELACIJU	48
4.11 ODREĐIVANJE UNIJE RELACIJA POMOĆU NAREDBE SELECT	48

5. INDEKSI U JEZIKU SQL.....	49
6. INSERT NAREDBA.....	51
6.1 NAČINI KORIŠTENJA INSERT NAREDBE.....	51
6.2 INSERT NAREDBA I SERIAL TIP PODATKA.....	52
7. DELETE NAREDBA	52
8. UPDATE NAREDBA	53
9. VIEW (POGLED, VIRTUALNA RELACIJA).....	55
10. NULL VRIJEDNOSTI.....	56
10.1 IS NULL OPERATOR	56
10.2 UVJETI SELEKCIJE I SPAJANJA U PRISUSTVU NULL VRIJEDNOSTI	56
10.3 AGREGATNE FUNKCIJE I NULL VRIJEDNOSTI.....	56
10.4 GRUPIRANJE I NULL VRIJEDNOSTI.....	57
10.5 PODUPITI I NULL VRIJEDNOSTI.....	58
10.6 POREDAK REZULTATA UPITA U PRISUSTVU NULL VRIJEDNOSTI	58
10.7 NULL KONSTANTA	59
11. VANJSKO SPAJANJE	59
11.1 RAZLIKA IZMEĐU UVJETA SPAJANJA I UVJETA SELEKCIJE U UPITIMA S VANJSKIM SPAJANJEM.....	61
11.2 VANJSKO SPAJANJE S TRI ILI VIŠE RELACIJA.....	63
DODATAK A: PROGRAMSKI ALAT RELATIONAL OBJECT MANAGER.....	67
A.1. POKRETANJE SQL EDITORA U PROGRAMSKOM ALATU ROM.....	67
A.2. EDITIRANJE I OBAVLJANJE SQL NAREDBI	68
DODATAK B: SQL NAREDBE ZA KREIRANJE PROBNE BAZE PODATAKA	70
DODATAK C: E-R DIJAGRAM PROBNE BAZE PODATAKA	71

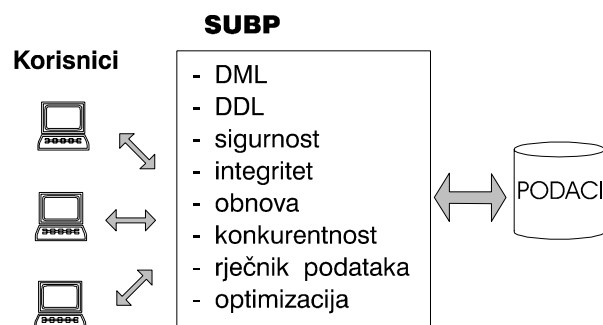
1. Osnovni pojmovi

1.1 Sustav za upravljanje bazama podataka

Baza podataka je skup međusobno povezanih podataka, pohranjenih zajedno bez štetne ili nepotrebne (nekontrolirane) zalihosti (redundancije), s ciljem da ih koriste različite aplikacije. Podaci su pohranjeni u obliku neovisnom od programa koji ih koriste. Unos, izmjena i dohvat podataka obavlja se ISKLJUČIVO kroz zajedničko i kontrolirano sučelje.

Programski sustav koji omogućava upravljanje bazom podataka je **sustav za upravljanje bazama podataka, SUBP** (DBMS - Database Management System).

Korisnik ili korisnički program postavlja zahtjev za obavljanjem neke operacije s podacima, a SUBP ga analizira, provjerava, optimizira, transformira u niz operacija koje je potrebno obaviti na fizičkoj razini, obavlja operacije i vraća rezultat.



Najvažnije zadatke SUBP-a su:

- zaštititi bazu podataka od neovlaštenog korištenja (data security)
- spriječiti narušavanja pravila integriteta (data integrity)
- osigurati obnovu podataka u slučaju uništenja (recovery)
- spriječiti štetne interferencije korisnika u višekorisničkim sustavima (concurrency)
- omogućiti korištenje rječnika podataka (podaci o podacima)
- optimizirati sve funkcije i obavljati ih efikasno koliko je to moguće

1.2 Jezici u sustavima za upravljanje bazama podataka

Neophodni dijelovi SUBP su jezik za definiciju podataka i jezik za upravljanje podacima:

- DDL (data description language)** - jezik za definiciju ili deklaraciju objekata u bazi podataka. Objekti su relacije (tablice), indeksi, pogledi (view), itd.
- DML (data manipulation language)** - omogućava manipulaciju objektima baze podataka. Osnovne su operacije: dohvat podataka, unos, izmjena i brisanje.

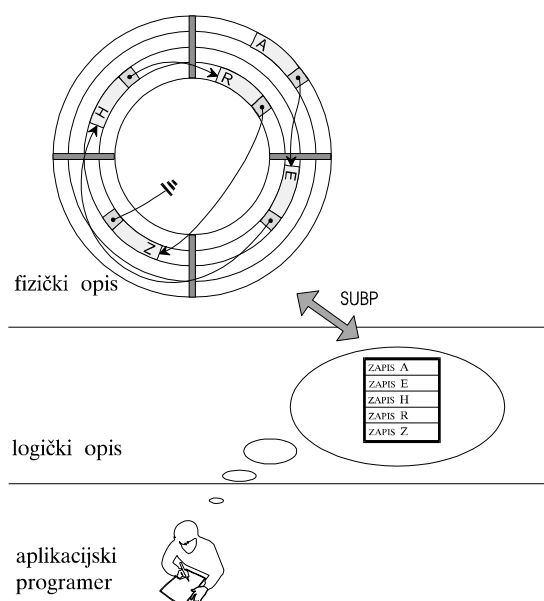
Ovi se jezici mogu upotrebljavati na različite načine:

1. Obavljanjem DML i DDL naredbi uz pomoć interaktivnih alata - korisnik uz pomoć tekst editora zadaje naredbu koja se odmah obavlja. Rezultat obavljanja naredbe ispisuje se na ekranu, zapisuje u datoteku ili ispisuje na pisaču.

2. Ugrađivanjem DML i DDL naredbi unutar "jezika domaćina" (*host language*). Naredbe se ugrađuju u kod jezika treće ili četvrte generacije, npr. FORTRAN, COBOL, C, C++, Java, IBM Informix 4GL, itd. Prije prevođenja takvog programskog koda standardnim prevoditeljem dotičnog jezika, izvorni programski kod se predprocesorom transformira tako da se ugrađene DDL i DML naredbe transformiraju u pozive bibliotečnih funkcija koje su posebno pripremljene za pojedini jezik domaćin.
3. Pomoću korisničkih sučelja koja omogućavaju upotrebu DML i DDL jezika u grafičkom obliku. Ovaj je način pogodan za korisnike koji nisu specijalisti u korištenju DML i DDL jezika, ali najčešće se na taj način znatno ograničava ekspresivnost i mogućnosti jezika, te je za razvoj svih, osim trivijalnih aplikacija, ipak nužno dobro poznavanje DML i DDL jezika.

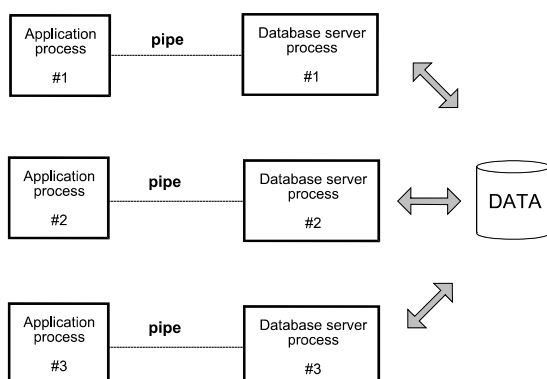
1.3 Fizička i logička organizacija podataka

Važna posljedica primjene SUBP jest razdvajanje fizičke i logičke organizacije podataka. Dok logička organizacija podataka predstavlja organizaciju sa stanovišta korisnika baze podataka ili programera te je koncentrirana na vrste podataka i njihove međusobne logičke veze, fizička organizacija predstavlja organizaciju fizičke pohrane podataka unutar računala. Oblik i organizacija pohranjenih podataka tu su često potpuno različiti od njihovog logičkog oblika i organizacije. U okviru toga, zadaća je SUBP-a omogućiti korisniku (programeru) manipuliranje podacima uz poznavanje samo logičkog opisa baze podataka, a ne nužno i poznavanja načina fizičke pohrane podataka.



1.4 Klijent-poslužitelj arhitektura

Jedno od važnih svojstava današnjih sustava za upravljanje bazama podataka jest arhitektura u kojoj se odvojeno promatraju procesi koji obavljaju ukupnu komunikaciju s korisnikom (*front-end-process*, *application process*) i procesi koji upravljaju podacima u bazi podataka (*back-end process*, *database process*). Najčešće se i na tržištu programskih sustava programski alati dijele na one namijenjene komunikaciji s korisnikom (*front-end tools*) i alate poslužitelje (*back-end tools*, *servers*).



Uobičajeno je da *front-end* i *back-end* procesi komuniciraju pomoću neimenovanih cjevovoda (*unnamed pipes*) ili zajedničke memorije (*shared memory*) ako se nalaze na istom računalu, odnosno uz pomoć nekog od mrežnih protokola (npr. TCP/IP) ukoliko se nalaze na različitim računalima. Koriste se standardna programska sučelja kao

što su ODBC (*Open Database Connectivity*) i JDBC (*Java Database Connectivity*). Time se, u velikoj mjeri, postiže nezavisnost alata koji se koriste za korisničko sučelje od konkretno primijenjenog sustava za upravljanje bazama podataka. Npr. korisničko sučelje izrađeno u Microsoft Access-u (tipični predstavnik alata za izgradnju korisničkog sučelja), može (uz relativno male teškoće) pristupiti Oracle, IBM Informix ili nekom drugom sustavu za upravljanje bazama podataka.

2. SQL standard

SQL predstavlja akronim za Structured Query Language. Originalno je razvijen 70-tih godina u *IBM Research Laboratory at San Jose - California* u okviru relacijskog sustava za upravljanje bazama podataka *System R*. SQL je nakon toga standardiziran i postao je referentni jezik za relacijske baze podataka.

SQL je upitni jezik temeljen na relacijskoj algebri i predikatnom računu. Važna značajka jezika je neproceduralnost - opisuje se **što** se želi dobiti kao rezultat, ali ne i **kako** se do tog rezultata dolazi. Ugrađeni optimalizator upita pronalazi najefikasniji način obavljanja upita. SQL se koristi kao **programski jezik i interaktivni upitni jezik**. Kao programski jezik može se ugrađivati u jezike treće i četvrte generacije.

SQL objedinjuje funkcije jezika za definiciju podataka (Data Definition Language - DDL) i jezika za upravljanje podacima (Data Manipulation Language - DML). Zadaća SQL-a je omogućiti definiciju podataka, upravljanje podacima i provođenje kontrole nad podacima u relacijskoj bazi podataka.

Primjer SQL naredbe iz DDL dijela jezika (kreiranje relacije **mjesto**):

```
CREATE TABLE mjesto
( pbr          INTEGER      NOT NULL
, nazMjesto    CHAR(40)    NOT NULL
, sifZupanija  SMALLINT
);
```

Primjer SQL naredbe iz DML dijela jezika (iz relacije **mjesto** dohvaća sve n-torke kojima je vrijednost atributa poštanski broj jednaka 31000):

```
SELECT * FROM mjesto
WHERE pbr = 31000
```

Uobičajena su dva načina izgovora akronima SQL:

- ☐ S-Q-L (s pojedinačno izgovorenim slovima)
- ☐ sequel (pod tim je nazivom jezik bio originalno poznat)

SQL je standard prema ANSI (American National Standards Institute), ISO (Organization for International Standardization), UNIX (X/Open), IBM Standard i FIPS (Federal Information Processing Standard).

Prva verzija SQL standarda je ANSI X3.135-1986, poznata pod nazivom SQL-86. Standard obuhvaća osnovne operacije s relacijama, definiciju pogleda (view) i nekih pravila integriteta i ugrađivanje SQL naredbi u jezike treće generacije (FORTRAN, COBOL, C, itd.). Standard je proširen 1989. godine, ANSI X3.135-1989. (djelomična definicija referencijskog integriteta). Standard iz 1989. godine poznat je pod nazivom SQL-89.

Sljedeća verzija standarda objavljena je 1992. godine, ANSI X3.135-1992, poznata pod nazivom SQL-92, ali također i pod nazivom SQL-2. Standard je donio veliki broj novih elemenata jezika (npr. pravila integriteta definirana na nivou relacije ili n-torke, DATETIME tip podatka, upravljanje transakcijama, privremene relacije, itd.). Najnoviji standard, poznat pod nazivom SQL-99 ili SQL-3 sadržava nove mogućnosti poput ugradnje aktivnih pravila integriteta i okidača, rekurzivnih operacija, ugradnje novih, korisnički definiranih tipova podataka, itd.

U komercijalno raspoloživim sustavima za upravljanje bazama podataka SQL-2 standard je ovog trenutka najšire prihvaćen. Ipak, zbog njegove složenosti do sada još niti jedan sustav ne podržava taj standard u potpunosti. SQL-92 standard definira tri kategorije: *Entry SQL*, *Intermediate SQL* i *Full SQL*. Entry SQL kategorija je slična SQL-89 standardu. *Intermediate SQL* pokriva mogućnosti standarda koje su najznačajnije u primjeni u danas raspoloživim komercijalnim produktima.

Proizvođači komercijalnih sustava također ugrađuju i svoje, uglavnom nestandardne, DDL i DML naredbe. Ti su nestandardni dijelovi problematični jer programski kod postaje neprenosiv između različitih SQL sustava, a također se bitno otežava usaglašavanje oko budućih standarda.

3. SINTAKSA I OSNOVE UPORABE SQL JEZIKA

3.1 Uvod

Sintaksa SQL jezika u ovim materijalima nije pokrivena u punom opsegu. Opisani su samo dijelovi koji, prema mišljenju autora, predstavljaju najvažnije koncepte SQL jezika i koji su predmet proučavanja u okviru ovog kolegija. Sintaksni dijagrami tih SQL naredbi mogu se pronaći na

<http://www.zpm.fer.hr/courses/idb/sqlsyntax/index.html>

Ovi se nastavni materijali temelje na SQL-92 standardu (*Entry Level*), ali budući da se u primjerima koristi konkretan sustav za upravljanje bazama podataka (IBM Informix Dynamic Server), bilo je potrebno u rijetkim slučajevima dodati opise nestandardnih proširenja jezika. Mjesta u tekstu gdje se koriste takva, nestandardna rješenja, jasno su označena.

Gdje je bilo moguće, SQL naredba je opisana na najjednostavniji mogući način, tako da su ključne riječi napisane jačim otiskom, a "varijabilni" je dio naredbe otisnut kosim slovima, kao npr. u naredbi:


DROP DATABASE *databaseName*

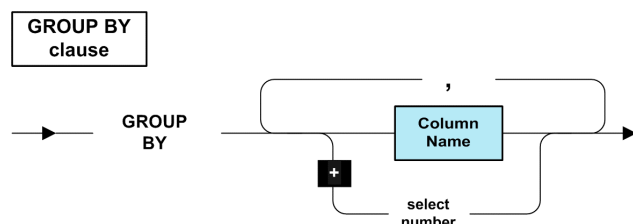
Ovo znači da se konkretna naredba (za uništavanje baze podataka) formira tako da se zamijeni *databaseName* sa stvarnim imenom neke baze podataka, npr.:

DROP DATABASE *mojaBazaPodataka*

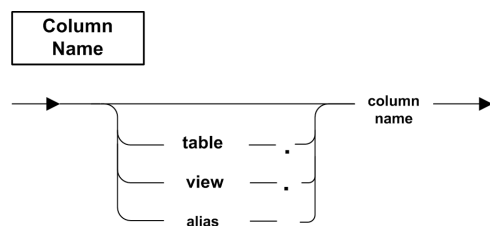
U većini slučajeva SQL naredbe imaju složeniju sintaksu koju nije moguće opisati na gore navedeni način. Koriste se sintaksni dijagrami u obliku prikazanom u sljedećem primjeru:

Sintaksni dijagram je usmjereni graf kojim su opisani dopušteni oblici naredbe ili nekog segmenta naredbe (*Clause*). Dopusšten je svaki oblik koji se može formirati obilaskom grafa polaskom s lijeve strane grafa uz obavezno pridržavanje pravila da se smjer obilaska grafa ne smije mijenjati.

Oznaka  u nekoj grani grafa značit će da grana grafa opisuje proširenje ANSI SQL-92 *entry-level* sintakse. Ova oznaka će se koristiti i u tekstu, kako bi se naglasilo da se radi o nestandardnom proširenju sintakse.



Sintaksni dijagram može sadržavati poddijagrame. Poddijagrami su označeni pravokutnicima. U prethodnom dijagramu postoji npr. poddijagram za *Column Name*, a taj je dijagram opisan sljedećom slikom:



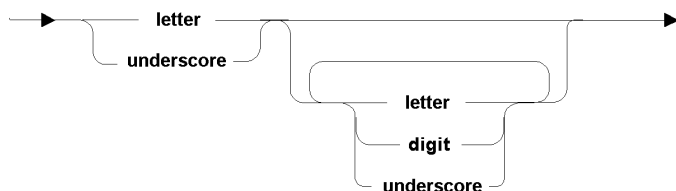
3.2 Osnovni objekti u SQL jeziku

Osnovni objekti u SQL-u su:

- Baza podataka	(Database)
- Tablica	(Table) - relacija
- Stupac	(Column) - atribut, kolona, stupac
- Pogled ili virtualna tablica	(View)
- Sinonim	(Synonym)
- Pravilo integriteta	(Constraint)
- Indeks	(Index)
- Pohranjena procedura	(Stored Procedure)
- Okidač	(Trigger)

Imena objekata (identifikatori) u SQL-u formiraju se iz slova, znaka '_' i znamenki. Prvi znak od ukupno 128 značajnih (signifikantnih) znakova mora biti slovo.

Sintaksni dijagram koji pokazuje na koji se način može formirati identifikator prikazan je na sljedećoj slici:



Primjer:

ispravno formirani identifikatori: **stud ispiti2000godine stud_ispit _1mjesec**

neispravno formirani identifikatori: **_11.mjesec 11mjesec stud-ispit *mjesec**

3.3 Velika i mala slova u SQL-u

SQL je "neosjetljiv" na razliku između velikih i malih slova (*case insensitive*). To znači da su sljedeće dvije naredbe jednake:

```
select *
  From MJESTO
 where NAzmJESTO = 'Dubrovnik'
```

```
SELECT *
  FROM mjesto
 WHERE nazMjesto = 'Dubrovnik'
```

Radi preglednosti, uobičajeno je ključne riječi pisati velikim slovima, a ostale elemente naredbe malim slovima (prva naredba u prethodnom primjeru je u tom smislu izrazito nepregledno napisana).

Ipak, važno je uočiti da **postoji razlika** među nizovima znakova napisanim malim i velikim slovima. Sljedeće dvije SQL naredbe NISU jednake:

```
SELECT *
  FROM mjesto
 WHERE nazMjesto = 'Dubrovnik'
```

```
SELECT *
  FROM mjesto
 WHERE nazMjesto = 'DUBROVNIK'
```

3.4 Obavljanje SQL naredbi u interaktivnom okruženju

3.4.1 Separator

Ukoliko se u interaktivnom okruženju želi izvesti jedna za drugom dvije ili više naredbi, kao separator među naredbama koristi se znak točka-zarez, ";"

Npr. naredba a;
 naredba b; ...

3.4.2 Komentari u SQL-u

- jedan ili više redaka omeđenih vitičastim zagradama { }
 { ovo je komentar
 koji se slobodno proteže kroz više redova teksta }
- mjesto u retku na kojem se nalaze znakovi -- predstavlja početak komentara
 SELECT * FROM -- ovo je komentar
 tekst nakon znakova -- predstavlja komentar koji se proteže samo do kraja retka

3.5 Format naredbi

SQL je jezik slobodnog formata naredbi. Sljedeće naredbe su ispravno napisane:

```
SELECT *  
FROM mjesto  
WHERE pbr = 31000
```

```
SELECT *  
FROM mjesto  
WHERE pbr = 31000
```

3.6 Tipovi podataka

INTEGER Cijeli broj pohranjen u 4 byte-a. Alternativni naziv je **INT**. Na raspolaganju je n bitova (n=32). Dopusćeni raspon brojeva se računa pomoću sljedećeg izraza


$$-(2^{n-1}) \rightarrow 2^{n-1} - 1$$


Prema izrazu, raspon brojeva bi trebao biti [-2147483648, 2147483647]. U stvarnosti je manji, jer se vrijednost -2147483648 koristi za pohranu posebne vrijednosti, tzv. NULL vrijednosti. Dopusćeni raspon brojeva je prema tome [-2147483647, 2147483647].

SMALLINT Cijeli broj pohranjen u 2 byte-a. Raspon brojeva koji se mogu prikazati je [-32767, 32767].

BOOLEAN Vrijednost suda (*true, false*). Dopusćene vrijednosti su 't' ili 'T', odnosno 'f' ili 'F'. Vrijednosti se mogu uspoređivati međusobno ili s logičkim izrazom. Za pohranu podatka koristi se 1 byte.

CHAR Znakovni niz (string). Npr.: CHAR(24). Maksimalna duljina niza znakova je 32767. Alternativni naziv tipa je **CHARACTER**.

NCHAR  Jednako kao i CHAR tip podatka, ali uz dodatak kojim se omogućava ispravno leksikografsko uređenje nizova znakova koji sadrže nacionalne kodne stranice (*character set*). Kad je npr. potrebno leksikografski poredati nizove znakova uz korištenu ISO8859-2 kodnu stranicu (kodna stranica koja se standardno koristi u hrvatskom jeziku) nužno je umjesto CHAR tipa podatka koristiti NCHAR tip.

DATE  Vrijednosti ovog tipa se uvijek prikazuju u obliku datuma. Format ispisa (npr. DD.MM.YYYY) određen je *environment* varijablom DBDATE. Interno se podatak predstavlja brojem dana proteklih od 31.12.1899. Zauzeće memorije je 4 byte-a. Oblik datumske konstante određen je varijablom DBDATE, i uvijek se piše pod navodnicima, npr. '14.5.1996'.

Ovaj tip podatka omogućava korištenje operacija zbrajanja i oduzimanja:

<code>datum1 - datum2</code>	rezultat je cijeli broj: broj dana proteklih između datum1 i datum2
<code>datum + broj</code>	rezultat je datum: koji će datum biti broj dana nakon dana datum
<code>datum - broj</code>	rezultat je datum: koji je datum bio broj dana prije dana datum

FLOAT	Odgovara double tipu u jeziku C, što je specifična karakteristika operacijskog sustava. Najčešće se radi o prostoru za pohranu veličine 8 byte-ova. Alternativni naziv je DOUBLE PRECISION .														
SMALLFLOAT	+ Ovaj tip podatka odgovara float tipu podatka u jeziku C. Raspon brojeva i preciznost ovisi o operacijskom sustavu - najčešće se radi o upotrebi 4 byte-a i tehnici prikaza floating point. Alternativni naziv je REAL . Konstante koje se mogu koristiti su oblika npr. 23 -1 -343.23 232.233E3 23.0e-3 itd.														
DECIMAL	Tip podatka DECIMAL je u posebnom formatu pohranjen (baza 100) decimalni broj. Pri definiciji ovog tipa podatka specificira se ukupan broj znamenki (<i>precision</i> , m ≤ 32) i broj znamenki koje se nalaze iza decimalne točke (<i>scale</i> , n ≤ m). Npr., DECIMAL (15, 3) predstavlja decimalni broj sa ukupno 15 znamenki, od toga se 3 nalaze iza decimalne točke. Ukoliko se <i>scale</i> ne navede, varijabla se smatra realnim brojem (floating real) koji ima preciznost m , a dopušteni interval vrijednosti definiran je s $[10^{-130}, 10^{124}]$. Zauzeće memorije ovisi o deklariranom ukupnom broju znamenki. Ako je deklarirani broj znamenki m , potreban prostor iznosi m /2 + 1 byte-ova. Alternativni nazivi su NUMERIC i DEC .														
MONEY	+ Tip podatka koji je gotovo identičan tipu DECIMAL. Razlika postoji u obliku ispisa - MONEY tip podatka implicira da se uz vrijednost varijable ispisuje i oznaka valute. Oznaka valute određena je <i>environment</i> varijablom DBMONEY.														
SERIAL	+ Generira niz cijelih brojeva, a po veličini i zauzeću memorije identičan je tipu INTEGER. Ukoliko se pri unosu n-torke za vrijednost atributa tipa SERIAL upiše vrijednost 0, SUBP sam generira prvi sljedeći broj u nizu (ne prvi slobodni, jer se "praznine" ne popunjavaju). Ukoliko se za vrijednost upiše vrijednost različita od 0, SUBP će upisati tu vrijednost. SERIAL tip podatka za atribut ne implicira da je taj atribut ključ relacije.														
DATETIME	Pohranjuje informaciju o trenutku u vremenu. Preciznost se može odrediti korištenjem sintakse DATETIME <i>max_kval</i> TO <i>min_kval</i> pri čemu se za kvalifikatore mogu odabrati: <table> <tr> <td>YEAR</td><td>- godina (1 do 9999 A.D.)</td></tr> <tr> <td>MONTH</td><td>- mjesec (1 do 12)</td></tr> <tr> <td>DAY</td><td>- redni broj dana u mjesecu (1 do 31)</td></tr> <tr> <td>HOURL</td><td>- sat (0-ponoć do 23)</td></tr> <tr> <td>MINUTE</td><td>- minuta (0 do 59)</td></tr> <tr> <td>SECOND</td><td>- sekunda (0 do 59)</td></tr> <tr> <td>FRACTION</td><td>- dijelić sekunde (preciznost do 5 znamenki)</td></tr> </table> <p>Varijabla koja je definirana kao DATETIME YEAR TO FRACTION(2) pohranjivat će informaciju o godini, mjesecu, danu, satu, sekundi i stotinkama sekunde. Oblik konstante koja se može koristiti u ovom slučaju je npr. DATETIME (1996-05-14 12:31:18.56) YEAR TO FRACTION</p>	YEAR	- godina (1 do 9999 A.D.)	MONTH	- mjesec (1 do 12)	DAY	- redni broj dana u mjesecu (1 do 31)	HOURL	- sat (0-ponoć do 23)	MINUTE	- minuta (0 do 59)	SECOND	- sekunda (0 do 59)	FRACTION	- dijelić sekunde (preciznost do 5 znamenki)
YEAR	- godina (1 do 9999 A.D.)														
MONTH	- mjesec (1 do 12)														
DAY	- redni broj dana u mjesecu (1 do 31)														
HOURL	- sat (0-ponoć do 23)														
MINUTE	- minuta (0 do 59)														
SECOND	- sekunda (0 do 59)														
FRACTION	- dijelić sekunde (preciznost do 5 znamenki)														
INTERVAL	+ Ovaj tip podatka namijenjen je pohrani informacije o trajanju ("količini vremena"). Postoje dvije osnovne grupe: <i>godina-mjesec</i> intervali i <i>dan-vrijeme</i> intervali. U <i>godina-mjesec</i> intervalima mogu se pohraniti intervali s granicama određenim godinom i mjesecom, dok se u <i>dan-vrijeme</i> intervalima mogu pohraniti intervali s granicama određenim danom, satom, minutom, sekundom i dijelom sekunde. Preciznost se može odrediti korištenjem sintakse INTERVAL <i>max_kval(n)</i> TO <i>min_kval(n)</i> . Kvalifikatori su isti kao u DATETIME tipu podatka. <i>n</i> smije biti maksimalno 9, osim za FRACTION kvalifikator, kada je najveća vrijednost tog kvalifikatora određena s 5.														
	Varijabla čiji je tip definiran kao INTERVAL DAY(4) TO SECOND pohranjivat će informaciju o trajanju vremena (najviše 9999 dana, 24 sata, 60 minuta i 60 sekundi). Oblik konstante koja se može koristiti u ovom slučaju je npr. INTERVAL (428 12:31:36) DAY TO SECOND i ima značenje trajanja od 428 dana, 12 sati, 31 minute i 36 sekundi.														

CHARACTER VARYING

Niz znakova varijabilne duljine. Mora se specificirati maksimalna duljina, a može se specificirati i minimalna duljina niza. Utrošak memorije ovisi o duljini stvarno upisanog niza znakova, ali u svakom slučaju se troši barem onoliko memorije koliko je bilo specificirano minimalnom duljinom niza. Maksimalna duljina niza je 255 znakova.

VARCHAR



Isto što i CHARACTER VARYING, ali nije u skladu sa SQL standardom

NVARCHAR



NVARCHAR je u odnosu na CHARACTER VARYING isto što i NCHAR u odnosu na CHAR. Vidjeti opis razlike između NCHAR i CHAR.

BYTE



Pripada u grupu BLOB tipova podataka. Korištenjem ovog tipa podatka može se pohraniti bilo koji neprekinuti niz binarnih podataka. Praktično ne postoji ograničenje u veličini podatka koji se može pohraniti. Utrošak memorije ovisi o veličini stvarno pohranjenog objekta.

TEXT



Vrlo sličan BYTE tipu podataka. Razlika je u tome što ovaj tip podatka dopušta korištenje samo "tekstualnih" podataka (*printable characters*, *tabs*, *newlines*, *newpages*), pa se koristi za pohranu tekstova. Utrošak memorije ovisi o veličini stvarno pohranjenog teksta.

3.7 Naredbe za definiciju baze podataka i relacije

3.7.1 CREATE DATABASE

Naredba oblika



```
CREATE DATABASE databaseName
```

namijenjena je za stvaranje nove baze podataka. Obavljanjem ove naredbe kreiraju se strukture podataka potrebne za opsluživanje baze podataka (*rječnik podataka*). Vlasnik kreirane baze podataka je korisnik koji je obavio naredbu.

Sljedećom naredbom kreira se baza podataka pod imenom **stuslu**.

```
CREATE DATABASE stuslu
```

Unutar jedne instance poslužitelja baze podataka ne mogu postojati dvije baze podataka istog imena. Ukoliko isti ili neki drugi korisnik pokuša još jednom kreirati bazu podataka pod istim imenom (a da pri tom prethodno kreirana baza podataka nije u međuvremenu uništena), dobit će poruku o pogrešci.

Ukoliko ima dozvolu, korisnik može pristupiti svakoj bazi podataka unutar instance sustava za upravljanje bazama podataka ("otvoriti bazu podataka") pomoću naredbe:



```
DATABASE databaseName  
ili
```



```
DATABASE databaseName EXCLUSIVE
```

Ukoliko se navede rezervirana riječ **EXCLUSIVE**, baza podataka je zaključana ekskluzivno. Niti jedan drugi proces ne može pristupiti bazi podataka sve dok proces koji je pristupio bazi podataka ekskluzivno, ne terminira ili obavi naredbu CLOSE DATABASE.



```
CLOSE DATABASE
```

Proces koji obavi ovu naredbu prekida vezu s trenutno "otvorenom" bazom podataka.

Komplement naredbe **CREATE DATABASE** je naredba



DROP DATABASE *databaseName*

Naredbom se **bespovratno** uništavaju podaci i cjelokupna struktura baze podataka *databaseName*. Npr.

DROP DATABASE *stuslu*

Ova se naredba ne može obaviti ukoliko je bilo koji od procesa trenutno spojen na dotičnu bazu podataka. Svi takvi procesi moraju prethodno ili terminirati ili obaviti naredbu **CLOSE DATABASE**.

3.7.2 CREATE TABLE

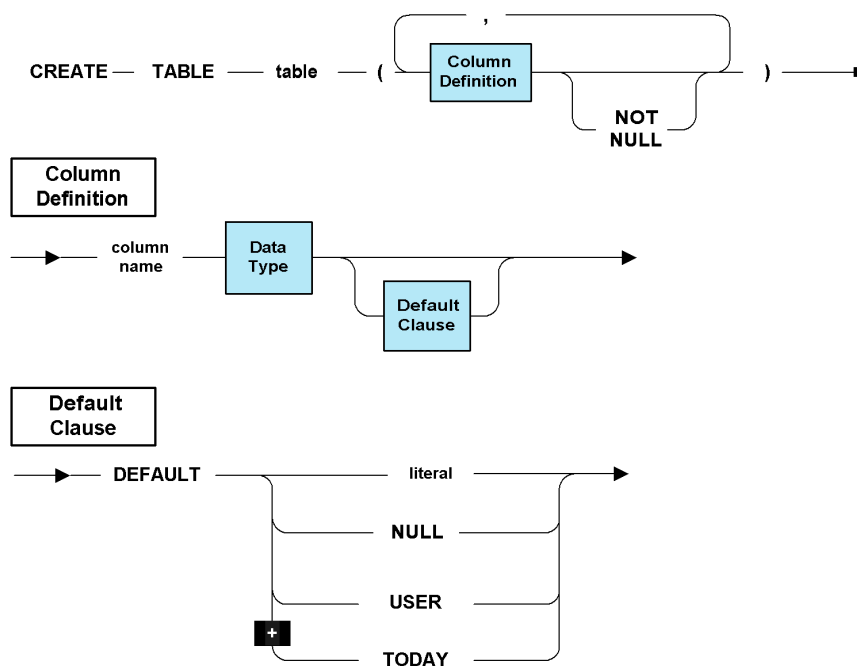
Pojednostavljeno, relacija je dvodimenzionalna tablica čiji retci predstavljaju n-torke, a stupci atributi (svojstva). Svakom n-torkom opisuje se jedan entitet iz stvarnog svijeta.

Primjer: *mbrStud*, *prezStud*, *imeStud*, *prosJOcjena* i *pbrRodStud* su atributi relacije **stud**. Svakim retkom relacije opisana su svojstva jednog studenta.

<u>mbrStud</u>	<u>prezStud</u>	<u>imeStud</u>	<u>prosJOcjena</u>	<u>pbrRodStud</u>
12345	Horvat	Ivan	3.76	10000
12346	Kolar	Marko	4.34	51000
12347	Novak	Berislav	3.39	20000
12348	Horvat	Ivica	4.56	31000

Svaka relacija ima svoj ključ. To je atribut ili skup atributa čije vrijednosti u relaciji moraju biti jedinstvene, odnosno u relaciji se ne smiju pojaviti dvije n-torke koje imaju jednaku vrijednost ključa. U relaciji može postojati više ključeva, ali se jedan od ključeva uvijek proglašava primarnim ključem.

Relacija (njezina shema i sadržaj) u SQL-u se predstavlja objektom koji se naziva tablica (*table*). Struktura svake relacije (tablice) mora biti jednom definirana uz pomoć naredbe **CREATE TABLE**. Osnovna, donekle pojednostavljena sintaksa naredbe je:



Pri definiranju relacije, potrebno je za svaki atribut (*Column Definition*) specificirati naziv (*column name*), tip podatka (*Data Type*), te opcionalno navesti **NOT NULL** kvalifikator. **NOT NULL** kvalifikatorom specificiraju se atributi čije vrijednosti u n-torki ne smiju poprimiti NULL vrijednost. Koncept NULL vrijednosti opisan je u posebnom poglavlju o NULL vrijednostima.

Sintaksni dijagram za *Data Type* nije naveden jer su dopušteni tipovi podataka već opisani.

Opcionalno je uz definiciju svakog atributa relacije moguće zadati tzv. *default* vrijednost. To je vrijednost koja se u trenutku upisa n-torke u relaciju pridružuje atributu ukoliko vrijednost za atribut nije eksplicitno navedena.

Oznaka *literal* u sintaksnom dijagramu *Default Clause* predstavlja konstantu (znakovnog, numeričkog ili datumskog tipa) koja po tipu mora odgovarati tipu atributa kojem se pridružuje. USER i TODAY su "variable" koje sadrže tekući datum (vrijednost trenutno postavljenu na nivou operacijskog sustava) i ime (*login*) korisnika.

Primjer:

```
CREATE TABLE stud
(  mbrStud      INTEGER                NOT NULL
  , prezStud    CHAR(25)               NOT NULL
  , imeStud     CHAR(25)
  , prosjOcjena DECIMAL(3,2)  DEFAULT 0.0  NOT NULL
  , pbrRodStud  INTEGER              DEFAULT 10000  -- 10000 Zagreb
)
```

3.7.3 Privremena relacija - CREATE TEMP TABLE

+ Privremena relacija ima sve karakteristike trajne relacije kreirane naredbom CREATE TABLE, osim što se ne vidi u rječniku podataka, te biva uništena u trenutku kada završava program u kojem je kreirana (npr. kad se napusti SQL Editor). Privremena relacija se također može uništiti i uporabom naredbe DROP TABLE. Privremena relacija se koristi prvenstveno za pohranu međurezultata, u slučajevima kad je nepraktično ili nemoguće obaviti operaciju bez dijeljenja te operacije u nekoliko odvojenih koraka.

Naredba kojom se kreira privremena relacija vrlo je slična "običnoj" CREATE TABLE naredbi (potrebno je samo dodati ključnu riječ TEMP). Za privremenu relaciju postoje određena ograničenja u odnosu na običnu relaciju (npr. ne mogu se definirati strani ključevi).

Primjer:

```
CREATE TEMP TABLE privMjesto
(  pbr          INTEGER                NOT NULL
  , nazMjesto   CHAR(20)              NOT NULL
  , sifZupanija CHAR(2)               DEFAULT 'ZG'
)
```

Privremena relacija se može kreirati i uz pomoć SELECT naredbe, ali o tome će biti riječi u posebnom poglavlju o pohrani rezultata SELECT naredbe u privremenu relaciju.

3.7.4 DROP TABLE

Naredba **DROP TABLE** je komplement naredbe **CREATE TABLE**. Njome se bespovratno uništavaju podaci i struktura relacije ili privremene relacije *tableName*.

+

```
DROP TABLE tableName
```

3.7.5 NULL vrijednosti

Informacija koja se pohranjuje u bazu podataka često je nekompletna. Vrijednost atributa za neku n-torku može biti privremeno ili trajno nedostupna (čeka se na unos podatka, dobivanje potrebne informacije je preskupo, informacija ne postoji).

Specifična oznaka za takve vrijednosti je NULL (to nije i način fizičke prezentacije NULL vrijednosti). Sustav mora biti u stanju razlikovati NULL vrijednosti od ostalih vrijednosti istog tipa podatka (od onih za koje je vrijednost atributa poznata).

NOT NULL uz definiciju atributa **A** u relaciji **R** predstavlja pravilo prema kojem nije dopušteno obaviti operaciju koja bi u bilo kojoj n-torki relacije **R** postavila za vrijednost atributa **A** NULL vrijednost (odnosno, zahtijeva se da je vrijednost atributa **A** bezuvjetno definirana za svaku n-torku).

Korištenje NULL vrijednosti zahtijeva pažnju jer se u prisustvu NULL vrijednosti uvode nova pravila pri evaluiranju relacijskih, logičkih, aritmetičkih i ostalih izraza.

3.8 Četiri osnovne SQL naredbe za upravljanje podacima

U ovom su poglavlju navedeni najjednostavniji oblici osnovnih SQL naredbi.

SELECT naredba

Postavljanje SQL upita (*query*) nad bazom podataka (pregled podataka).

Primjer:

```
SELECT nazMjesto
FROM mjesto
WHERE pbr = 10000 OR pbr = 51000
```

nazMjesto je naziv atributa čija se vrijednost želi vidjeti u izlaznoj listi. *mjesto* je naziv relacije čije se n-torke pregledavaju. *pbr = 10000 OR pbr = 51000* je *predikat* kojim se određuje koje su n-torke kvalificirane za izlaznu listu - rezultat. Slobodno se upotrebljavaju konstante, okrugle zgrade, funkcije i operatori. Konverzije tipova se obavljaju automatski u svim situacijama u kojima je to moguće.

INSERT naredba

Unos n-torke (zapis, redak, *row*) u relaciju.

```
INSERT INTO mjesto
VALUES (31000, 'Osijek')
```

UPDATE naredba

Ažuriranje n-torke (n-torki) u relaciji.

```
UPDATE mjesto
SET nazMjesto = 'OSIJEK'
WHERE pbr = 31000
```

DELETE naredba

Brisanje n-torke (n-torki) iz relacije.

```
DELETE FROM mjesto
WHERE pbr = 31000
```

3.9 Naredbe za prijenos podataka u/iz datoteke operacijskog sustava

```
51000#Rijeka#           ← Primjer sadržaja "ASCII" datoteke
10000#Zagreb#
```

Prijenos podataka iz relacije u datoteku u ASCII formatu obavlja se naredbom **UNLOAD**. Opći oblik naredbe **UNLOAD** je sljedeći:

UNLOAD



filename je ime datoteke ili kompletni *path* u koji se pohranjuje ASCII ekvivalent podataka iz baze. Vrijednosti atributa (i izraza) koji su navedeni u listi za selekciju ispisat će se u navedenom redoslijedu u zadanu datoteku. Vrijednosti pojedinih atributa (izraza) odijeljeni su znakom za razgraničenje (*delimiter*) koji je definiran eksplicitno u naredbi ili pomoću *environment* varijable *DBDELIMITER*.

Primjer:

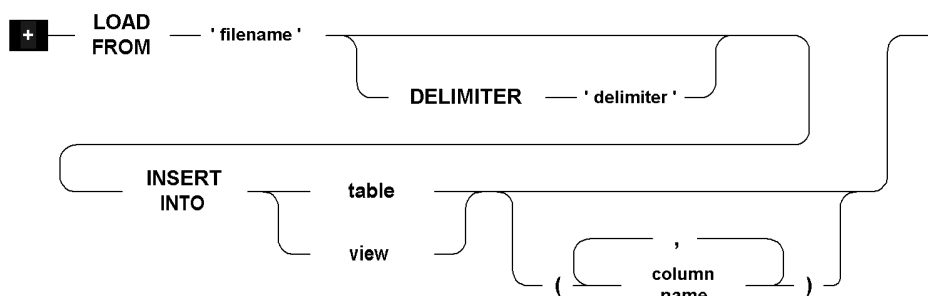
```
UNLOAD TO 'mjesto.unl' DELIMITER '#' SELECT * FROM mjesto
```

```
UNLOAD TO 'c:\temp\mjesto.unl' SELECT mbrStud, imeStud FROM stud
```

Podaci se u ASCII datoteku mogu ispisati u bilo kojem obliku koji se može specificirati u **SELECT** naredbi, pa se mogu upotrijebiti i vrlo složeni oblici **SELECT** naredbe.

Za prebacivanje podataka iz datoteke u ASCII formatu u relaciju u bazi podataka, koristi se naredba **LOAD**. **LOAD** naredba je komplementarna naredbi **UNLOAD**. Opći oblik naredbe **LOAD** je:

LOAD



Primjer:

```
LOAD FROM 'mjesto.unl' DELIMITER '#' INSERT INTO mjesto
```

```
LOAD FROM '/usr1/stud.unl' INSERT INTO stud (mbrStud, prezStud)
```

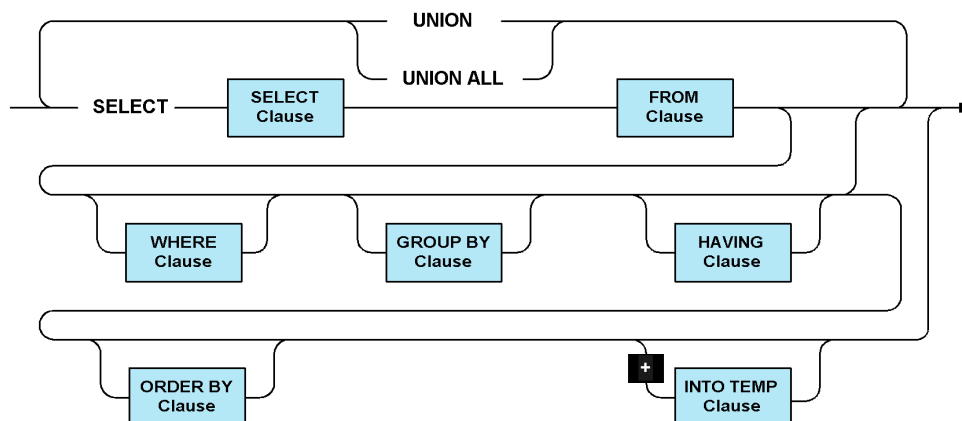
U drugoj naredbi pretpostavljeno je da se kao delimiter koristi znak definiran pomoću *environment* varijable *DBDELIMITER* (određene na nivou operacijskog sustava), te da se u toj datoteci nalaze redom vrijednosti za matični broj studenta i prezime studenta. Ostalim atributima u relaciji **stud** se prilikom punjenja podataka pridružuju **NULL** vrijednosti ili vrijednosti definirane s *DEFAULT* opcijom.

Napomena: često se u sintaksnim dijagramima pojavljuje opcija *Table Name* ili *View Name*. *View* ili pogled je objekt sličan relaciji koji se gradi na temelju relacije ili nekog drugog pogleda.

4. Sintaksa i način uporabe naredbe SELECT

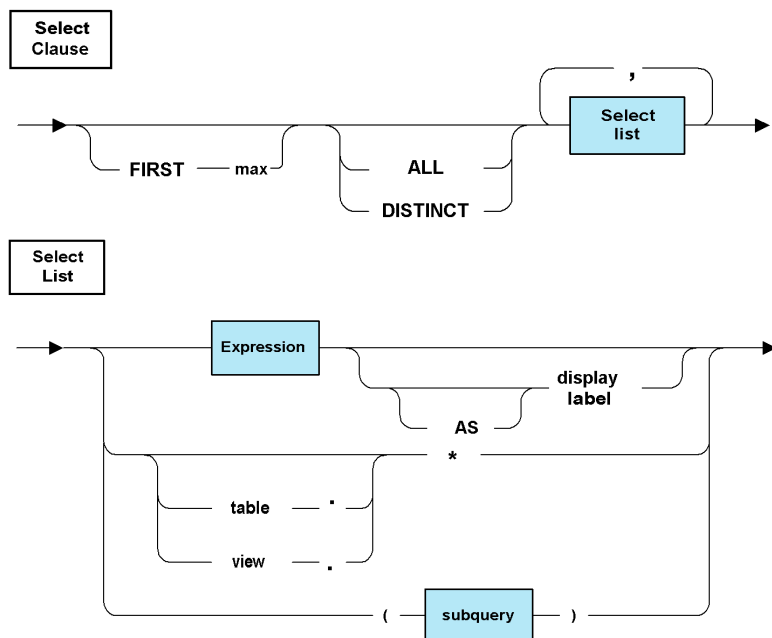
Prema prikazanom sintaksnom dijagramu svi dijelovi osim *SELECT Clause* i *FROM Clause* su opcionalni. Pojedini dijelovi naredbe će biti postupno opisani u sljedećim poglavljima.

SELECT



4.1 Jednostavni upiti - SELECT - FROM ...

Minimalni oblik SELECT naredbe sadrži *SELECT Clause* i *FROM Clause*. *SELECT Clause* sadrži opcionalni kvalifikator *FIRST max*, opcionalni kvalifikator *ALL* ili *DISTINCT*, te listu atributa ili izraza (*Select list*, tj. lista za selekciju) čije se vrijednosti trebaju pojaviti u izlaznoj listi rezultata. U *From Clause* navodi se ime relacije (tablice) iz koje se dohvaćaju podaci.



Grana dijagrama u kojoj se nalazi *subquery* poddijagram će biti objašnjena u posebnom poglavlju o podupitima.

U sljedećem primjeru prikazana je SELECT naredba kojom se dohvaćaju vrijednosti atributa **mbrStud**, **imeStud** i **prezStud** za sve n-torke relacije **stud**

```
SELECT mbrStud, imeStud, prezStud
FROM stud
```

U prethodnom primjeru je u listi za selekciju, na mjestu izraza (*expression*), upotrebljeno ime atributa iz relacije koja je navedena u FROM listi. *Expression* može biti vrlo jednostavan (npr. samo ime atributa), ali može biti i vrlo složeni izraz koji se sastoji od konstanti, naziva atributa, aritmetičkih operatora i različitih funkcija. Detaljniji opis izraza (*Expression*) nalazi se u sljedećem poglavlju.

Ispred imena atributa uvijek se **može** navesti ime relacije, ali se ime relacije **mora** navoditi ispred imena atributa **jedino** u slučajevima kad ime atributa nije jednoznačno određeno unutar liste za selekciju. Nejednoznačnost se može pojaviti tek ukoliko se podaci dohvaćaju iz više relacija (*join*), te je u sljedećem primjeru navođenje imena relacije suvišno (ali ne i pogrešno, tj. neće uzrokovati pogrešku).

```
SELECT stud.pbrStan
FROM stud
```

Select List može sadržavati oznaku "*" ili **imeRelacije**. * što znači da će se u izlaznoj listi pojaviti vrijednosti svih atributa iz relacije. Na taj način se izbjegava potreba taksativnog navođenja svih atributa relacije svaki put kad je potrebno ispisati vrijednosti svih atributa. Tako je na primjer

```
SELECT * FROM stud      ili      SELECT stud.* FROM stud
```

ekvivalentno sa

```
SELECT mbrStud
      , imeStud
      , prezStud
      , pbrRod
      , pbrStan
      , datRodStud
      , jmbgStud FROM stud
```

Uz svaki atribut moguće je navesti tzv. *alias* ime, koje će se u izlaznoj relaciji pojaviti kao naziv stupca (atributa). Ukoliko se *alias* ime ne navede, naziv stupca određuje se prema nazivu atributa u rječniku podataka. *Alias* ime služi isključivo kao "naziv atributa" u rezultatnoj relaciji, te se ne može koristiti unutar upita osim u *ORDER BY Clause*. Valja napomenuti i na ovom mjestu da je drugačija situacija s *alias* imenima relacija, koja se mogu slobodno koristiti unutar upita, što će biti prikazano u poglavlju o spajanju relacija.

```
SELECT mbrStud MATICNI_BROJ,
      prezStud PREZIME,
      imeStud IME
FROM stud

ili

SELECT mbrStud AS MATICNI_BROJ,
      prezStud AS PREZIME,
      imeStud AS IME
FROM stud
```

Pomoću kvalifikatora *FIRST max* može se rezultat SELECT naredbe reducirati na prvih *max* n-torki koje bi sadržavao rezultat SELECT naredbe bez tog kvalifikatora. Ukoliko je broj n-torki koje bi se bez tog kvalifikatora pojavile u rezultatu jednak *k*, te je $k < max$, rezultat će sadržavati *k* n-torki.

Npr, sljedećim upitima prikazuje se prvih dvadeset n-torki iz relacije (u prvom primjeru samo atributi navedeni u listi za selekciju, a u drugom primjeru svi atributi relacije).

```
SELECT FIRST 20 mbrStud, prezStud, imeStud FROM stud

SELECT FIRST 20 * FROM stud
```

Ukoliko se navede kvalifikator *DISTINCT*, u izlaznoj listi rezultata neće biti višestrukih pojava istih vrijednosti n-torki. Npr, ukoliko izlazna lista treba sadržavati sve međusobno različite poštanske brojeve rođenja iz relacije **stud**, koristit će se sljedeći upit:

```
SELECT DISTINCT pbrRod
FROM stud
```

Prethodna naredba je primjer obavljanja operacije projekcije nad relacijom, $\pi_{pbrRod}(stud)$

Kvalifikator *ALL* ima suprotno značenje od *DISTINCT*. Može se ispustiti jer, ukoliko se ne navede kvalifikator *DISTINCT*, kvalifikator *ALL* primjenjuje se po definiciji (*by default*).

Sljedeći primjer pokazuje istovremenu upotrebu kvalifikatora *FIRST* i *DISTINCT*. Ispisuje se prvih deset različitih imena studenata iz relacije *stud*.

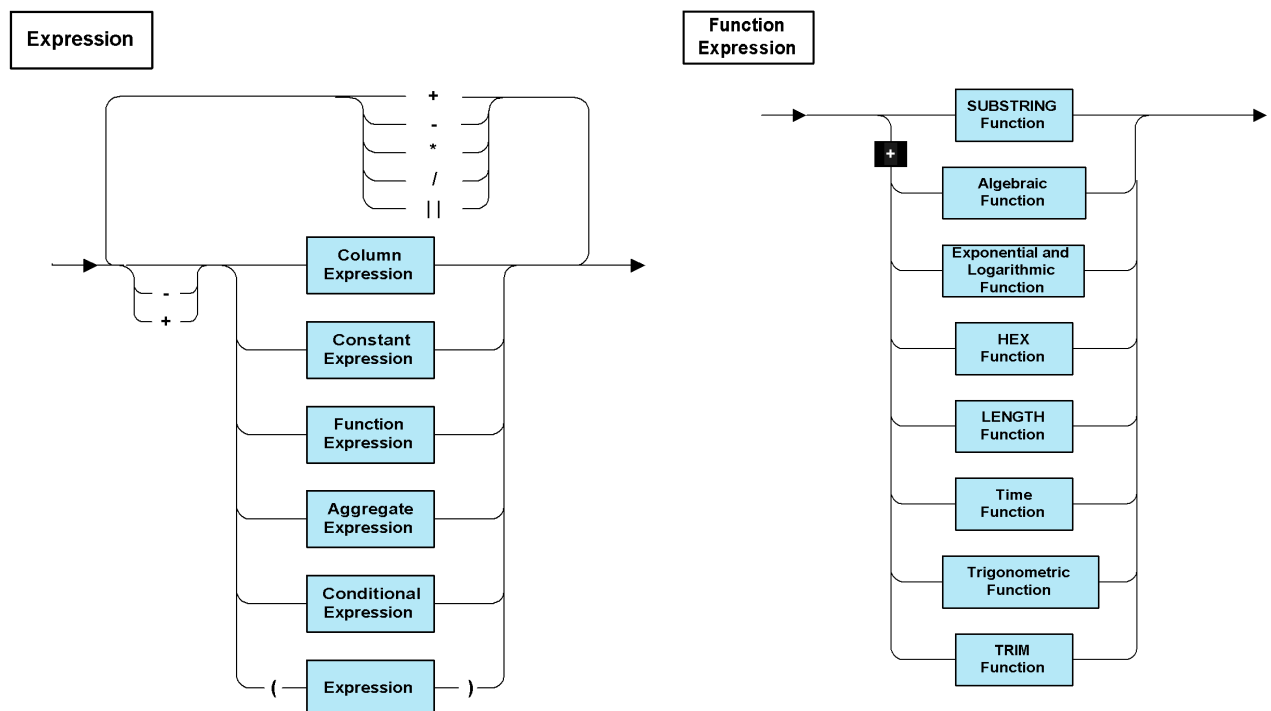
```
SELECT FIRST 10 DISTINCT imeStud
FROM stud
```

Najjednostavniji oblik *FROM Clause* sadrži ključnu riječ *FROM* i ime jedne relacije. U poglavlju u spajanju relacija bit će opisano zašto se i na koji način u *FROM* listi navodi više od jedne relacije.

4.2 Izraz (Expression)

4.2.1 Atributi, konstante, aritmetički i znakovni operatori, funkcije

Lista za selekciju se gradi iz izraza (*Expression*). Izrazi se mogu formirati iz naziva atributa, konstanti, aritmetičkih operatora i različitih funkcija. Redoslijed obavljanja operacija je uobičajen kao i u ostalim jezicima, a ukoliko je taj redoslijed neprikladan, izrazi se mogu grupirati pomoću zagrada. U nastavku su prikazani i ukratko objašnjeni sintaksni dijagrami pomoću kojih se grade izrazi.



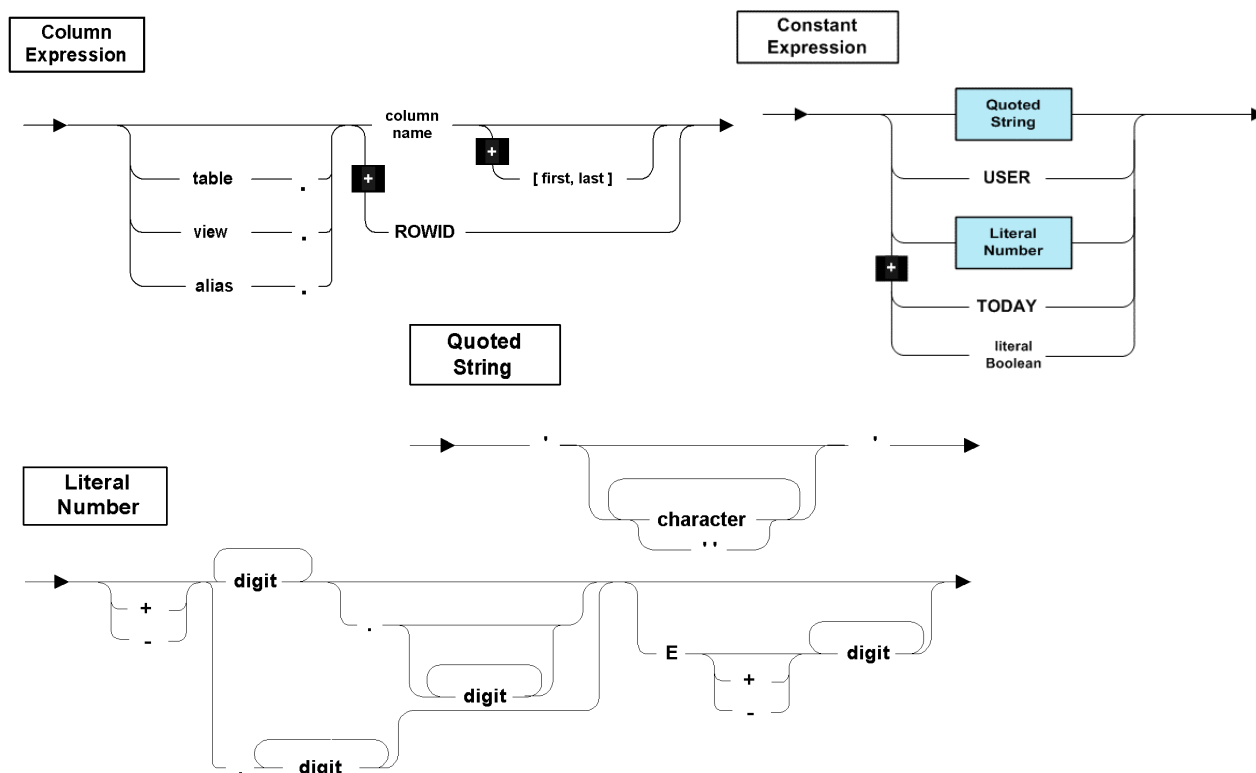
Binarni operatori za zbrajanje, oduzimanje, množenje i dijeljenje, te unarni operatori *+* i *-* odgovaraju istim takvim operatorima u drugim programskim jezicima. Dvije okomite crte (*pipe sign*), bez razmaka *||*, predstavljaju operator za konkatenciju nizova znakova.

Column Expression već korišten u prijašnjim primjerima. Dijagram prikazuje da se na atribut tipa niza znakova može primijeniti operator za podniz.



columnName[*m*, *n*] - znači: podniz vrijednosti atributa od pozicije **m** do **n**. Umjesto ovog nestandardnog načina određivanja podniza, bolje je koristiti standardnu SQL funkciju SUBSTRING, koja je objašnjena kasnije

Constant Expression može biti niz znakova (*Quoted String*), numerička konstanta (*Literal Number*) ili "pseudovarijable" TODAY odnosno USER. TODAY sadrži trenutnu vrijednost datuma dobivenu od operacijskog sustava, a USER je ime korisnika (*login name*).



Znakovna konstanta (*quoted string*) se omeđuje jednostrukim navodnicima. Ukoliko je jednostruki navodnik dio niza, na mjestu svakog takvog navodnika se upišu dva jednostruka navodnika.

Primjer: Znakovna konstanta koja sadrži prezime O' Shaughnessy u SQL-u se opisuje na sljedeći način:

'O' 'Shaughnessy'

Numeričke konstante (*literal number*) su slične kao i u ostalom programskim jezicima. Npr.

11
-123
+121
6.022E23

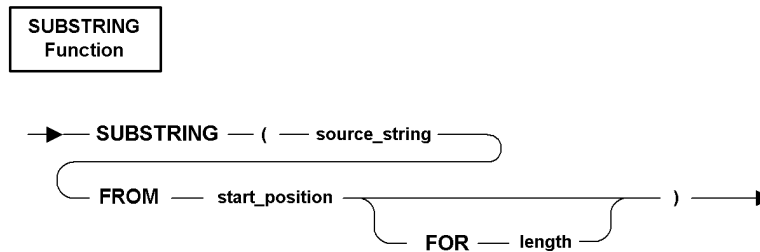
Logičke konstante (*literal Boolean*) su 't' odnosno 'T' i 'f' odnosno 'F'.

Aggregate Expression izrazi će biti razmatrani u posebnom poglavlju.

Function Expression datumske, algebarske, eksponencijalne, logaritamske, trigonometrijske funkcije, te funkcije HEX, TRIM, LENGTH i SUBSTRING.

- Datumske funkcije su prikazane u posebnom poglavlju.
- Algebarske funkcije su **ABS**, **MOD**, **POW**, **ROOT**, **ROUND**, **SQRT**, **TRUNC**

- Eksponencijalne i logaritamske funkcije su EXP, LOGN, LOG10
- Trigonometrijske funkcije su COS, SIN, TAN, ASIN, ACOS, ATAN, ATAN2
- Funkcija HEX za zadani cjelobrojni izraz vraća njegovu heksadecimalnu notaciju
- Funkcija **TRIM**(*nizZnakova*) vraća niz znakova iz kojeg su izbačene sve praznine ('blank' znakovi) s početka i kraja niza
- Funkcija **LENGTH**(*nizZnakova*) vraća broj znakova u nizu znakova ne uzimajući u obzir 'blank' znakove s kraja niza
- Funkcija **SUBSTRING** se koristi za određivanje podniza znakova. Preporuča se korištenje te funkcije, umjesto nestandardnog načina određivanja podniza pomoću izraza s uglatim zagradama oblika *column[start_position, end_position]*. Ukoliko se opcionalni dio izraza *FOR length* ne navede, rezultatni podniz sadrži sve znakove od pozicije *start_position* do kraja niza *source_string*.



Primjer: Ispis imena i prezimena studenta u obliku jednog stupca

```

SELECT imeStud || ' ' || prezStud
FROM stud

Ivan Horvat
Jasminko Kolar
  
```

Pomoću TRIM funkcije izbacuju se iz rezultata praznine (*blank*) koje nastaju jer se vrijednost atributa tipa CHAR nadopunjava prazninama do duljine atributa koja je deklarirana prilikom kreiranja relacije.

```

SELECT TRIM(imeStud) || ' ' || prezStud
FROM stud

Ivan Horvat
Jasminko Novak
  
```

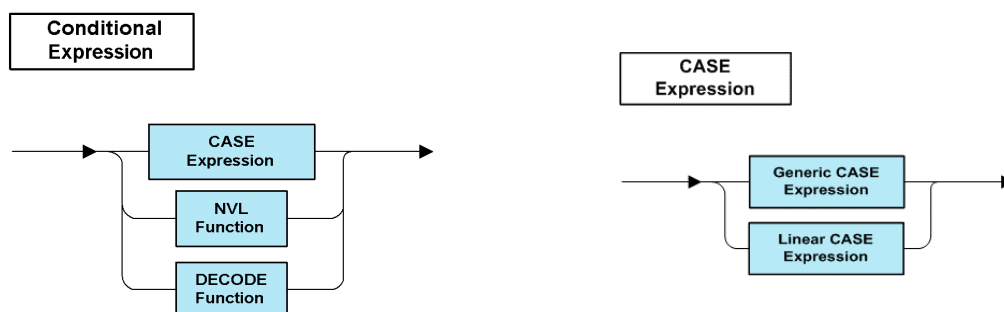
Primjer: Ispis podniza od trećeg do sedmog znaka imena studenta

```

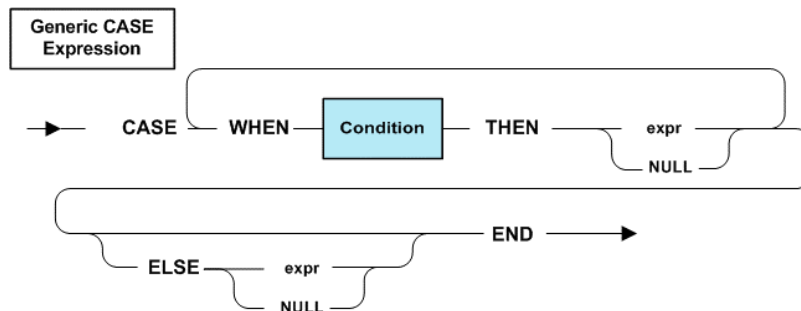
SELECT SUBSTRING(imeStud FROM 3 FOR 5)
FROM stud

an
smink
  
```

Conditional Expression omogućava transformaciju vrijednosti na temelju zadanih uvjeta. Postoje tri oblika: *CASE Expression*, *NVL Function*, *Decode Function*.



CASE Expression je izraz sličan naredbi CASE u drugim programskim jezicima. Izrazom se odabire jedna od navedenih vrijednosti na temelju evaluiranja uvjeta. *CASE Expression* ima dva oblika: *Generic CASE Expression* i *Linear CASE Expression*.



Generic CASE Expression rezultira vrijednošću koja je navedena uz jedan od navedenih uvjeta (*Condition*). Odabire se vrijednost koja se nalazi uz prvi uvjet (*Condition*) koji je zadovoljen. Vrijednost navedena uz svaki od uvjeta može biti *Expression* ili NULL vrijednost. Ukoliko niti jedan od navedenih uvjeta nije zadovoljen, rezultat je vrijednost navedena u ELSE dijelu izraza. Ukoliko niti jedan od navedenih uvjeta nije zadovoljen, a ELSE dio nije naveden, rezultat je NULL.

Primjer: Zadana je relacija **student**.

mbrStud	prosjeak
100	4.23
101	2.34
102	3.10
103	4.75

```
SELECT mbrStud,
       CASE WHEN prosjeak >= 4.50 AND prosjeak <= 5.00 THEN 'izvrstan'
            WHEN prosjeak >= 3.50 AND prosjeak < 4.50 THEN 'vrlodobar'
            ELSE 'ostalo'
       END
FROM student
```



rezultat:

mbrStud	(expression)
100	vrlodobar
101	ostalo
102	ostalo
103	izvrstan

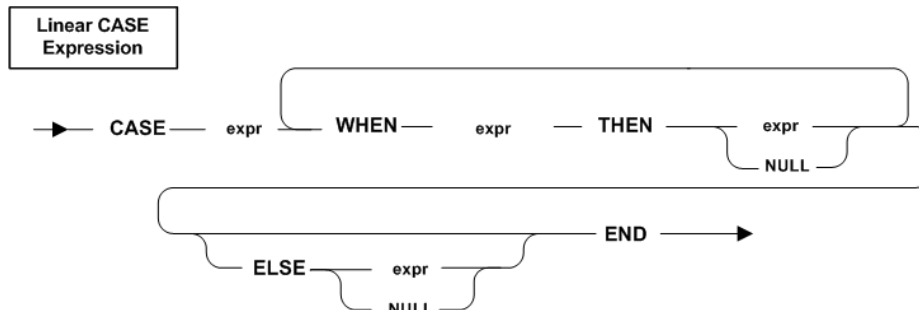
```
SELECT mbrStud,
       CASE WHEN prosjeak >= 4.50 AND prosjeak <= 5.00 THEN 'izvrstan'
            WHEN prosjeak >= 3.50 AND prosjeak < 4.50 THEN 'vrlodobar'
            -- ELSE nije naveden
       END
FROM student
```



rezultat:

mbrStud	(expression)
100	vrlodobar
101	NULL
102	NULL
103	izvrstan

Linear CASE Expression rezultira vrijednošću koja je navedena uz izraz koji je **jednak** testiranom izrazu. Testirani izraz je naveden iza ključne riječi CASE. Ukoliko niti jedna od navedenih vrijednosti nije jednaka testiranoj vrijednosti, rezultat je vrijednost navedena u ELSE dijelu izraza. Ukoliko niti jedna od navedenih vrijednosti nije jednaka testiranoj vrijednosti, a ELSE nije naveden, rezultat je NULL.



Primjer: Zadana je relacija **ispit**.

ispit	
mbrStud	ocjena
100	4
101	2
102	3
103	5

```

SELECT mbrStud,
       CASE ocjena
         WHEN 5 THEN 'izvrstan'
         WHEN 4 THEN 'vrlodobar'
         ELSE 'ostalo'
       END
FROM ispit

```



rezultat:	
mbrStud	(expression)
100	vrlodobar
101	ostalo
102	ostalo
103	izvrstan

```

SELECT mbrStud,
       CASE ocjena
         WHEN 5 THEN 'izvrstan'
         WHEN 4 THEN 'vrlodobar'
         -- ELSE nije naveden
       END
FROM ispit

```



rezultat:	
mbrStud	(expression)
100	vrlodobar
101	NULL
102	NULL
103	izvrstan

NVL Function vraća vrijednost prvog ili drugog argumenta, zavisno od toga je li prvi argument NULL vrijednost. Ukoliko prvi argument nije NULL, vraća vrijednost prvog argumenta. Ukoliko je prvi argument NULL, vraća vrijednost drugog argumenta.

NVL Function

→ **NVL** (— expression_1 — , — expression_2 —) →

Primjer: Zadana je relacija **ispit**.

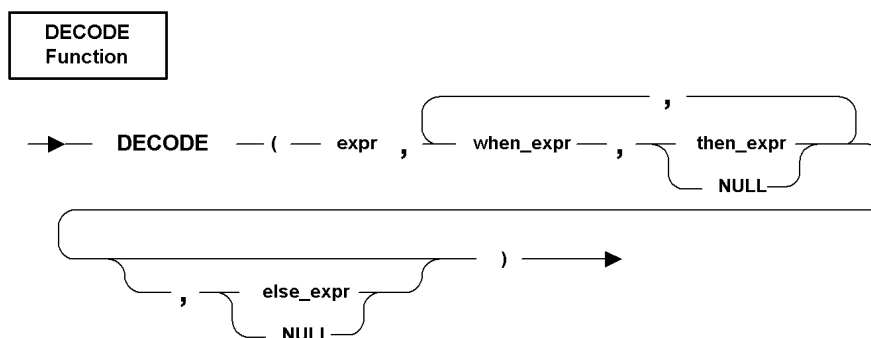
<u>mbrStud</u>	<u>ocjena</u>
100	4
101	2
102	NULL
103	5

```
SELECT mbrStud,
       NVL (ocjena * 100, -1)
FROM ispit
```



rezultat:	
<u>mbrStud</u>	<u>(expression)</u>
100	400
101	200
102	-1
103	500

DECODE Function je vrlo sličan izrazu *Linear CASE Expression*. Prvi argument odgovara testiranom izrazu u *Linear CASE Expression*. Slijede "parovi argumenata". Ukoliko je testirani izraz jednak prvom članu jednog od parova, tada se kao rezultat izraza dobiva drugi član tog para. Ukoliko prvi član niti jednog para nije jednak testiranom izrazu, kao rezultat se dobiva posljednji argument (*else_expr*). Ukoliko prvi element niti jednog para nije jednak testiranom izrazu, a taj posljednji argument (*else_expr*) nije naveden, rezultat je NULL.



Primjer: Zadana je relacija **ispit**.

ispit	
<u>mbrStud</u>	<u>ocjena</u>
100	4
101	2
102	3
103	5

```
SELECT mbrStud,
       DECODE (ocjena
              , 5, 'izvrstan'
              , 4, 'vrlodobar'
              , 'ostalo')
FROM ispit
```



rezultat:	
<u>mbrStud</u>	<u>(expression)</u>
100	vrlodobar
101	ostalo
102	ostalo
103	izvrstan

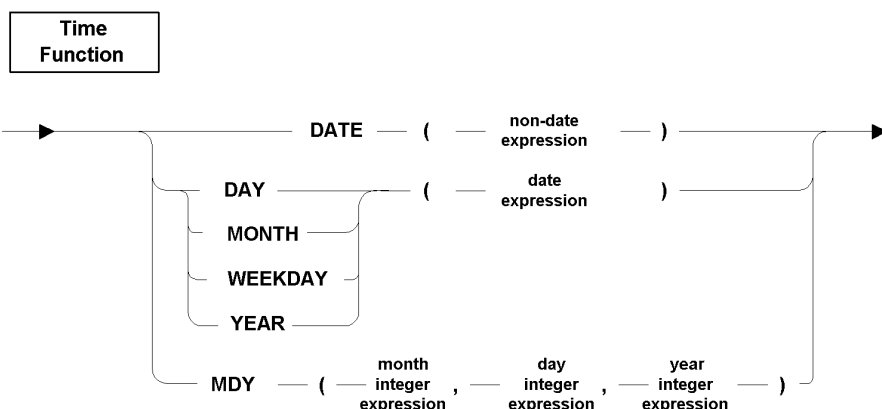

```
SELECT mbrStud,
       DECODE (ocjena
              , 5, 'izvrstan'
              , 4, 'vrlodobar')
FROM ispit
```



rezultat:

mbrStud	(expression)
100	vrlodobar
101	NULL
102	NULL
103	izvrstan

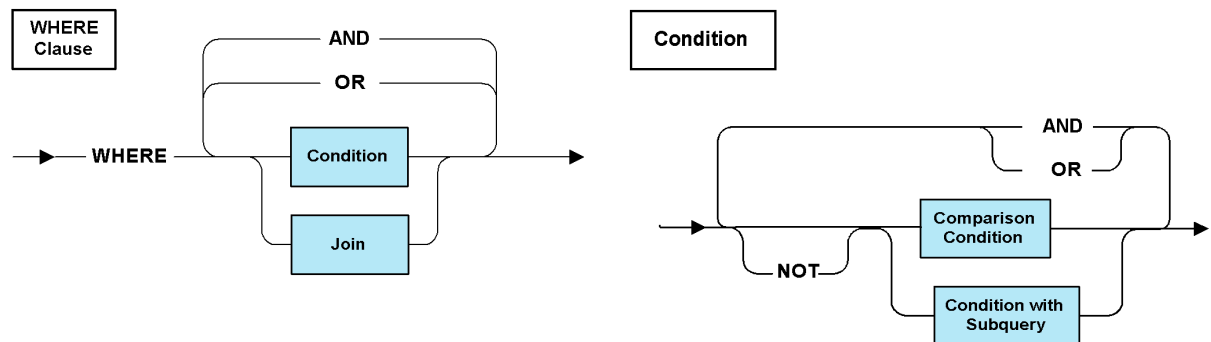
4.2.2 Datumske funkcije u SQL-u



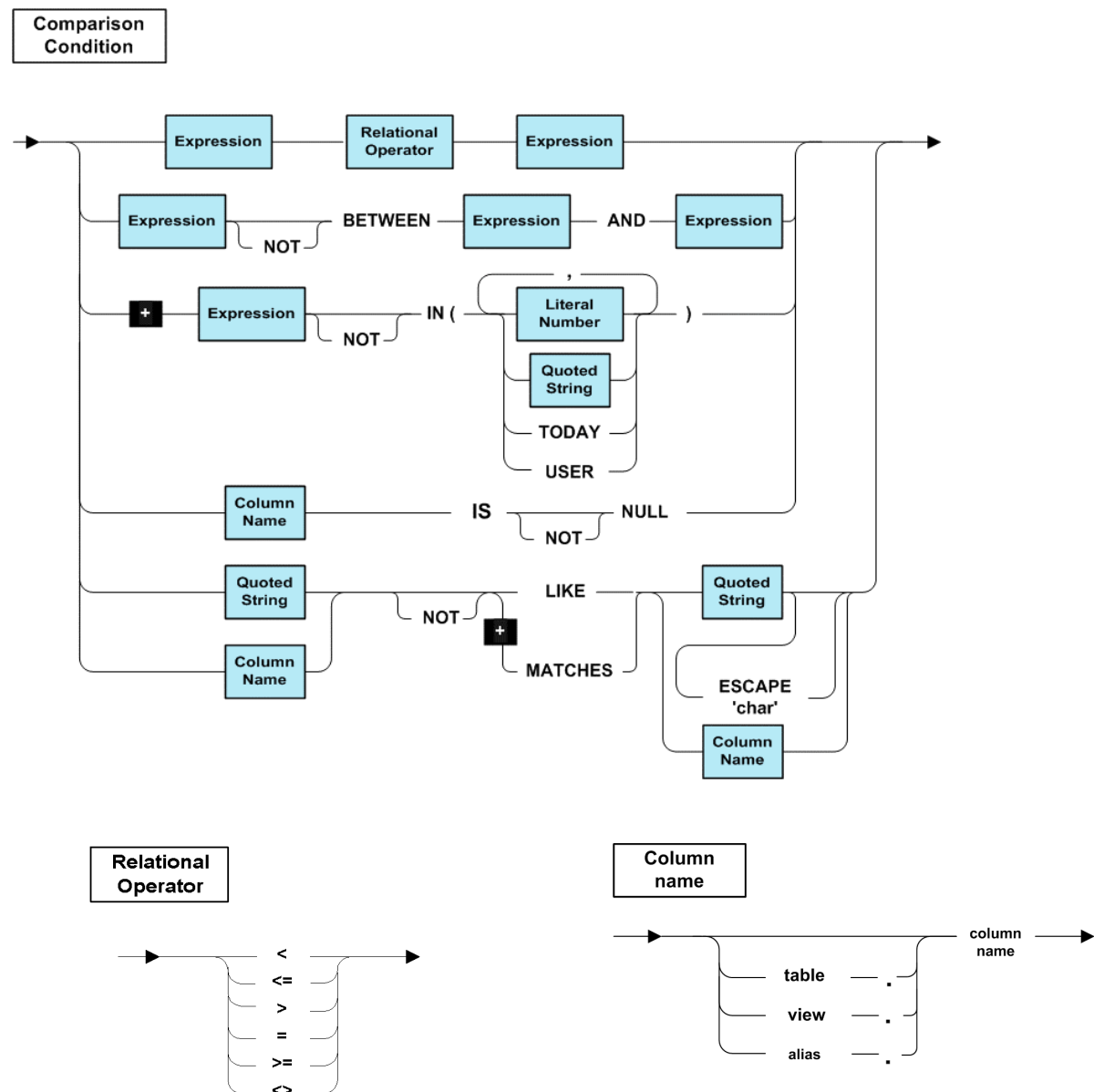
- DATE** - evaluira datum iz INTEGER ili CHAR tipa podatka
 DATE (27245) evaluira se kao datumska vrijednost 05.08.1974
 DATE (0) evaluira se kao datumska vrijednost 31.12.1899
- DAY** - redni broj dana u mjesecu za zadani datum
 DAY ('18.11.1987') je cijeli broj 18
- MDY** - evaluira datum iz tri INTEGER vrijednosti (mjesec, dan, godina)
 MDY (5, 14, 1996) je datumska vrijednost 14.5.1996
- MONTH** - za zadani datum vraća redni broj mjeseca
 MONTH ('18.11.1987') je cijeli broj 11
- WEEKDAY** - redni broj dana u tjednu za zadani datum (0-nedjelja, 1-ponedjeljak itd.)
 WEEKDAY ('14.04.1992') je cijeli broj 2 (jer je 14.04.1992 bio utorak)
- YEAR** - redni broj godine za zadani datum
 YEAR ('18.11.1987') je cijeli broj 1987

4.3 Definiranje uvjeta dohvata

4.3.1 Uvjeti za dohvata i uvjeti usporedbe



Dijelove dijagrama u kojima se spominje *Condition with Subquery* ili *Join* za sada treba zanemariti. Radi se o uvjetima spajanja i podupitima koji će biti opisani u posebnim poglavljima.



Uvjet za dohvat (*Condition*) je izreka izgrađena od jednog ili više uvjeta usporedbe (*Comparison Condition*) koji se međusobno povezuju logičkim operatorima AND, OR i NOT. Prioritet logičkih operatora je jednak kao u drugim programskim jezicima: NOT, AND, OR, ali se uvjeti usporedbe mogu grupirati uz pomoć okruglih zagrada čime se utječe na redoslijed evaluacije.

U dijagramu za uvjet usporedbe (*Comparison Condition*) može se prepoznati šest različitih oblika uvjeta:

1. Usporedba izraza (*Expression*) s drugim izrazom pomoću relacijskog operatora
2. Ispitivanje je li izraz unutar (ili izvan) intervala omeđenog s druga dva izraza
3. Ispitivanje je li izraz element nekog skupa (ili nije element skupa) koji može sadržavati numeričke, znakovne konstante, te "varijable" TODAY i USER. TODAY je datumska "varijabla" koja sadrži današnji datum (dobiven od operacijskog sustava), a USER je znakovna "varijabla" koja sadrži naziv (*login*) korisnika, također dobiven od operacijskog sustava
4. Ispitivanje je li vrijednost atributa NULL (ili vrijednost atributa nije NULL)
5. Ispitivanje zadovoljava li (ili ne zadovoljava) vrijednost atributa ili znakovna konstanta zadani uzorak (*pattern*) uz korištenje LIKE operatora
6. Ispitivanje zadovoljava li (ili ne zadovoljava) vrijednost atributa ili znakovna konstanta zadani uzorak (*pattern*) uz korištenje MATCHES operatora

Primjeri:

1. Nastavnici (svi podaci o nastavnicima) čiji je koeficijent za plaću veći ili jednak od 5.50. Ovdje prvi izraz predstavlja vrijednost atributa **koef** u n-torki relacije **nastavnik**, a drugi izraz je numerička konstanta. Ove se dvije vrijednosti uspoređuju pomoću relacijskog operatora \geq , te se ispisuju one n-torke za koje se rezultat usporedbe evaluira kao istinit.

```
SELECT *
FROM nastavnik
WHERE koef >= 5.50
```

Naredba je primjer za operaciju selekcije $\sigma_{\text{koef} \geq 5.50}$ (nastavnik)

- 2a. Studenti (podaci o matičnom broju, prezimenu i datumu rođenja studenta) čiji je datum rođenja između '1.1.1981' i '5.4.1981' (uporaba operatora *BETWEEN*). Ovdje prvi izraz predstavlja vrijednost atributa **datRodStud** u n-torki relacije **stud**, a druga dva izraza su datumske konstante koje definiraju zatvoreni interval.

```
SELECT mbrStud, prezStud, datRodStud
FROM stud
WHERE datRodStud BETWEEN '1.1.1981' AND '5.4.1981'
```

- 2b. Studenti čiji datum rođenja NIJE između '1.1.1981' i '5.4.1981'.

```
SELECT mbrStud, prezStud, datRodStud
FROM stud
WHERE datRodStud NOT BETWEEN '1.1.1981' AND '5.4.1981'
```

- 3a. Popis matičnih brojeva studenata koji su polagali ispit iz predmeta koji ima šifru unutar zadanog skupa šifara predmeta. Ujedno je prikazana uporaba kvalifikatora DISTINCT. DISTINCT kvalifikator osigurava da se matični brojevi studenata koji su polagali više nego jedan ispit s popisa ili su više puta polagali neki predmet s popisa, ne pojavljuju više od jednom u rezultatu upita.

```
SELECT DISTINCT mbrStud
FROM ispit
WHERE sifPred IN (346, 347, 348)
```

- 3b. Studenti čiji datum rođenja ne poprima niti jednu od vrijednosti '23.7.1981', '5.11.1980' i '31.1.1982' (uporaba operatora *NOT IN*)

```
SELECT *
FROM stud
WHERE datRodStud NOT IN ('23.7.1981', '5.11.1980', '31.1.1982')
```

- 4a. Studenti čiji je datum rođenja nepoznat ili nije upisan (uporaba operatora IS NULL)

```
SELECT *
FROM stud
WHERE datRodStud IS NULL
```

- 4b. Studenti čiji je datum rođenja upisan (uporaba operatora IS NOT NULL)

```
SELECT *
FROM stud
WHERE datRodStud IS NOT NULL
```

- 5a. Ispis svih podataka o studentima čije ime sadrži kombinaciju znakova 'ar' (uporaba operatora *LIKE*).

```
SELECT stud.*
FROM stud
WHERE imeStud LIKE '%ar%'
```

Pri korištenju operatora *LIKE* mogu se koristiti sljedeći **wildcard** znakovi:

znak % zamjenjuje bilo koju kombinaciju znakova

znak _ zamjenjuje jedan znak

znak \ ukida specijalno značenje znaka ispred kojeg se pojavi. Koristi se za ukidanje specijalnog značenja za znakove _ % \

- 5b. Ispis svih podataka o predmetima u čijem se nazivu, na bilo kojoj poziciji, pojavljuje znak %.

```
SELECT pred.*
FROM pred
WHERE nazPred LIKE '%\%%'
```

- 6a. Ispis svih podataka o studentima čije ime sadrži kombinaciju znakova 'ar' (uporaba operatora *MATCHES*)

```
SELECT *
FROM stud
WHERE imeStud MATCHES '*ar*'
```

MATCHES operator omogućava korištenje sljedećih **wildcard** znakova i kombinacija sa slovima

- | | | |
|------------|---|--|
| * | - | zamjenjuje bilo koju kombinaciju znakova (kao % kod operatora LIKE) |
| [znakovi] | - | zamjenjuje jedan od znakova koji se nalaze unutar zagrada |
| [^znakovi] | - | zamjenjuje jedan od znakova koji se NE nalaze unutar zagrada |
| [z1-z2] | - | zamjenjuje jedan znak iz intervala z1-z2 |
| ? | - | zamjenjuje jedan znak (kao "_" kod operatora LIKE) |
| \ | - | Escape znak - ukida specijalno značenje znaka ispred kojeg se pojavi. Koristi se za ukidanje specijalnog značenja za znakove \ ? * [] |

- 6b. Dohvat podataka za sve studente čije ime sadrži jednu od kombinacija znakova - **AR, Ar, aR, ar**

```
SELECT *
FROM stud
WHERE imeStud MATCHES '*[Aa][Rr]*'
```

- 6c. Ispis svih studenata čije prezime počinje nekim od znakova koji se nalaze između slova A i D, te završava jednim od znakova a, b, c, d, e.

```
SELECT *
FROM stud
WHERE prezStud MATCHES '[A-D]*[abcde]'
```

- 6d. Ispis svih studenata čije prezime počinje nekim od znakova koji se nalaze između slova E i G, i nakon tog prvog znaka sadrži barem dva znaka koji nisu samoglasnici.

```
SELECT *
FROM stud
WHERE prezStud MATCHES '[E-G]*[^aeiouAEIOU]*[^aeiouAEIOU]*'
```

6e. Ispis svih predmeta čiji naziv završava sa znakom **a** ili **e**, a sadrži znak *****.

```
SELECT *
FROM pred
WHERE nazPred MATCHES '*\**[ae]'
```

7. Uvjeti za usporedbu se mogu međusobno kombinirati pomoću logičkih operatora AND, OR i NOT. Ispis svih studenata koji su rođeni u mjestu s poštanskim brojem 20000 ili im prezime počinje znakom D, te uz taj uvjet, također zadovoljavaju uvjet da stanuju u mjestu s poštanskim brojem 22000.

```
SELECT *
FROM stud
WHERE ( pbrRod = 20000 OR prezStud MATCHES 'D*' ) AND pbrStan = 22000
```

Zagrade se ne smiju ispustiti jer bi redoslijed evaluacije uvjeta bez njih bio drugačiji nego sa zagradama!

Uvjet dohvata se može promatrati kao predikat kojeg n-torka iz relacije mora zadovoljiti da bi se pojavila u izlaznoj listi rezultata upita. Naime, uvrštavanjem konkretnih vrijednosti iz n-torke u predikat dobiva se sud, koji se evaluira kao istinit (*true*) ili lažan (*false*). Samo one n-torke za koje se taj sud evaluira kao **istinit**, pojavit će se u rezultatu SELECT naredbe.

Primjer: Neka relacija mjesto sadrži n-torke 10000, Zagreb; 20000, Dubrovnik; 31000, Osijek

```
SELECT *
FROM mjesto
WHERE pbr > 10000 AND nazMjesto MATCHES 'D*'
```

Uvrštavanje	1. n-torka	10000 > 10000 AND 'Zagreb' MATCHES 'D*'	→ false
vrijednosti	2. n-torka	20000 > 10000 AND 'Dubrovnik' MATCHES 'D*'	→ true
u predikat	3. n-torka	31000 > 10000 AND 'Osijek' MATCHES 'D*'	→ false

Rezultat upita će biti n-torka 20000, Dubrovnik.

4.3.2 Primjeri korištenja složenijih izraza (*Expression*) u uvjetima za usporedbu i listi za selekciju

Primjeri:

1. Ispis predmeta za koje vrijedi da im je umnožak broja upisanih studenata i tjednog opterećenja u satnici podijeljen sa 100 veći od 1.1

```
SELECT *
FROM pred
WHERE upisanoStud * brojSatiTjedno / 100 > 1.1
```

2. Ispis studenata za koje vrijedi da su im prva dva slova prezimena i prva dva slova imena jednaka.

```
SELECT *
FROM stud
WHERE prezStud[1,2] = imeStud[1,2]
```

Sljedeće rješenje je bolje jer poštuje SQL standard:

```
SELECT *
FROM stud
WHERE SUBSTRING(prezStud FROM 1 FOR 2) = SUBSTRING(imeStud FROM 1 FOR 2)
```

3. Ispis studenata čija je ukupna duljina imena i prezimena veća od 18.

```
SELECT *
FROM stud
WHERE LENGTH(prezStud) + LENGTH(imeStud) > 18
```

```
SELECT *
FROM stud
WHERE LENGTH( TRIM(prezStud) || imeStud ) > 18
```

4. Ispis studenata čija je starost izražena u danima veća od 20*365 (upit je neovisan o datumu izvođenja).

```
SELECT *
FROM stud
WHERE TODAY - datRodStud > 20*365
```

5. Dohvat zapisa svih studenata koji su rođeni 1981. godine.

```
SELECT *
FROM stud
WHERE YEAR(datRodStud) = 1981
```

6. Dohvat zapisa svih studenata koji su rođeni bilo kojeg ponedjeljka u 1981. godini.

```
SELECT *
FROM stud
WHERE YEAR(datRodStud) = 1981 AND WEEKDAY(datRodStud) = 1
```

7. Dohvat zapisa svih studenata koji su stari točno 19 godina, tj. danas im je rođendan (upit je neovisan o datumu izvođenja).

```
SELECT *
FROM stud
WHERE datRodStud = MDY (MONTH(TODAY), DAY(TODAY), YEAR(TODAY)-19)
```

***** ako se upit izvede 29. veljače, sustav će dojaviti pogrešku**

8. Dohvat zapisa svih studenata koji su rođeni na današnji dan prije 19 godina (upit je neovisan o datumu izvođenja).

```
SELECT *
FROM stud
WHERE YEAR(datRodStud) = YEAR(TODAY)-19
AND MONTH(datRodStud) = MONTH(TODAY)
AND DAY(datRodStud) = DAY(TODAY)
```

***** ako se upit izvede 29. veljače, neće se dobiti niti jedan zapis**

9. Dohvat zapisa svih studenata za koje ne postoji podatak o mjestu rođenja.

```
SELECT *
FROM stud
WHERE pbrRod IS NULL
```

10. Ispis matičnih brojeva i starosti svakog studenta izražene u danima (upit je neovisan o datumu izvođenja).

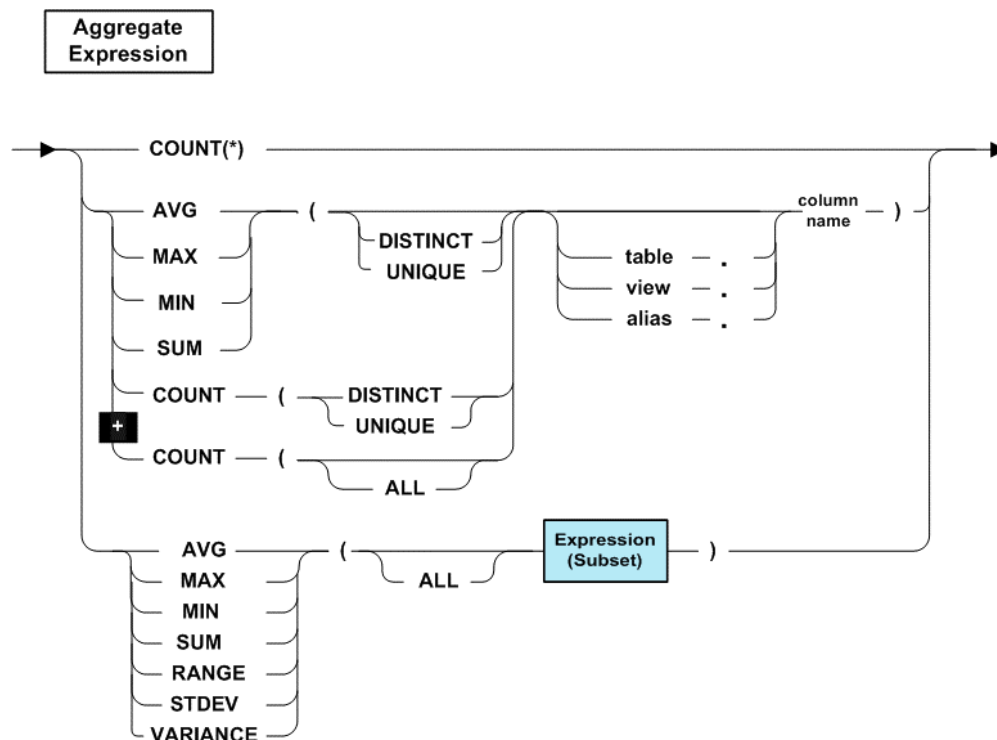
```
SELECT mbrStud, TODAY - datRodStud
FROM stud
```

11. Za svakog studenta ispisati sljedeće podatke: inicijal imena i prezime (razdvojeno točkom), poštanski broj mjesta rođenja, za one studente koji su rođeni u mjestu čiji je poštanski broj izražen u cijelim tisućicama (npr. 10000, 21000, 23000, ali ne i npr. 23100)

```
SELECT imeStud[1,1] || '.' || prezStud, pbrRod
FROM stud
WHERE MOD(pbrRod, 1000) = 0
```

4.4 Agregatne funkcije

Agregatne funkcije izračunavaju jednu vrijednost na temelju vrijednosti atributa iz jedne ili više n-torki.



Expression (Subset) u prethodnom dijagramu označava da se na tom mjestu može koristiti izraz, ali uz manja ograničenja - npr. ne može se koristiti izraz tipa agregatne funkcije. Npr. nije dopušteno napisati `AVG (COUNT ((*)))`.

Primjer: Agregatna funkcija koja izračunava prosječnu ocjenu dobivenu na svim ispitima za predmet sa šifrom 1034. U prosjek ulaze i pozitivne i negativne ocjene.

```
SELECT AVG(ocjena)
FROM ispit
WHERE sifPred = 429
```

Rezultat je jedna vrijednost: 2.57

Primjer: Prosječna ocjena i broj izlazaka na ispit, u istom upitu, za predmet sa šifrom 1034.

```
SELECT AVG(ocjena), COUNT(*)
FROM ispit
WHERE sifPred = 429
```

Rezultat je po jedna vrijednost za svaku agr.funkciju: 3.24, 7

Agregatne funkcije se koriste u dva oblika upita:

- jednostavniji način primjene agregatnih funkcija je kad se agregatne funkcije primjenjuje na sve selektirane n-torke. Na taj se način kao rezultat upita dobiva jedna n-torka (u toj n-torki ima onoliko vrijednosti koliko je agregatnih funkcija navedeno u listi za selekciju).
- složeniji slučaj, koji će biti razmatran u poglavlju o grupiranju, dopušta da se selektirane n-torke podijele u grupe prema nekom kriteriju, te da se agregatna vrijednost izračuna za svaku pojedinu grupu. Rezultat je više n-torki, po jedna za svaku dobivenu grupu.

Na raspolaganju su sljedeće agregatne funkcije

COUNT (*)	-	broj n-torki
COUNT (DISTINCT x)	-	broj n-torki s različitim vrijednostima od x
+ COUNT (x)	-	broj n-torki u kojima atribut x ima vrijednost različitu od NULL

SUM (x)	-	suma vrijednosti atributa x ili nekog izraza (<i>Expression</i>)
SUM (DISTINCT x)	-	suma različitih vrijednosti atributa x
AVG (x)	-	prosječna vrijednost atributa x ili nekog izraza (<i>Expression</i>)
AVG (DISTINCT x)	-	prosječna vrijednost različitih vrijednosti atributa x
MAX (x)	-	maksimalna vrijednost atributa x ili nekog izraza (<i>Expression</i>)
MIN (x)	-	minimalna vrijednost atributa x ili nekog izraza (<i>Expression</i>)
RANGE (x)	-	raspon vrijednosti atributa x ili nekog izraza (<i>Expression</i>)
STDEV (x)	-	standardna devijacija za vrijednost atributa x ili nekog izraza (<i>Expression</i>)
VARIANCE (x)	-	varianca za vrijednost atributa x ili nekog izraza (<i>Expression</i>)

Primjeri:

1. Ispisati podatak o ukupnom broju mjesta u B-zgradi (pretpostavlja se da dvorana iz B-zgrade ima oznaku koja započinje slovom B)

```
SELECT SUM(kapacitet)
FROM dvorana
WHERE oznDvorana MATCHES 'B*'
```

2. Koliko različitih kapaciteta dvorana postoji

```
SELECT COUNT(DISTINCT kapacitet)
FROM dvorana
```

3. Koliki je raspon između najvećeg i najmanjeg kapaciteta dvorane

```
SELECT RANGE(kapacitet)
FROM dvorana

SELECT MAX(kapacitet) - MIN(kapacitet)
FROM dvorana
```


4. Broj studenata koji imaju upisan datum rođenja (vrijednost datuma rođenja im nije NULL vrijednost)

```
SELECT COUNT(datRodStud)
FROM stud
```

Važna ograničenja kod korištenja agregatnih funkcija:


1. Ukoliko se ne koristi grupiranje, uz agregatne funkcije u listi za selekciju je dopušteno koristiti samo dodatne agregatne funkcije, a attribute ili izraze jedino ako su u ulozi argumenata tih agregatnih funkcija.

Primjer: Upit je sintaktički neispravan. Rezultat se ne može formirati jer bi se trebalo ispisati više različitih vrijednosti za **mbrStud** i samo jedna vrijednost za prosječnu ocjenu.

 `SELECT mbrStud, AVG(ocjena)`
FROM ispit

2. Agregatne je funkcije dopušteno koristiti u listi za selekciju, ali ne i u uvjetima dohvata. Usporedbe s agregatnim vrijednostima dopuštene su jedino u posebnom dijelu SELECT naredbe (→ vidjeti *HAVING Clause* u posebnom poglavlju).

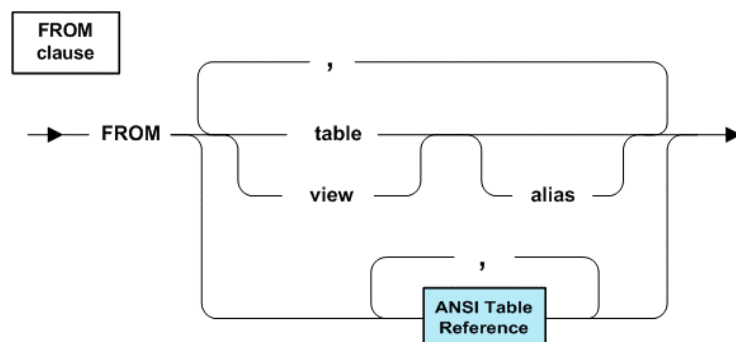
Primjer: Ovakav upit, kojim se pokušava odrediti dvorana koja ima broj mjesta veći od prosječnog broja mjesta, **NE VRIJEDI**, jer se agregatne funkcije ne smiju koristiti u WHERE dijelu naredbe.

 `SELECT * FROM dvorana`
WHERE kapacitet > AVG(kapacitet)

U poglavlju o uvjetima sa podupitima prikazano je ispravno rješenje ovog zadatka, ali uz uporabu podupita.

4.5 Spajanje relacija

U *FROM* dijelu svake do sada prikazane *SELECT* bila je navedena samo jedna relacija. Sljedeći dijagrami pokazuju da je u jednoj *SELECT* naredbi dopušteno navesti više od jedne relacije:



Podijagram *ANSI Table Reference* će biti objašnjen u poglavlju o vanjskom spajanju.

U dijelu SQL naredbe koji se naziva *FROM Clause* mogu se navesti relacije (jedna ili više njih) čiji se podaci koriste u upitu. U većini slučajeva se više nego jedna relacija navodi u slučajevima kada nad relacijama treba obaviti prirodno spajanje, spajanje uz uvjet ili Kartezijev produkt (obavljanje Kartezijevog produkta relacija ima rijetku primjenu u praksi).

4.5.1 Kartezijev produkt relacija

STUDENT		MJESTO		ISPIT		
<u>imeStud</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>	<u>imeStud</u>	<u>nazPredmet</u>	<u>ocjena</u>
Ivo	10000	10000	Zagreb	Ivo	Matematika	4
Zrinka	10000	51000	Rijeka	Ivo	Fizika	2
Marko	51000			Marko	Fizika	5
				Zrinka	Fizika	5

SELECT *		<u>imeStud</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
FROM student, mjesto		Ivo	10000	10000	Zagreb
		Ivo	10000	51000	Rijeka
		Zrinka	10000	10000	Zagreb
		Zrinka	10000	51000	Rijeka
		Marko	51000	10000	Zagreb
		Marko	51000	51000	Rijeka

Naredba je primjer za operaciju relacijske algebre Kartezijev produkt, **student × mjesto**.

Napomena: zato što je u listi za selekciju naveden znak *, prikazuju se svi atributi iz obje relacije. Znak * zamjenjuje sve attribute u relaciji ili relacijama navedenim u *FROM Clause*. Isti rezultat bi se dobio i sa *SELECT student.*, mjesto.**. Redoslijed atributa pri ispisu određen je redoslijedom atributa u relaciji ili relacijama.

Obavljanje Kartezijevog produkta relacija često je posljedica pogreške programera - kad se zaboravi navesti uvjet spajanja. U prethodnom primjeru kao rezultat upita dobije se 3*2 n-torki. U slučaju da su relacije imale npr. svaka po 10 000 zapisa, rezultat bi sadržavao 10⁸ n-torki !!! Moguće je zamisliti koliko će računalskih resursa SUBP utrošiti za obavljanje upita koji, pri tome, rezultira posve beskorisnim rezultatom.

4.5.2 Prirodno spajanje relacija

Za ispravno obavljanje prirodnog spajanja nužno je navesti **uvjet spajanja** (*Join*) koji se navodi u *WHERE Clause*. Bez tog uvjeta rezultat spajanja relacija će uvijek biti Kartezijev produkt relacija.

```
SELECT student.*, mjesto.nazMjesto
FROM student, mjesto
WHERE student.pbr = mjesto.pbr
```

<u>imeStud</u>	<u>pbr</u>	<u>nazMjesto</u>
Ivo	10000	Zagreb
Zrinka	10000	Zagreb
Marko	51000	Rijeka

Primijetite da je lista za selekciju promijenjena u odnosu na prethodni primjer! Oni atributi koji se pojavljuju u obje relacije se u listi za selekciju navode samo jednom, s ciljem da se dobije rezultat koji odgovara definiciji prirodnog spajanja.

Prethodna je naredba primjer za operaciju prirodno spajanje (**student ▷◁ mjesto**)

Konceptualno (ne i fizički, jer bi takav postupak bio neoptimalan), može se smatrati da se upiti sa spajanjem relacija obavljaju u tri faze:

1. Određuje se Kartezijev produkt relacija
2. Obavlja se selekcija onih n-torki koje zadovoljavaju uvjete spajanja
3. Ispisuju se vrijednosti atributa navedenih u listi za selekciju za sve n-torke iz 2. koraka, ili se obavlja projekcija po atributima navedenim u listi za selekciju (ukoliko je naveden DISTINCT kvalifikator).

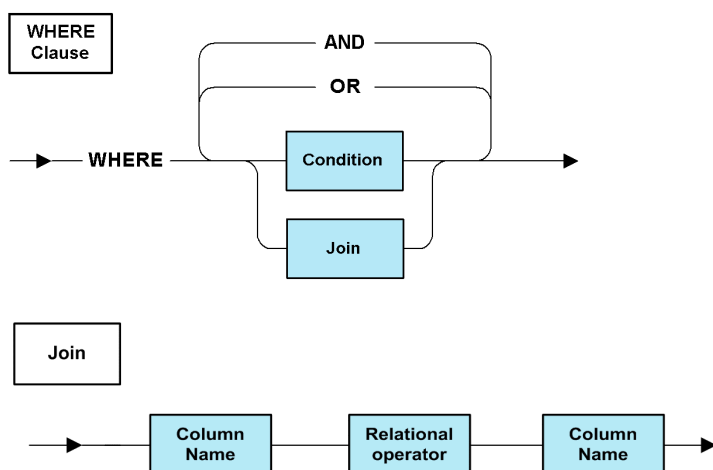
Sljedeći primjer ilustrira mogućnost spajanja više relacija, te dodavanje uvjeta selekcije uz uvjete spajanja.

Dohvat imena studenta, naziva predmeta i ocjena svih studenata čije mjesto stanovanja ima naziv koji počinje slovom **Z**, a koji su na ispitu dobili ocjenu veću od **3**.

```
SELECT student.imeStud, ispit.nazPredmet, ispit.ocjena
FROM student, mjesto, ocjena
WHERE student.pbr = mjesto.pbr
      AND student.imeStud = ispit.imeStud
      AND mjesto.nazMjesto MATCHES 'Z*'
      AND ispit.ocjena > 3
```

<u>imeStud</u>	<u>nazPredmet</u>	<u>ocjena</u>
Ivo	Matematika	4
Zrinka	Fizika	5

Sintaksni dijagram za *Join* opisuje način na koji se definira prirodno spajanje i spajanje uz uvjet.



4.5.3 Način spajanja relacija u slučaju postojanja "paralelne" veze

Relacija student sada sadrži attribute kojima se opisuju mjesto stanovanja i mjesto rođenja.

STUDENT			MJESTO	
<u>imeStud</u>	<u>pbrRod</u>	<u>pbrStan</u>	<u>pbr</u>	<u>nazMjesto</u>
Ivo	10000	10000	10000	Zagreb
Zrinka	10000	51000	51000	Rijeka
Marko	51000	10000		

Ispis studenata u kojem se pojavljuje naziv mjesta rođenja lako se rješava na način kakav je opisan u prethodnom primjeru:

```
SELECT imeStud, pbrRod, nazMjesto
FROM student, mjesto
WHERE pbrRod = pbr
```

- * **primjetite da u prethodnom upitu attribute nije trebalo kvalificirati imenima relacija, jer njihovi nazivi jednoznačno određuju o kojim se atributima iz kojih relacija radi.**

Upit postaje nešto složeniji kad se postavi zahtjev za ispisom liste studenata, ali tako da se uz svakog studenta istovremeno ispišu naziv njegovog (njezinog) mjesta rođenja i mjesta stanovanja.

NEISPRAVNO POSTAVLJEN UPIT:

```
SELECT student.*,
        mjesto.nazMjesto nazMjestoRod, mjesto.nazMjesto nazMjestoStan
FROM student, mjesto
WHERE pbrRod = mjesto.pbr
AND pbrStan = mjesto.pbr
```

Prethodni upit ne vrijedi jer bi se kao rezultat pojavile samo one n-torke iz relacije **student** u kojima su mjesto rođenja i mjesto stanovanja jednaki. Trebalo bi koristiti dvije relacije: **mjestoRodjenja** i **mjestoStanovanja**. Budući da posebne relacije **mjestoRodjenja** i **mjestoStanovanja** ne postoje, očito se relacija **mjesto** mora u upitu pojaviti dva puta, ali jednom u ulozi mjesta rođenja, a jednom u ulozi mjesta stanovanja.

Ukoliko ista relacija ima različite uloge (relacija **mjesto** ima ulogu mjesta rođenja i mjesta stanovanja), formira se upit u kojem se ta relacija u *FROM Clause* pojavljuje dva puta. Da bi se njezine uloge razlikovale, obavezno je korištenje alternativnih ili zamjenskih naziva relacija (*alias*). Alternativni nazivi relacija se specificiraju jednostavno navođenjem iza originalnog naziva relacije. Uvedeno alternativno ime tada se može koristiti u upitu kao da se radi o "pravoj" relaciji.

Napomena: postoji važna razlika između *alias* imena atributa/izraza iz liste za selekciju i *alias* imena relacije/pogleda. Dok se *alias* ime atributa/izraza **smije** u upitu pojaviti još jedino u ORDER BY dijelu naredbe, *alias* ime relacije/pogleda se **mora** koristiti umjesto "originalnog" imena relacije u cijelom upitu.

```
SELECT student.*, mjestoRod.*, mjestoStan.*
FROM student, mjesto mjestoRod, mjesto mjestoStan
WHERE student.pbrRod = mjestoRod.pbr
AND student.pbrStan = mjestoStan.pbr
```

<u>ime</u>	<u>pbrRod</u>	<u>pbrStan</u>	<u>pbr</u>	<u>nazMjesto</u>	<u>pbr</u>	<u>nazMjesto</u>
Ivo	10000	10000	10000	Zagreb	10000	Zagreb
Zrinka	10000	51000	10000	Zagreb	51000	Rijeka
Marko	51000	10000	51000	Rijeka	10000	Zagreb

4.5.4 Način spajanja relacija u slučaju postojanja "refleksivne" veze

Kada u relaciji postoji refleksivna veza, pojedine n-torke iz relacije povezane su s drugim n-torkama iz iste relacije. U konkretnom slučaju, svaka organizacijska jedinica iz relacije **orgjed** povezana je sa svojom nadređenom organizacijskom jedinicom pomoću atributa **sifNadorgjed**.

ORGJED

<u>sifOrgjed</u>	<u>nazOrgjed</u>	<u>sifNadorgjed</u>
1	Uprava	NULL
2	Odjel A	1
3	Odjel B	1
4	Pododjel X	2
5	Pododjel Y	2
6	Pododjel Z	3

U slučajevima kada je potrebno uspoređivati pojedine organizacijske jedinice s njihovim nadređenim ili podređenim organizacijskim jedinicama, potrebno je obaviti spajanje relacije "same sa sobom". Problem je sličan i slično se rješava kao u paralelnoj vezi. Relacija **orgjed** pojavljuje se u upitu dva puta, jednom u ulozi organizacijske jedinice, a jednom u ulozi njezine nadređene (ili podređene) organizacijske jedinice. Zbog toga je nužno korištenje zamjenskih (*alias*) naziva relacije.

Upit kojim se dohvaćaju organizacijske jedinice i nazivi njihovih nadređenih organizacijskih jedinica izgleda ovako:

```
SELECT orgjed.sifOrgjed
      , orgjed.nazOrgjed
      , orgjed.sifNadorgjed
      , nadorgjed.nazOrgjed nadredjenaNaziv
FROM orgjed, orgjed nadOrgjed
WHERE orgjed.sifNadorgjed = nadOrgjed.sifOrgjed
```

Nakon što se obavi Kartezijev produkt (u konkretnom slučaju rezultat je 36 n-torki), te se eliminiraju n-torke koje ne zadovoljavaju uvjet spajanja, dobije se sljedeći rezultat:

<u>sifOrgjed</u>	<u>nazOrgjed</u>	<u>sifNadorgjed</u>	<u>nadredjenaNaziv</u>
2	Odjel A	1	Uprava
3	Odjel B	1	Uprava
4	Pododjel X	2	Odjel A
5	Pododjel Y	2	Odjel A
6	Pododjel Z	3	Odjel B

*** Primjetite da se organizacijska jedinica "Uprava" nije pojavila u rezultatu, jer je za nju vrijednost sifNadorgjed NULL, te ne postoji n-torka iz nadOrgjed čija je šifra "jednaka" NULL vrijednosti.**

Upit kojim se dohvaćaju organizacijske jedinice i nazivi njihovih podređenih organizacijskih jedinica izgleda ovako:

```
SELECT orgjed.sifOrgjed
      , orgjed.nazOrgjed
      , podorgjed.sifOrgjed sifPodorgjed
      , podorgjed.nazOrgjed podredjenaNaziv
FROM orgjed, orgjed podOrgjed
WHERE podorgjed.sifNadorgjed = orgjed.sifOrgjed
```

*** Primjetite da se organizacijske jedinice koja nemaju podređene organizacijske jedinice neće pojaviti u rezultatu.**

<u>sifOrgjed</u>	<u>nazOrgjed</u>	<u>sifPodorgjed</u>	<u>podredjenaNaziv</u>
1	Uprava	2	Odjel A
1	Uprava	3	Odjel B
2	Odjel A	4	Pododjel X
2	Odjel A	5	Pododjel Y
3	Odjel B	6	Pododjel Z

4.5.5 Spajanje relacija uz uvjet

Slično prirodnom spajanju, obavlja se i spajanje relacija uz uvjet. U sljedećem primjeru ispisuju se zrakoplovi i mogući letovi (s obzirom na dolet i udaljenost koju je potrebno preletjeti)

ZRAKOPLOV		LET	
<u>tip</u>	<u>dolet</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	CA-825	7200
DC9	3000	A-224	3000
AIRBUS	1800	CA-878	4200

```
select * from zrakoplov, let
where zrakoplov.dolet >= udaljenost
```

<u>tip</u>	<u>dolet</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	A-224	3000
B747	6000	CA-878	4200
DC9	3000	A-224	3000

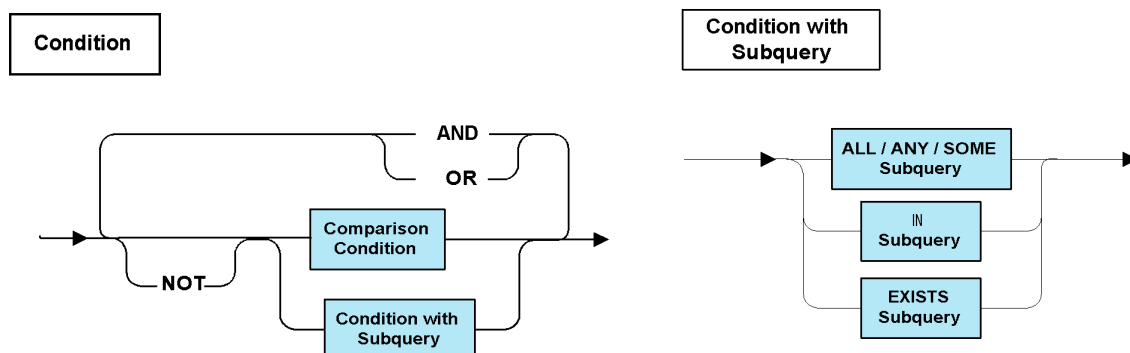
* **Primjetite da se u rezultatu ne pojavljuje zrakoplov AIRBUS, niti let CA-825**

Naredba je primjer operacije spajanja uz uvjet, **zrakoplov ▷◁ let**
dolet ≥ udaljenost

4.6 SQL naredbe koje sadrže uvjete s podupitom

Prema sintaksnom dijagramu za uvjet dohvata (*Condition*) vidi se da se osim uvjeta usporedbe (*Comparison Condition*), uvjet dohvata može graditi i iz uvjeta s podupitima (*Condition With Subquery*).

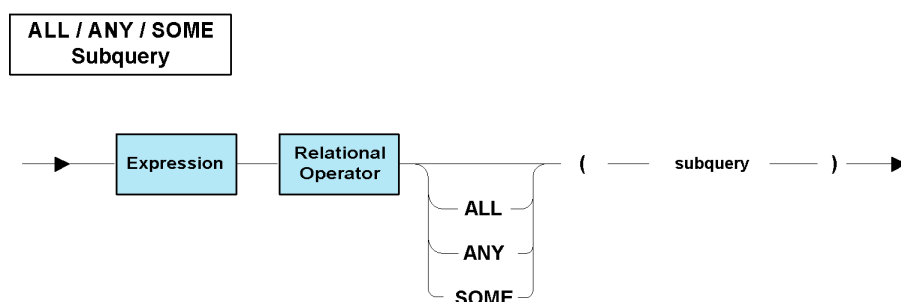
SELECT naredba ugniježđena unutar *WHERE Clause* neke druge SELECT, UPDATE ili DELETE naredbe naziva se podupit (*subquery*). Svaki podupit mora se nalaziti unutar okruglih zagrada. Upit u kojem je podupit neposredno sadržan naziva se vanjski upit (*outer statement*).



Podupit se može ugraditi u vanjski upit na tri osnovna načina (načini ugradnje podupita):

1. Usporedba izraza iz vanjskog upita s rezultatom podupita
2. Ispitivanje je li neki izraz iz vanjskog upita sadržan u rezultatu podupita
3. Ispitivanje da li se kao rezultat podupita pojavljuje barem jedna n-torka

1) Usporedba izraza iz vanjskog upita s rezultatom podupita



Relational Operator može biti bilo koji od relacijskih operatora (>, <, =, itd.). Rezultat SELECT naredbe koja predstavlja podupit (*subquery*), može biti jedna n-torka, više n-torki ili niti jedna n-torka. Da bi n-torka vanjskog upita zadovoljila uvjet selekcije napisan na ovaj način, mora biti zadovoljen uvjet usporedbe izraza *Expression* s jednom ili više n-torki koje se dobiju kao rezultat podupita.

Ukoliko se koristi ključna riječ **ALL**, **sve n-torke** dobivene podupitom moraju zadovoljiti uvjet usporedbe s *Expression*, da bi se izraz evaluirao kao istinit. Pri tome, ako podupit ne vrati niti jednu vrijednost, izraz se uvijek evaluira kao istinit (TRUE).

Ukoliko se upotrijebi ključna riječ **ANY** (ključna riječ **SOME** je sinonim), bit će dovoljno da uvjet usporedbe bude zadovoljen za **barem jednu n-torku** iz podupita. Pri tome, ako podupit ne vrati niti jednu vrijednost, izraz se uvijek evaluira kao lažan (FALSE).

Podupit upotrijebljen na ovaj način mora vraćati točno jedan stupac (dakle u listi za selekciju SELECT naredbe koja se koristi na mjestu *subquery*, smije se nalaziti samo jedan atribut ili izraz). U suprotnom, upit završava u pogrešci. Također, SELECT naredba koja se koristi za *subquery* ne smije sadržavati ORDER BY Clause.

Sljedećom naredbom određuju se predmeti na kojima je upisan tako malen broj studenata da se nastava može odjednom (bez podjele po dvoranama) održati u bilo kojoj dvorani:

```
SELECT *
FROM pred
WHERE upisanoStud <= ALL (SELECT kapacitet FROM dvorana)
```

Ovaj se upit (konceptualno promatrano) obavlja na sljedeći način: prvo se evaluira podupit kojim se formira skup čiji su elementi kapaciteti dvorana. Uzima se prva n-torka iz relacije **pred** pa se vrijednost atributa **upisaniStud** uspoređi sa svakom vrijednošću iz skupa kapaciteta. Ukoliko je vrijednost **upisaniStud** manja ili jednaka od svake vrijednosti s kojom se uspoređila, prva n-torka iz relacije **pred** će se pojaviti u rezultatu. Postupak se ponavlja za svaku n-torku relacije **pred**.

Sljedećom naredbom određuju se predmeti za koje postoji barem jedna dvorana s dovoljnim kapacitetom da se u njoj nastava održi odjednom za sve studente (bez dijeljenja studenata u više dvorana):

```
SELECT *
FROM pred
WHERE upisanoStud <= ANY (SELECT kapacitet FROM dvorana)
```

Sljedećom naredbom određuju se predmeti za koje se nastava ne može odjednom za sve upisane studente održati niti u jednoj dvorani:

```
SELECT *
FROM pred
WHERE upisanoStud > ALL (SELECT kapacitet FROM dvorana)
```

Ako podupit pouzdano vraća samo jednu n-torku (*single-valued subquery*) mogu se ključne riječi **ALL**, **ANY** i **SOME** ispustiti. U tom slučaju se vrijednost podupita može promatrati kao rezultat funkcije, a uvjet s podupitom kao usporedba s vrijednošću koja se dobije evaluiranjem funkcije.

Ukoliko bi se pri tako postavljenom podupitu dogodilo da podupit vrati više od jedne n-torke, upit bi završio u pogrešci. Ukoliko podupit ne vrati niti jednu n-torku, uvjet usporedbe se evaluira kao laž (FALSE).

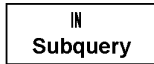
Sljedećom naredbom ispisuju se sve dvorane u kojima je broj mjesta veći od prosječnog broja mjesta. Primjetite da se u dvjema različitim ulogama koristi ista relacija:

```
SELECT *
FROM dvorana
WHERE kapacitet > (SELECT AVG(kapacitet) FROM dvorana)
```

Ispis dvorane (ili dvorana, jer ih može biti nekoliko s jednakim brojem mjesta) koje imaju najveći broj mjesta:

```
SELECT *
FROM dvorana
WHERE dvorana.kapacitet = (SELECT MAX(kapacitet) FROM dvorana)
```

2) Ispitivanje da li je neki izraz iz vanjskog upita sadržan u rezultatu podupita



Da bi n-torka vanjskog upita zadovoljila uvjet (tj. pojavila se u rezultatu):

ukoliko se NE koristi modifikator NOT - *Expression mora* biti sadržan u skupu dobivenom obavljanjem podupita

ukoliko se koristi modifikator NOT - *Expression ne smije* biti sadržan u skupu dobivenom obavljanjem podupita

Podupit upotrijebljen na ovaj način mora vraćati točno jedan stupac (dakle u listi za selekciju SELECT naredbe koja se koristi na mjestu *subquery*, smije se nalaziti samo jedan atribut ili izraz). U suprotnom, upit završava u pogrešci. Također, SELECT naredba koja se koristi za *subquery* ne smije imati *ORDER BY Clause*.

Primjer: ispis svih podataka o dvoranama u kojima se ne održava nastava

```
SELECT *
FROM dvorana
WHERE oznDvorana NOT IN (SELECT DISTINCT oznDvorana FROM rezervacija)
```

Uporaba ključne riječi DISTINCT neće utjecati na rezultat, ali se eliminacijom uspoređivanja s duplikatima često može ubrzati obavljanje upita.

3) Ispitivanje da li se kao rezultat podupita pojavljuje barem jedna n-torka



Rezultat podupita je jedna, više ili niti jedna n-torka. **Broj atributa u listi za selekciju ovog podupita nije ograničen niti je bitan za rezultat.**

Da bi n-torka vanjskog upita zadovoljila uvjet (i pojavila se u rezultatu):

ukoliko se NE koristi modifikator NOT - podupit mora rezultirati s barem jednom n-torkom

ukoliko se koristi modifikator NOT - podupit ne smije vratiti niti jednu n-torku

Primjer: Ispis svih dvorana ukoliko DANAS nije obavljen niti jedan ispit

```
SELECT * FROM dvorana
WHERE NOT EXISTS
  (SELECT * FROM ispit
   WHERE datIspit = TODAY)
```

Upitna je svrha postavljanja ovakvog upita. U principu, upiti ovakve vrste koriste se jedino u kombinaciji s koreliranim podupitima. Korelirani podupiti objašnjeni su u sljedećem poglavlju.

4.6.1 Korelirani podupiti

Podupit koji pri evaluaciji koristi vrijednosti vanjskog upita naziva se korelirani podupit (*correlated subquery*). Svaka druga vrsta podupita naziva se nekorelirani podupit (*uncorrelated*).

Svi podupiti u do sad prikazanim primjerima su nekorelirani.

Primjer: ispisati sve predmete za koje je tjedno opterećenje predmeta veće od broja različitih dana u tjednu u kojima se odvija nastava iz tog predmeta

```
SELECT pred.*
FROM pred
WHERE pred.brojSatiTjedno >
      (SELECT COUNT(DISTINCT oznVrstaDan)
       FROM rezervacija
       WHERE rezervacija.sifPred = pred.sifPred)
```

Ovaj podupit je koreliran jer se za njegovu evaluaciju koristi vrijednost atributa **pred.sifPred** koja je rezultat vanjskog upita.

Upit se (konceptualno promatrano) obavlja na sljedeći način: uzima se prva n-torka iz relacije **pred** pa se koristeći vrijednost **pred.sifPred** izračunava broj različitih dana u tjednu za koje je obavljena rezervacija za taj predmet. Ukoliko je broj različitih dana u tjednu iz vanjskog upita veći od dobivenog broja, n-torka iz vanjskog upita se pojavljuje u rezultatu. Postupak se ponavlja za svaku n-torku relacije **pred**.

Svaki **nekorelirani** podupit se obavlja samo jednom, a zatim se njegov rezultat koristi za usporedbu s n-torkama dobivenim vanjskim upitom. **Korelirani** podupiti se moraju obaviti jednom za svaku n-torku iz vanjskog upita.

Obje vrste, i korelirani i nekorelirani podupiti se mogu ugrađivati u upite na sva tri opisana načina (usporedba izraza s rezultatom podupita, ispitivanje da li je neki izraz sadržan u rezultatu podupita, ispitivanje da li u rezultatu podupita postoji barem jedna n-torka).

Primjer: ispisati sve predmete za koje ne postoji niti jedna rezervacija dvorane

```
SELECT pred.sifPred, pred.nazPred
FROM pred
WHERE NOT EXISTS
  ( SELECT *
    FROM rezervacija
    WHERE rezervacija.sifPred = pred.sifPred )
```

Primjer: ispisati podatke o dvoranama za koje ne postoji neka druga dvorana s jednakim kapacitetom

```
SELECT oznDvorana, kapacitet
FROM dvorana
WHERE kapacitet NOT IN
  ( SELECT kapacitet FROM dvorana drugaDvorana
    WHERE drugaDvorana.oznDvorana <> dvorana.oznDvorana )
```

4.6.2 Kombiniranje uvjeta s podupitima

Budući da *subquery* (SELECT naredba koja se koristi kao podupit) može opet biti bilo koja SELECT naredba (osim što ne smije sadržavati *ORDER BY Clause*), SQL naredba može sadržavati podupit u kojeg je ugniježđen podupit, u kojem je opet ugniježđen podupit itd., npr.


```

SELECT ... FROM ...
  WHERE izraz1 >= ALL
    (SELECT ... FROM ...
      WHERE ...
      AND izraz2 NOT IN
        (SELECT ... FROM ...
          WHERE ...
        )
    )
  )

```

Primjer: ispisati podatke svih studenata koji nisu položili niti jedan ispit sa barem prosječnom ocjenom (prosjeck se računa kao prosjek pozitivnih ocjena svih ispita)

```

SELECT *
FROM stud
WHERE mbrStud NOT IN (
  SELECT DISTINCT mbrStud
  FROM ispit
  WHERE ocjena >= (SELECT AVG(ocjena) FROM ispit
                  WHERE ocjena > 1)
)

```

Budući da je uvjet s podupitom vrsta uvjeta (*Condition*) koji se može povezivati s drugim uvjetima pomoću logičkih operatora, podupiti se mogu kombinirati i na sljedeći način:

```

SELECT ... FROM ...
  WHERE EXISTS
    (SELECT ... FROM ...
      WHERE ...
    )
  OR izraz1 IN
    (SELECT ... FROM ...
      WHERE ...
    )

```

Primjeri: ispisati mjesta čiji se poštanski brojevi ne koriste u relaciji stud

```

SELECT * FROM mjesto
WHERE pbr NOT IN (SELECT pbrRod FROM stud)
AND pbr NOT IN (SELECT pbrStan FROM stud)

```

(drugo moguće rješenje je)

```

SELECT * FROM mjesto
WHERE NOT EXISTS (SELECT * FROM stud
                  WHERE pbrRod = mjesto.pbr
                  OR pbrStan = mjesto.pbr)

```

Konačno, podupiti se mogu kombinirati i s uvjetima usporedbe, npr.

```

SELECT ... FROM ...
  WHERE izraz1 >= ALL
    (SELECT ... FROM ...
      WHERE ...
      AND izraz2 NOT IN
        (SELECT ... FROM ...
          WHERE ...
        )
    )
  )
  OR EXISTS
    (SELECT ... FROM ...
      WHERE ...
      AND izraz3 < izraz4
    )
  AND izraz5 > izraz6

```

4.6.3 Podupiti u listi za selekciju

Podupiti se mogu koristiti kao izrazi u listi za selekciju (vidjeti sintaksni dijagram za listu za selekciju - *Select List*). Podupiti u listi za selekciju mogu biti korelirani ili nekorelirani. Kao i kod podupita u *WHERE Clause* dijelu naredbe, korelirani podupit se mora obaviti po jednom za svaku n-torku iz vanjskog upita.

Primjer: uz svako mjesto ispisati broj studenata koji su se u tom mjestu rodili i broj studenata koji u tom mjestu stanuju. Ovo je primjer koreliranog podupita:

```
SELECT mjesto.pbr
      , mjesto.nazMjesto
      , (SELECT COUNT(*) FROM stud WHERE stud.pbrRod = mjesto.pbr)
      , (SELECT COUNT(*) FROM stud WHERE stud.pbrStan = mjesto.pbr)
FROM mjesto
```

Primjer: ispisati broj županija, broj mjesta i broj nastavnika. Ovo je primjer nekoreliranog podupita:

```
SELECT (SELECT COUNT(*) FROM zupanija) brojZupanija
      , (SELECT COUNT(*) FROM mjesto) brojMjesta
      , (SELECT COUNT(*) FROM nastavnik) brojNastavnika
FROM mjesto
WHERE pbr = 10000
```

Odabrana je jedna n-torka iz relacije **mjesto** pomoću koje su se ispisale tražene vrijednosti. Mogla se odabrati bilo koja (postojeća) n-torka iz bilo koje relacije.

4.6.4 Podupite ne treba koristiti bez razloga

Podupite ne treba primjenjivati za rješavanje problema u kojima primjena podupita nije potrebna, kao što je prikazano u sljedećim primjerima:

Ispisati sve podatke o ispitima svih studenata koji stanuju u Koprivnici



```
SELECT * FROM ispit
WHERE mbrStud IN (
  SELECT mbrStud FROM stud
  WHERE pbrStan IN (
    SELECT pbr FROM mjesto
    WHERE nazMjesto = 'Koprivnica'
  )
)
```

Efikasnije i jasnije, upit se rješava na sljedeći način:



```
SELECT ispit.* FROM ispit, stud, mjesto
WHERE ispit.mbrStud = stud.mbrStud
AND stud.pbrStan = mjesto.pbr
AND mjesto.nazMjesto = 'Koprivnica'
```

Ispisati sva mjesta i pripadne nazive županija:



```
SELECT mjesto.pbr
      , mjesto.nazMjesto
      , (SELECT nazZupanija
          FROM zupanija
          WHERE zupanija.sifZupanija = mjesto.sifZupanija)
FROM mjesto
```

Efikasnije i jasnije, upit se rješava na sljedeći način:

```
⇒ SELECT mjesto.pbr
      , mjesto.nazMjesto
      , zupanija.nazZupanija
FROM mjesto, zupanija
WHERE mjesto.sifZupanija = zupanija.sifZupanija
```

4.6.5 Određivanje presjeka i razlike relacija pomoću SELECT naredbe

MjestoRod			MjestoStan		
oznDrz	pbr	nazMjesto	oznDrz	pbr	nazMjesto
HR	10000	Zagreb	HR	10000	Zagreb
HR	51000	Rijeka	HR	47000	Karlovac
HR	31000	Osijek	HR	31000	Osijek
GB	10000	London	GB	10000	London
GB	47000	Glasgow			

Ključ u obje relacije je (oznDrz, pbr)

mjestoRod \cap mjestoStan

```
SELECT * FROM mjestoRod
WHERE EXISTS
  (SELECT mjestoStan.pbr FROM mjestoStan
   WHERE mjestoStan.oznDrz = mjestoRod.oznDrz
     AND mjestoStan.pbr = mjestoRod.pbr
     AND mjestoStan.nazMjesto = mjestoRod.nazMjesto)
```

U obje relacije je ključ **oznDrz, pbr**. Iz toga slijedi da, ukoliko je ispunjen uvjet

```
mjestoStan.oznDrz = mjestoRod.oznDrz
AND mjestoStan.pbr = mjestoRod.pbr
```

tada je ispunjen i uvjet

```
AND mjestoStan.nazMjesto = mjestoRod.nazMjesto
```

te se taj dio može ispustiti:

```
SELECT * FROM mjestoRod
WHERE EXISTS
  (SELECT mjestoStan.pbr FROM mjestoStan
   WHERE mjestoStan.oznDrz = mjestoRod.oznDrz
     AND mjestoStan.pbr = mjestoRod.pbr)
```

operacija presjeka može se riješiti i pomoću spajanja relacija

```
SELECT mjestoRod.* FROM mjestoRod, mjestoStan
WHERE mjestoRod.oznDrz = mjestoStan.oznDrz
  AND mjestoRod.pbr = mjestoStan.pbr
  AND mjestoRod.nazMjesto = mjestoStan.nazMjesto
```

posljednji redak može se ispustiti, jer atributi **oznDrz** i **pbr** čine ključ u relacijama

```
SELECT mjestoRod.* FROM mjestoRod, mjestoStan
WHERE mjestoRod.oznDrz = mjestoStan.oznDrz
  AND mjestoRod.pbr = mjestoStan.pbr
```

u općem se slučaju operacija presjeka ne može riješiti pomoću podupita IN



```
SELECT * FROM mjestoRod
WHERE mjestoRod.oznDrzava IN
      (SELECT mjestoStan.oznDrzava
       FROM mjestoStan
       )
AND mjestoRod.pbr IN
      (SELECT mjestoStan.pbr
       FROM mjestoStan
       )
)
```

Jedino u slučaju da je ključ u relacijama samo jedan atribut, može se koristiti podupit IN

MjestoRod		MjestoStan	
<u>pbr</u>	<u>nazMjesto</u>	<u>pbr</u>	<u>nazMjesto</u>
10000	Zagreb	10000	Zagreb
51000	Rijeka	47000	Karlovac
31000	Osijek	31000	Osijek

```
SELECT * FROM mjestoRod
WHERE mjestoRod.pbr IN
      (SELECT mjestoStan.pbr
       FROM mjestoStan
       )
)
```

mjestoRod \ mjestoStan

MjestoRod			MjestoStan		
<u>oznDrz</u>	<u>pbr</u>	<u>nazMjesto</u>	<u>oznDrz</u>	<u>pbr</u>	<u>nazMjesto</u>
HR	10000	Zagreb	HR	10000	Zagreb
HR	51000	Rijeka	HR	47000	Karlovac
HR	31000	Osijek	HR	31000	Osijek
GB	10000	London	GB	10000	London
GB	47000	Glasgow			

Ključ u obje relacije je (oznDrz, pbr)

```
SELECT * FROM mjestoRod
WHERE NOT EXISTS
      (SELECT mjestoStan.pbr FROM mjestoStan
       WHERE mjestoStan.oznDrz = mjestoRod.oznDrz
         AND mjestoStan.pbr = mjestoRod.pbr
         AND mjestoStan.nazMjesto = mjestoRod.nazMjesto)
)
```

U relacijama je ključ **oznDrz, pbr**. Iz toga slijedi da se posljednji redak može ispustiti

```
SELECT * FROM mjestoRod
WHERE NOT EXISTS
      (SELECT mjestoStan.pbr FROM mjestoStan
       WHERE mjestoStan.oznDrz = mjestoRod.oznDrz
         AND mjestoStan.pbr = mjestoRod.pbr)
)
```

U općem slučaju se operacija razlike ne može riješiti pomoću podupita IN



```
SELECT * FROM mjestoRod
WHERE mjestoRod.oznDrz NOT IN
      (SELECT mjestoStan.oznDrz
       FROM mjestoStan
       )
AND
      mjestoRod.pbr NOT IN
      (SELECT mjestoStan.pbr
       FROM mjestoStan
       )
)
```

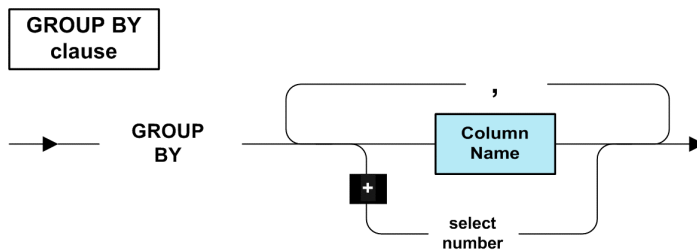
Jedino u slučaju da je ključ relacija samo jedan atribut, može se koristiti podupit IN

MjestoRod		MjestoStan	
<u>pbr</u>	<u>nazMjesto</u>	<u>pbr</u>	<u>nazMjesto</u>
10000	Zagreb	10000	Zagreb
51000	Rijeka	47000	Karlovac
31000	Osijek	31000	Osijek

```
SELECT * FROM mjestoRod
WHERE mjestoRod.pbr NOT IN
      (SELECT mjestoStan.pbr
       FROM mjestoStan
      )
```

4.7 Grupiranje rezultata

Grupiranje se obavlja prema jednom ili više atributa iz relacije (ili više relacija ukoliko su navedene u *FROM Clause*). *GROUP BY Clause* sadrži popis atributa ili izraza prema kojima se obavlja grupiranje.



Grupiranje se obavlja tako da se n-torke koje imaju jednake vrijednosti atributa navedenih u listi za grupiranje, kombiniraju u zajedničku grupu. Za svaku dobivenu grupu, u rezultatu se pojavljuje samo jedna n-torka.

```
SELECT sifPred
FROM rezervacija, dvorana
WHERE rezervacija.oznDvorana = dvorana.oznDvorana
GROUP BY sifPred
```

Konceptualno se upit obavlja tako da se prvo grupiraju sve n-torke prema vrijednostima atributa navedenih listi za grupiranje:

<u>sifPred</u>	<u>oznVrstaDan</u>	<u>oznDvorana</u>	<u>sat</u>	<u>kapacitet</u>
MAT	PO	B-1	8	70
MAT	PO	B-1	9	70
MAT	UT	A-101	9	30
FIZ	SR	A-202	11	40
FIZ	CE	B-1	11	70
RAC	UT	A-102	14	30
RAC	SR	A-201	18	40
TEH	PE	B-5	12	50

Konačni rezultat se dobije tako da se za svaku grupu ispiše samo jedna n-torka

<u>sifPred</u>
MAT
FIZ
RAC
TEH

Napomena: prethodni upit je bio samo primjer grupiranja. Naravno, gornji se rezultat mogao dobiti i na jednostavniji način, naredbom

```
SELECT DISTINCT sifPred
FROM rezervacija
```

Grupiranje je, međutim, vrlo korisno u kombinaciji s agregatnim funkcijama. U sljedećem upitu određuje se, za svaki predmet posebno, ukupan broj mjesta u rezerviranim dvoranama.

```
SELECT sifPred, SUM(kapacitet)
FROM rezervacija, dvorana
WHERE rezervacija.oznDvorana = dvorana.oznDvorana
GROUP BY sifPred
```

Rezultat je

<u>sifPred</u>	<u>(SUM)</u>
MAT	170
FIZ	110
RAC	70
TEH	50

Dopušteno je koristiti sve agregatne funkcije koje su navedene u poglavlju o agregatnim funkcijama.

Važno je sljedeće pravilo: svi atributi ili izrazi koji se nalaze u listi za selekciju, a koji nisu unutar agregatnih funkcija, moraju biti navedeni u **GROUP BY** listi. Međutim, dopušteno je u GROUP BY listi koristiti i one attribute koji se ne nalaze u listi za selekciju, npr.

```
SELECT SUM(kapacitet)
FROM rezervacija, dvorana
WHERE rezervacija.oznDvorana = dvorana.oznDvorana
GROUP BY sifPred
```

Rezultat je

<u>(SUM)</u>
170
110
70
50

Sljedeći upit



```
SELECT sifPred, oznDvorana, SUM(kapacitet)
FROM rezervacija, dvorana
WHERE rezervacija.oznDvorana = dvorana.oznDvorana
GROUP BY sifPred
```

NE VRIJEDI jer se za svaki predmet dobije samo jedna grupa (dakle jedan redak izlaznog rezultata). Budući da se uz jednu šifru predmeta pojavljuje više različitih dvorana, nemoguće je odrediti koju od dvorana koje pripadaju grupi treba ispisati.

<u>sifPred</u>	<u>oznVrstaDan</u>	<u>oznDvorana</u>	<u>sat</u>	<u>kapacitet</u>		
MAT	PO	B-1	8	70	⇒	
MAT	PO	B-1	9	70		
MAT	UT	A-101	9	30		
FIZ	SR	A-202	11	40	⇒	
FIZ	CE	B-1	11	70		
RAC	UT	A-102	14	30	⇒	
RAC	SR	A-201	18	40		
TEH	PE	B-5	12	50	⇒	

<u>sifPred</u>	<u>oznDvorana</u>	<u>SUM</u>
MAT	?	170
FIZ	?	110
RAC	?	70
TEH	?	50

+ U listi atributa prema kojima se obavlja grupiranje mogu se umjesto imena atributa navoditi redni brojevi atributa iz liste za selekciju

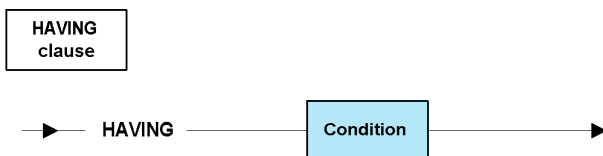
```
SELECT sifPred, sifNastavnik, AVG(ocjena)
FROM ispit
GROUP BY 1, 2
```

+ Osim prema atributima, grupiranje se može obaviti i prema izrazima (*Expression*), ali se u tom slučaju u listi atributa za grupiranje moraju koristiti redni brojevi atributa/izraza iz liste za selekciju.

```
SELECT ocjena*10, COUNT(*)
FROM ispit
GROUP BY 1      -- ne smije se napisati GROUP BY ocjena*10
```

4.8 Postavljanje uvjeta nad grupom zapisa

Za postavljanje uvjeta nad grupom zapisa koristi se *HAVING Clause*. Dok se uz pomoć *WHERE Clause* izdvajaju one **n-torke** koje će formirati grupe definirane u *GROUP BY Clause*, *HAVING Clause* služi za postavljanje uvjeta kojeg dobivene **grupe** moraju zadovoljiti da bi se pojavile u rezultatu. Za razliku od *WHERE* dijela naredbe, u *HAVING* dijelu naredbe dopušteno je koristiti agregatne funkcije. U *HAVING* dijelu naredbe dopušteno je koristiti samo one attribute koji se nalaze u *GROUP BY* listi. Atributi koji se ne nalaze u *GROUP BY* listi smiju se koristiti jedino kao argumenti agregatnih funkcija.



Primjer: ispis šifri predmeta i broja položenih ispita, ali samo onih predmeta za koje je položeno više od dva ispita.

```
SELECT sifPred, COUNT(*)
FROM ispit
WHERE ispit.ocjena > 1
GROUP BY sifPred
HAVING COUNT(*) > 2
```

WHERE dio naredbe određuje koje n-torke će formirati grupe (samo položeni ispiti). *GROUP BY* lista određuje strukturu grupa tj. po kojim atributima se obavlja grupiranje n-torki (sve n-torke koje imaju jednaku šifru predmeta ulaze u jednu grupu). *HAVING* dio naredbe određuje koje od nastalih grupa će biti prihvaćene kao rezultat (samo one grupe u kojima je broj n-torki, tj. *COUNT(*)* veći od dva). Ispisuje se po jedan zapis za svaku grupu koja zadovoljava taj uvjet.

Sljedeća slika prikazuje povezanost *WHERE*, *GROUP BY* i *HAVING* dijela *SELECT* naredbe:

```
SELECT sifPred, COUNT(*)
FROM ispit
WHERE ispit.ocjena > 1
GROUP BY sifPred
HAVING COUNT(*) > 2
```

sifPred	ocjena
1071	3
1071	2
1095	4
1095	1
1095	1
2091	1
2091	3
2091	5
2091	2
2091	3
5048	4
5048	1
5048	5
5048	4



sifPred
1071
1071
1095
2091
2091
2091
2091
5048
5048
5048



U **HAVING** dijelu naredbe dopušteno je koristiti uvjete s podupitima na jednak način kao što se koriste i u *WHERE Clause*.

Primjer: ispisati šifre predmeta za koje je prosjek ocjena veći od ukupnog prosjeka ocjena.

```
SELECT sifPred
FROM ispit
GROUP BY sifPred
HAVING AVG(ocjena) > (SELECT AVG(ocjena) FROM ispit)
```

4.8.1 Ispitivanje postojanja funkcijske zavisnosti na temelju trenutnog sadržaja relacije

Uporabom agregatnih funkcija jednostavno se ispituje vrijedi li neka funkcijska zavisnost u relaciji (POZOR: može se odrediti jedino vrijedi li funkcijska zavisnost za trenutni *sadržaj* relacije, ali ne i vrijedi li za relacijsku shemu). Iz definicije funkcijske zavisnosti proizlazi da funkcijska zavisnost $X \rightarrow Y$ ne vrijedi u relaciji ukoliko u nekoj grupi zapisa s jednakom X vrijednošću postoji više od jedne različite Y vrijednosti.

Kada se ispituje funkcijska zavisnost $X \rightarrow Y$, formira se upit u kojem se zapisi grupiraju prema atributima iz X . Ispisuju se one grupe za koje postoji više od jedne različite vrijednosti od Y .

Sljedećom se naredbom u relaciji **ispit** ispituje funkcijska zavisnost $\{ \text{mbrStud}, \text{datIspit} \} \rightarrow \text{sifPred}$.

```
SELECT mbrStud, datIspit
FROM ispit
GROUP BY mbrStud, datIspit
HAVING COUNT(DISTINCT sifPred) > 1
```

<u>mbrStud</u>	<u>datIspit</u>
36251744	13.01.1996
36251744	21.11.1995

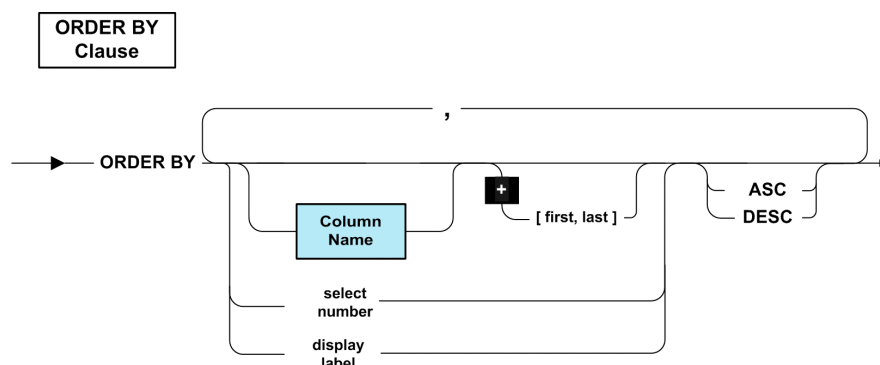
Dvije dobivene n-torke pokazuju da ispitivana funkcijska zavisnost ne vrijedi.

Budući se sljedećom naredbom ne dobiva niti jedna n-torka rezultata, pokazano je da za trenutni sadržaj relacije **ispit** vrijedi funkcijska zavisnost $\{ \text{mbrStud}, \text{datIspit}, \text{sifPred} \} \rightarrow \text{ocjena}$.

```
SELECT mbrStud, datIspit, sifPred
FROM ispit
GROUP BY mbrStud, datIspit, sifPred
HAVING COUNT(DISTINCT ocjena) > 1
```

Funkcijske zavisnosti oblika $X \rightarrow Y$ u kojima skup Y sadrži više od jednog atributa, npr. $Y = \{ A, B \}$, nije moguće ispitati na ovaj način, budući da nije dopuštena uporaba **HAVING COUNT(DISTINCT A, B)**. Međutim, u takvim se slučajevima mogu pojedinačno ispitati funkcijske zavisnosti $X \rightarrow A$ i $X \rightarrow B$, te ukoliko one vrijede, korištenjem pravila o aditivnosti, zaključiti da vrijedi i funkcijska zavisnost $X \rightarrow Y$.

4.9 Poredek rezultata



Rezultat SELECT naredbe se može poredati (sortirati) prema izrazima iz liste za selekciju. Iza ključne riječi ORDER BY navode se atributi ili redni brojevi izraza ili *alias* imena izraza prema kojima se obavlja sortiranje. Opcionalno se uz svaki izraz može navesti smjer sortiranja - ključna riječ ASC ili DESC. ASC (kratica za *ascending*) znači da će se poredak obaviti u uzlaznom smjeru, a DESC (kratica za *descending*) u silaznom. Ukoliko se smjer sortiranja ne navede, smatra se da se sortiranje obavlja uzlazno.

```
SELECT nazMjesto, prezStud
FROM stud, mjesto
WHERE stud.pbrStan = mjesto.pbr
ORDER BY nazMjesto DESC, prezStud ASC
```

<u>nazMjesto</u>	<u>prezStud</u>
Zagreb	Abram
Zagreb	Kolar
Rijeka	Jambrek
Rijeka	Kolar
Rijeka	Novak

ORDER BY lista se može napisati skraćeno, tako da se umjesto imena atributa ili izraza koriste redni brojevi izraza iz liste za selekciju. ORDER BY lista u prethodnom upitu mogla bi se napisati ovako:

```
ORDER BY 1 DESC, 2 ASC
```

Sortiranje se može obaviti jedino po atributima ili izrazima navedenim u listi za selekciju.



```
SELECT nazMjesto, prezStud
FROM stud, mjesto
WHERE stud.pbrStan = mjesto.pbr
ORDER BY nazMjesto DESC, prezStud ASC, imeStud ASC
```



Sortiranje se može obaviti prema podnizu nekog atributa tipa koji je tipa niza znakova.

```
SELECT nazMjesto, prezStud
FROM stud, mjesto
WHERE stud.pbrStan = mjesto.pbr
ORDER BY nazMjesto[2,5]
```

Sortiranje se može obaviti i prema izrazima (*Expression*) iz liste za selekciju, ali se u tom slučaju u listi atributa za poredak moraju koristiti ili redni brojevi ili *alias* imena atributa/izraza iz liste za selekciju.

```
SELECT sifPred, ocjena*10
FROM ispit
ORDER BY 2

SELECT sifPred, ocjena*10 umnozak
FROM ispit
ORDER BY umnozak
```

Kvalifikator *FIRST max* može se korisno upotrijebiti u kombinaciji s *ORDER BY*. Moguće je npr. dohvatiti deset studenata s najboljim (ili najlošijim) prosjekom

```
SELECT FIRST 10 mbrStud, AVG(ocjena) prosjek
FROM ispit
GROUP BY mbrStud
ORDER BY prosjek DESC -- ili ASC, za 10 studenata s najlošijim prosjekom
```

4.10 Pohrana rezultata upita u privremenu relaciju

INTO TEMP Clause koji se može vidjeti u sintaksnom dijagramu za *SELECT* naredbu služi za istovremeno kreiranje i punjenje privremene relacije.

Privremena relacija ima sve karakteristike trajne relacije kreirane naredbom *CREATE TABLE*, osim što se ne vidi u rječniku podataka, te biva uništena u trenutku kada završava program u kojem je kreirana (npr. kad se napusti SQL Editor ROM alata). Privremena relacija se može uništiti i uporabom naredbe *DROP TABLE*.

Schema privremene relacije određena je sadržajem liste za selekciju naredbe kojom je kreirana. Tipovi podataka privremene relacije određeni su tipovima podataka iz liste za selekciju, a nazivi atributa odgovaraju nazivima selektiranih atributa ili *alias* imenima koja su im pridružena. Članovi liste za selekciju koji nisu atributi (dakle izrazi - *Expression*) **moraju** biti imenovani uz pomoć *alias* imena.

Sljedeća naredba kreira privremenu relaciju koja sadrži matične brojeve studenata i pripadne prosječne ocjene, te je istovremeno napuni s podacima koji su rezultat *SELECT* naredbe.

```
SELECT mbrStud, AVG(ocjena) prosjecnaOcjena
FROM ispit
WHERE ocjena > 1
GROUP BY mbrStud
INTO TEMP prosjeci
```

Korisnik koji je izveo prethodnu naredbu može privremenu relaciju **prosjeci** s atributima **mbrStud** i **prosjecnaOcjena** koristiti kao bilo koju drugu relaciju, sve do trenutka kada program terminira ili dok relaciju **prosjeci** ne uništi pomoću naredbe *DROP TABLE*. Npr.

```
SELECT MAX(prosjecnaOcjena)
FROM prosjeci
```

4.11 Određivanje unije relacija pomoću naredbe *SELECT*

U sintaksnom dijagramu za naredbu *SELECT* vidi se da je moguće dvije ili više *SELECT* naredbi povezati pomoću **UNION** operatora. Rezultati *SELECT* naredbi pri tome moraju biti unijski kompatibilni na nivou tipova podataka, te samo posljednja *SELECT* naredba u nizu smije sadržavati *ORDER BY Clause* i *INTO TEMP Clause*.

Bez ključne riječi *ALL*, iz rezultata će biti izbačeni duplikati n-torki. S ključnom riječi *ALL* zadržavaju se sve n-torke iz svih *SELECT* naredbi.

```
SELECT pbr, nazMjesto
FROM mjestoRod
UNION
SELECT pbr, nazMjesto
FROM mjestoStan
```

Prethodna naredba primjer je za operaciju unije u relacijskoj algebri: **mjestoRod \cup mjestoStan**

Ukoliko se *UNION* u prethodnom primjeru zamijeni s *UNION ALL*, upit ne bi obavio operaciju unije na ispravan način, jer iz rezultata ne bi eliminirao n-torke koje se javljaju dva ili više puta.

ORDER BY Clause se može smjestiti isključivo iza posljednje *SELECT* naredbe. Pri tome se u *ORDER BY* listi ne smiju koristiti nazivi atributa, već njihovi redni brojevi unutar liste za selekciju. *INTO TEMP Clause* se također dodaje isključivo iza posljednje *SELECT* naredbe.

5. Indeksi u jeziku SQL

Brzina pristupa podacima u relaciji je važno svojstvo sustava za upravljanje podacima. Najjednostavniji način pristupa, sekvencijalna pretraga, u većini slučajeva ne zadovoljava.

Kreiranjem indeksa formira se struktura B-stabla koja omogućava nesekvencijalni pristup do n-torke u relaciji. Nesekvencijalni pristup moguć je prema vrijednostima onih atributa nad kojima je izgrađena indeksna struktura.

Nad jednom relacijom može biti izgrađeno više indeksa, od kojih svaki može sadržavati jedan ili više atributa.

- ako je nad relacijom R kreiran indeks za atribut A, može se nesekvencijalno pristupiti do n-torki uz korištenje atributa A kao ključa za dohvat.
- ako je nad relacijom R kreiran kompozitni indeks za npr. attribute A, B, C, do n-torke se može pristupiti korištenjem bilo koje od navedenih kombinacija atributa kao ključa za dohvat - A, AB, ABC.

Prilikom postavljanja upita u jeziku SQL, sustav sam određuje koji će se od indeksa iskoristiti za pristup. Ukoliko odgovarajućeg indeksa nema, pristup n-torkama može biti jedino sekvencijalan.

Osim radi poboljšanja performanci sustava, indeksi se kreiraju i radi osiguranja jedinstvenosti vrijednosti atributa u relaciji (npr. u relaciji mjesto ne smiju postojati dva grada s poštanskim brojem jednakim 10000).

Jedinstvenost vrijednosti u indeksu

Ukoliko je indeks kreiran kao indeks s jedinstvenim vrijednostima (*distinct index*), sustav ne dopušta da se u relaciji pojave dvije n-torke koje bi imale jednake vrijednosti atributa nad kojim je izgrađen takav indeks. Ovakva karakteristika indeksa koristi se za osiguravanje jedinstvenosti ključa u relaciji.

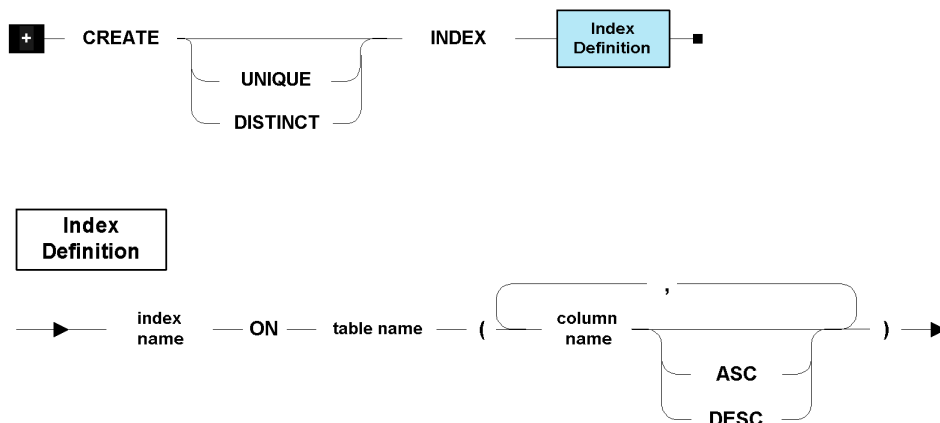
Utjecaj poretka vrijednosti u indeksu na mogućnost korištenja indeksa za sortiranje

Indeks može biti izgrađen tako da su vrijednosti u indeksnim blokovima poredane od manjih prema većim, ili obratno. IBM Informix SUBP može indeks pretraživati od naprijed ili straga, tako da se indeks može koristiti za sortiranje zapisa u smjeru u kojem su poredane vrijednosti u indeksnim blokovima, ali i u obrnutom smjeru. To znači da poredak vrijednosti u indeksnim blokovima indeksa sastavljenih od samo jednog atributa nije bitan. Ako je indeks sastavljen od više atributa, te se prema tim atributima obavlja sortiranje, o poretku vrijednosti u indeksu ovisi mogućnost korištenja tog indeksa prilikom sortiranja. Npr. ukoliko je indeks kreiran za attribute x DESC, y DESC, tada se taj indeks može koristiti za sortiranje u smjeru x DESC, y DESC, te za sortiranje u smjeru x ASC, y ASC, ali ne i za sortiranje oblika x ASC, y DESC ili x DESC, y ASC. Indeks koji omogućava posljednja dva navedena oblika sortiranja je indeks oblika: x DESC, y ASC (također je to moguće ostvariti i indeksom oblika: x ASC, y DESC).

Uporaba indeksa

Opći oblik naredbe za kreiranje indeksa je sljedeći:

CREATE INDEX



Primjer:

```
CREATE DISTINCT INDEX uniqueZupanja ON zupanja (nazZupanja)
```

Uništiti indeks se može naredbom

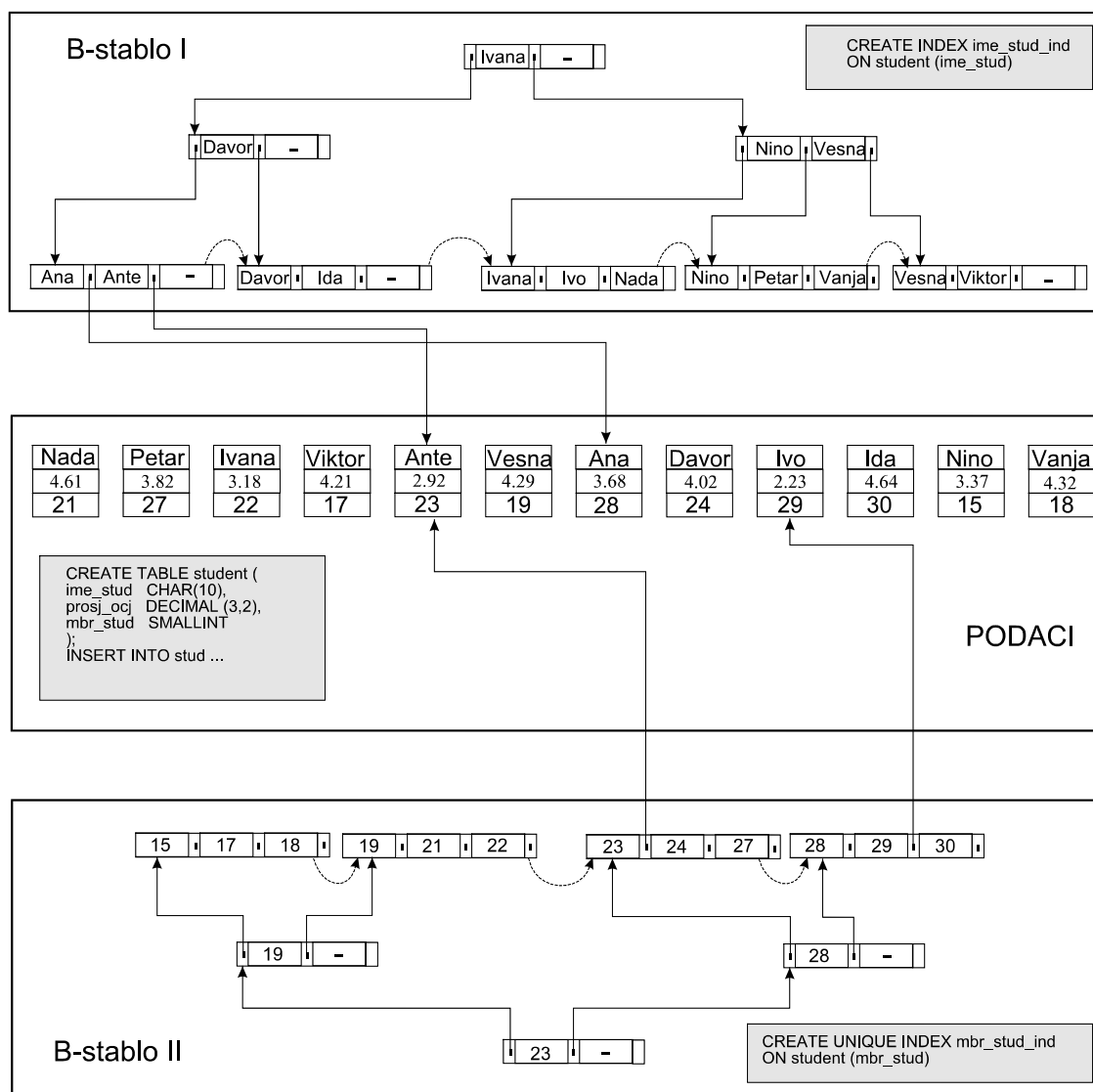
```
DROP INDEX indexName
```

Indekse bi u principu trebalo primjenjivati u sljedećim slučajevima:

- za attribute prema kojima se obavlja spajanje relacija
- za attribute koji se često koriste za postavljanje uvjeta selekcije
- za attribute prema kojima se često obavlja grupiranje ili sortiranje

Na sljedećoj slici prikazana je (strogo logički gledano!) struktura dvaju B-stabla koja su izgrađena nad istom relacijom. Sustav može efikasno pristupati podacima prema ključu pretrage koji sadrži atribut **ime_stud** ili atribut **mbr_stud**. Ako ključ pretrage sadrži atribut **prosj_ocj**, pretraga može biti jedino sekvencijalna.

Radi povećanja preglednosti slike, nisu povučene sve crte koje spajaju pokazivače s objektima na koje oni pokazuju.



Listovi B-stabla često su povezani dodatnim pokazivačima (na slici crtkano) da bi se omogućio dohvat zapisa čiji ključ pripada nekom intervalu vrijednosti (npr. za upit koji sadrži uvjet dohvata oblika: **ime_stud BETWEEN 'Davor' AND 'Nada'**).

Prilikom kreiranja indeksa treba voditi računa i o nekim njihovim negativnim aspektima, te ih treba koristiti samo tamo gdje je njihova uporaba opravdana, jer:

- indeksi zauzimaju značajan prostor
- ažuriranje vrijednosti atributa nad kojima je izgrađen indeks traje znatno dulje nego ažuriranje vrijednosti nad kojima nema indeksa (ovdje treba razlikovati attribute prema kojima se pronalaze n-torke koje treba ažurirati, od atributa čije se vrijednosti ažuriraju)

Indekse ne bi trebalo primjenjivati ukoliko:

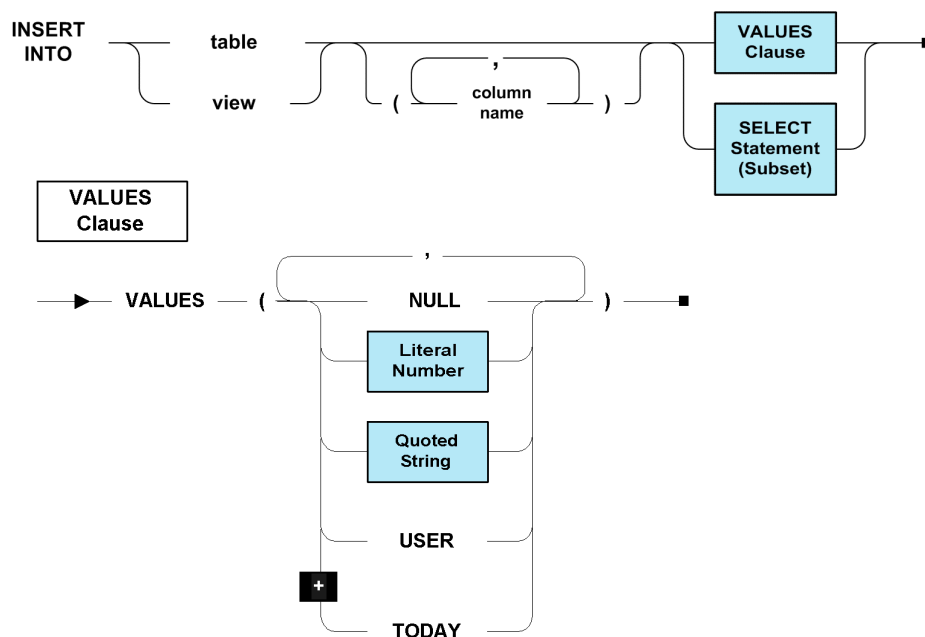
- vrijednosti atributa za kojeg se gradi indeks imaju relativno mali broj različitih vrijednosti (npr. **spol_osobe** s dopuštenim vrijednostima **M**, **Ž**, u relaciji s 30 000 n-torki)
- u relaciji predstoji velik broj upisa, izmjena ili brisanja n-torki. Preporučljivo je u takvim slučajevima postojeće indekse izbrisati, te ih ponovo izgraditi tek nakon obavljenih promjena na podacima
- relacija sadrži vrlo mali broj n-torki (npr. do stotinu). U takvim slučajevima sustav lakše pristupa sekvencijalnom pretragom, nego prolaskom kroz strukturu B-stabla

6. INSERT naredba

6.1 Načini korištenja INSERT naredbe

Naredba je namijenjena za upis jedne ili više n-torki u relaciju. Opći oblik INSERT naredbe je definiran sljedećim sintaksnim dijagramima:

INSERT



Lista s nazivima atributa (*column name*) navodi se u slučaju kada je redoslijed vrijednosti koje se upisuju u relaciju različit od redoslijeda atributa u relaciji ili se ne namjeravaju navesti sve vrijednosti za n-torku. Oblikom INSERT naredbe s *VALUES Clause* upisuje se jedna n-torka, dok se drugim oblikom (korištenjem *SELECT Statement*) upisuju sve n-torke koje su dobivene kao rezultat navedene SELECT naredbe. SELECT naredba kojom se određuju n-torke za upis u relaciju ne smije u svom FROM dijelu sadržavati ime relacije u koju se podaci upisuju. Također, SELECT naredba ne smije sadržavati *INTO TEMP Clause* niti *ORDER BY Clause*.

```
CREATE TABLE mjestoStanovanja (  
    pbr          INTEGER  
    , nazMjesto  CHAR(40)  
);
```

Primjeri:

```
INSERT INTO mjestoStanovanja VALUES (10000, 'Zagreb')

INSERT INTO mjestoStanovanja (nazMjesto, pbr) VALUES ('Zagreb', 10000)

INSERT INTO mjestoStanovanja (pbr) VALUES (10000)
```

U prethodnoj naredbi će za naziv mjesta biti postavljen NULL. Ukoliko bi uz atribut **nazMjesto**, prilikom definicije relacije, bila definirana *default* vrijednost, za naziv mjesta bi bila postavljena ta *default* vrijednost, a ne NULL.

```
INSERT INTO mjestoStanovanja
  SELECT DISTINCT mjesto.pbr, mjesto.nazMjesto
  FROM mjesto, stud
  WHERE mjesto.pbr = stud.pbrStan
```

6.2 INSERT naredba i SERIAL tip podatka

Ako se za vrijednost atributa tipa SERIAL pomoću INSERT naredbe upiše vrijednost 0, SUBP samostalno određuje serijski broj (prvi sljedeći nakon najveće do tada upisane vrijednosti, pri čemu se kao najveća vrijednost uzima najveća do tada upisana vrijednost, bez obzira je li n-torka s tom vrijednosti u međuvremenu obrisana). Ukoliko se upiše broj različit od 0, u relaciju će se upisati unesena vrijednost (neće biti generiran serijski broj). SERIAL tip podatka za atribut ne implicira da je taj atribut ključ relacije.

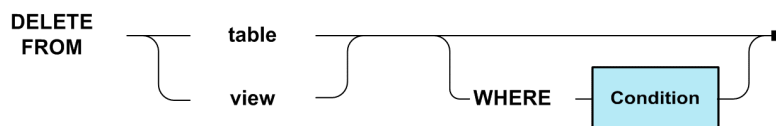
```
CREATE TABLE stavka (
  rbrStavke SERIAL
, nazivStavke CHAR(20)
);
```

INSERT INTO stavka VALUES (0, '1. stavka')	→	1	1. stavka
INSERT INTO stavka VALUES (0, '2. stavka')	→	2	2. stavka
INSERT INTO stavka VALUES (200, '3. stavka')	→	200	3. stavka
INSERT INTO stavka VALUES (0, '4. stavka')	→	201	4. stavka
INSERT INTO stavka VALUES (200, '5. stavka')	→	200	5. stavka
INSERT INTO stavka VALUES (0, '6. stavka')	→	202	6. stavka

7. DELETE naredba

Naredbom se briše jedna ili više n-torki iz relacije. Opći oblik naredbe je:

DELETE



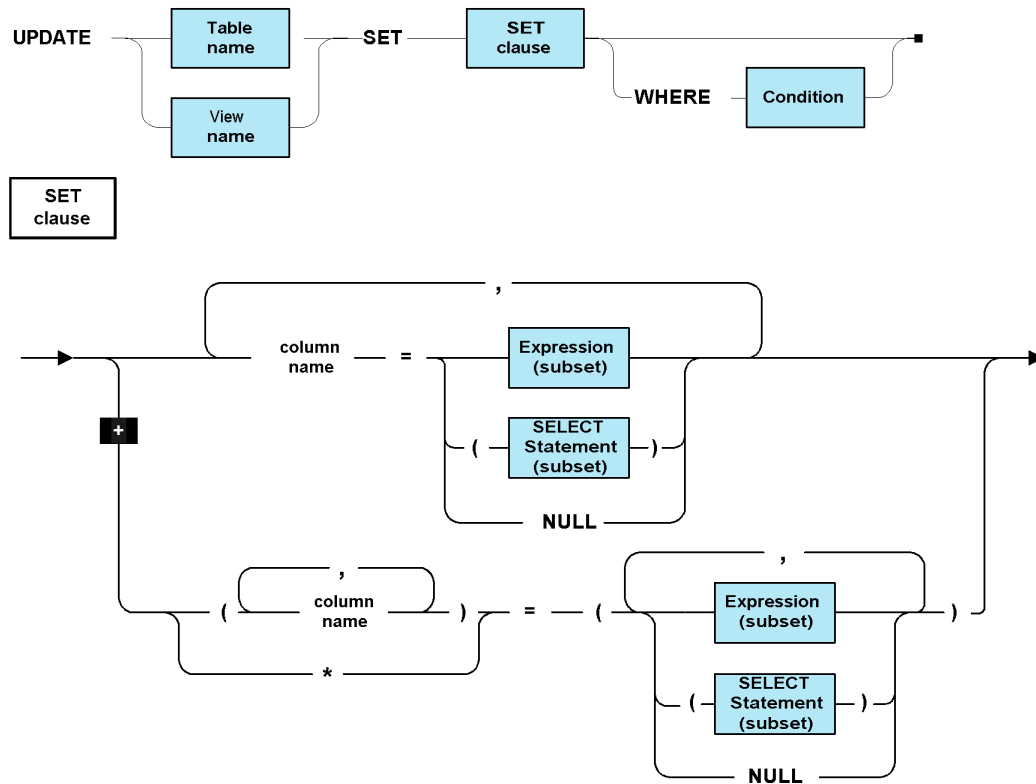
Condition određuje koje n-torke će biti obrisane. Ukoliko se uvjet ne navede, sve n-torke iz relacije će biti obrisane. Uvjet može biti sastavljen kao i u WHERE dijelu SELECT naredbe, ali ne smije sadržavati podupite koji bi u svom FROM dijelu sadržavali relaciju iz koje se zapisi brišu.

Primjer: brisanje svih n-torki iz relacije **zupanija**, čije se šifre ne koriste u relaciji **mjesto**

```
DELETE FROM zupanija
  WHERE sifZupanija NOT IN
    ( SELECT sifZupanija FROM mjesto )
```

8. UPDATE naredba

SQL naredba za izmjenu jedne ili više vrijednosti atributa u jednoj ili više n-torki relacije. Opći oblik naredbe je:



Pomoću sljedećih primjera ilustrirani su karakteristični načini pridruživanja vrijednosti atributima relacije.

Primjer: ažurira se relacija s imenom *table*. Atributu *atr1* pridružuje se vrijednost izraza *expr1*, a atributu *atr2* vrijednost izraza *expr2*. Atributu *atr3* pridružuje se vrijednost dobivena pomoću SELECT naredbe. Ažuriraju se one n-torke relacije *table* koje zadovoljavaju navedeni *condition*.

```

UPDATE table SET atr1 = expr1,
               atr2 = expr2,
               atr3 = (SELECT expr3 FROM ... WHERE ...)
WHERE condition

```

Primjer: ažurira se relacija s imenom *table*. Atributu *atr4* pridružuje se vrijednost izraza *expr4*, a atributu *atr5* vrijednost dobivena SELECT naredbom. Atributima *atr6* i *atr7* pridružuju se vrijednosti koje su dobivene jednom SELECT naredbom. Ažuriraju se one n-torke relacije *table* koje zadovoljavaju navedeni *condition*.

```

UPDATE table SET (atr4, atr5) = (expr4, (SELECT expr5 FROM ... WHERE ...)),
               (atr6, atr7) = ((SELECT expr6, expr7 FROM ... WHERE ...))
WHERE condition

```

Primjer: ažurira se relacija s imenom *table*. U ovom primjeru se umjesto taksativnog navođenja imena svih atributa relacije koristi zvjezdica. Značenje naredbe je: vrijednosti atributa relacije *table* postaviti na vrijednosti koje su navedene na desnoj strani znaka =. Redoslijed pridruživanja vrijednosti je jednak redoslijedu atributa prilikom definiranja relacije *table*.

```

UPDATE table SET * = ( expr1,
                      expr2,
                      (SELECT expr3 FROM ... WHERE ...),
                      expr4, (SELECT expr5 FROM ... WHERE ...),
                      (SELECT expr6, expr7 FROM ... WHERE ...)
                    )
WHERE condition

```

SELECT naredba kojom se određuje nova vrijednost smije vratiti samo jednu n-torku. U listi za selekciju treba biti onoliko atributa ili izraza koliko je potrebno da bi se odgovarajući broj novih vrijednosti pridružio odgovarajućem broju atributa relacije koja se ažurira.

Pomoću *Condition* određuje se koje će n-torke biti ažurirane. Ukoliko se uvjet ne navede, bit će ažurirane sve n-torke iz relacije. WHERE uvjet može sadržavati podupite, ali oni ne smiju u svom FROM dijelu sadržavati relaciju čije se n-torke ažuriraju, što znači da se ne smije obavljati upit nad relacijom čiji se zapisi mijenjaju.

Primjer: svim n-torkama u relaciji **mjesto**, čiji je poštanski broj veći od 30000, nazive mjesta promijeniti tako da se ispred postojećeg naziva doda znakovna konstanta 'GRAD-', a poštanski broj uvećati za 10. Prikazana su tri oblika rješenja:

```
UPDATE mesto
SET nazMjesto = 'GRAD-' || nazMjesto
, pbr = pbr + 10
WHERE pbr > 30000

UPDATE mesto
SET * = (pbr + 10, 'GRAD-' || nazMjesto, sifZupanija)
WHERE pbr > 30000

UPDATE mesto
SET (nazMjesto, pbr) =
('GRAD-' || nazMjesto, pbr + 10)
WHERE pbr > 30000
```

Povećanje koeficijenta za iznos od 0.5 za sve nastavnike kojima je koeficijent između 2.5 i 3.5, a u posljednje 3 godine obavili su više od 5 ispita.

```
UPDATE nastavnik SET koef = koef + 0.5
WHERE koef BETWEEN 2.5 AND 3.5 AND
sifNastavnik IN (
SELECT sifNastavnik
FROM ispit
WHERE datIspit >=
MDY(MONTH(TODAY), DAY(TODAY), YEAR(TODAY)-3)
GROUP BY sifNastavnik
HAVING COUNT(*) > 5
)
```

Nastavnicima koji su najviše puta srušili studente na ispitu, povećati koeficijent za 20 %

```
UPDATE nastavnik SET koef = koef * 1.2
WHERE sifNastavnik IN (
SELECT sifNastavnik FROM ispit
WHERE ocjena = 1
GROUP BY sifNastavnik
HAVING COUNT(*) >= ALL (
SELECT COUNT(*) FROM ispit
WHERE ocjena = 1
GROUP BY sifNastavnik
)
)
```

U sljedećem primjeru prikazano je kako se u relaciji **mjesto**, stari poštanski brojevi mogu zamijeniti novim, uz pomoć vrijednosti iz tablice za konverziju.

Mjesto		Konverzija		
<u>pbr</u>	<u>nazMjesto</u>	<u>stariPbr</u>	<u>noviPbr</u>	<u>noviNazMjesto</u>
41000	ZAGREB	41000	10000	Zagreb
51400	PAZIN	51400	52000	Pazin
52000	PULA	52000	52100	Pula
54000	OSIJEK	54000	31000	Osiijek

Zamjena starih poštanskih brojeva novim vrijednostima

```
UPDATE mjesto SET pbr = (SELECT noviPbr FROM konverzija
                          WHERE stariPbr = mjesto.pbr)
```

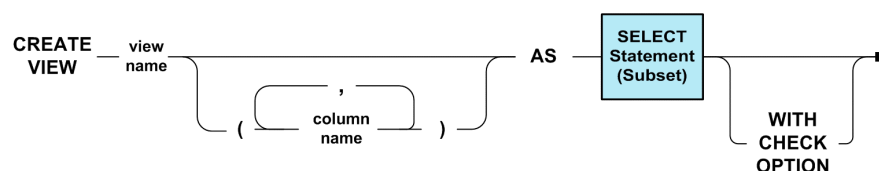
Napomena: ukoliko je nad poštanskim brojem u relaciji **mjesto** izgrađen DISTINCT indeks, potrebno ga je, uz ovakav sadržaj relacija, prije obavljanja gornje naredbe uništiti (vidjeti zapise PULA i PAZIN). Nakon obavljanja naredbe može se indeks ponovno izgraditi.

Sljedećim se upitom istovremeno zamjenjuju i nazivi mjesta novim sadržajem.

```
UPDATE mjesto SET (pbr, nazMjesto) = ((SELECT noviPbr, noviNazMjesto
                                       FROM konverzija
                                       WHERE stariPbr = mjesto.pbr))
```

9. VIEW (pogled, virtualna relacija)

CREATE VIEW



U sintaksnim dijagramima se često kao alternativa za korištenje relacije (tablice) nudi mogućnost korištenja pogleda (virtualne tablice, *view*). Pogled je objekt u bazi podataka čije je ponašanje i upotreba vrlo slična ponašanju i upotrebi relacija.

Sadržaj i shema pogleda određeni su pomoću SELECT naredbe. Uz pomoć liste naziva kolona (*column name*) mogu se eksplicitno definirati nazivi atributa pogleda (inače se kao nazivi atributa pogleda uzimaju nazivi atributa iz SELECT naredbe).

Primjer: kreiranje pogleda koji sadrži zapise iz relacije **stud** čije je mjesto stanovanja grad 'Bjelovar'

```
CREATE VIEW studStanBjelovar AS
SELECT stud.* FROM stud, mjesto
WHERE stud.pbrStan = mjesto.pbr
AND mjesto.nazMjesto = 'Bjelovar'
```

Primjer: korištenje virtualne relacije **studStanBjelovar**

```
SELECT * FROM studStanBjelovar
WHERE imeStud MATCHES 'T*'
```

"Sadržaj" pogleda evaluira se u trenutku izvođenja upita na temelju sadržaja objekta (obično se radi o relaciji) nad kojim je pogled definiran. Svaka promjena obavljena u "temeljnoj" relaciji, automatski se reflektira i na rezultatima onih SQL naredbi u kojima se koristi pogled koji je nad "temeljnom" relacijom definiran.

Po tome se pogled bitno razlikuje od privremene relacije čiji bi se sadržaj dobio kopiranjem "temeljne" relacije jer se promjene koje se obave nad temeljnom relacijom nakon kopiranja neće reflektirati na sadržaj privremene relacije.

Opcijom WITH CHECK OPTION onemogućava se takva izmjena podataka korištenjem pogleda, kojom bi se vrijednosti podataka postavile na vrijednosti koje ne bi zadovoljavale uvjet iz definicije pogleda.

10. NULL vrijednosti

10.1 IS NULL operator

Operator IS NULL se koristi za ispitivanje da li je vrijednost nekog atributa postavljena na NULL vrijednost

```
column IS NULL
ili
column IS NOT NULL
```

Rezultat ispitivanja je TRUE ili FALSE.

ISPLATA

sifra	datum	koef	iznos
NULL	NULL	0.5	50
1	01.01.95	1.0	100
NULL	01.01.97	1.5	200
1	01.01.96	NULL	250
NULL	NULL	2.0	NULL
2	01.01.96	2.5	300
2	NULL	NULL	NULL
2	NULL	3.0	300

```
SELECT * FROM isplata
WHERE koef IS NULL
```

sifra	datum	koef	iznos
1	01.01.96	NULL	250
2	NULL	NULL	NULL

10.2 Uvjeti selekcije i spajanja u prisustvu NULL vrijednosti

Sve n-torke čije vrijednosti atributa uvrštene u uvjet za dohvat ili uvjet spajanja daju rezultat TRUE, kvalificiraju se kao rezultat SELECT naredbe. n-torke za koje se uvjet za dohvat ili uvjet za spajanje evaluira kao FALSE ili NULL, ne ulaze u rezultat.

Također, u naredbama UPDATE ili DELETE, n-torke za koje se uvjet za dohvat evaluira kao TRUE, kvalificiraju se za izmjenu, odnosno brisanje. n-torke za koje se uvjet evaluira kao FALSE ili NULL, neće biti izmijenjene, odnosno obrisane.

```
SELECT * FROM isplata
WHERE koef * iznos <> 100
```

sifra	datum	koef	iznos
NULL	NULL	0.5	50
NULL	01.01.97	1.5	200
2	01.01.96	2.5	300
2	NULL	3.0	300

10.3 Agregatne funkcije i NULL vrijednosti

COUNT (*) broji n-torke, te eventualna pojava NULL vrijednosti u n-torki ne utječe na rezultat
COUNT (x) broj n-torki u kojima atribut x ima vrijednost različitu od NULL
COUNT (DISTINCT x) broj n-torki s različitim vrijednostima od x (n-torke čija je vrijednost atributa x jednaka NULL vrijednosti se ne broje)

SUM (x)
SUM (DISTINCT x)
AVG (x)

AVG (DISTINCT x)

MAX (x)

MIN (x)

RANGE (x)

STDEV (x)

VARIANCE (x)

bez obzira koristi li se ključna riječ DISTINCT, funkcije ignoriraju n-torke u kojima **x** poprima vrijednost **NULL** i izračunavaju rezultat na temelju ostalih vrijednosti, osim ako je vrijednost od **x** postavljena na **NULL** za sve n-torke: tada je rezultat **NULL**.

Primjer:

ISPLATA

sifra	datum	koef	iznos
NULL	NULL	0.5	50
1	01.01.95	1.0	100
NULL	01.01.97	1.5	200
1	01.01.96	NULL	250
NULL	NULL	2.0	NULL
2	01.01.96	2.5	300
2	NULL	NULL	NULL
2	NULL	3.0	300

UPLATA

sifra	datum	koef	iznos
1	01.01.90	0.8	400
NULL	01.01.93	0.9	150
4	01.01.95	1.5	NULL

```
SELECT COUNT(*) FROM isplata                → 8

SELECT COUNT(iznos) FROM isplata            → 6

SELECT COUNT(DISTINCT sifra) FROM isplata    → 2

SELECT COUNT(DISTINCT iznos) FROM uplata
WHERE sifra = 4                             → 0

SELECT SUM(iznos) FROM isplata              → 1200.00

SELECT SUM(iznos) FROM isplata
WHERE koef = 2.0                           → NULL
```

10.4 Grupiranje i NULL vrijednosti

Ukoliko vrijednosti izraza prema kojima se obavlja grupiranje, za neke dvije n-torke predstavljaju međusobnu kopiju, te dvije n-torke ulaze u istu grupu.

ISPLATA

sifra	datum	koef	iznos
NULL	NULL	0.5	50
1	01.01.95	1.0	100
NULL	01.01.97	1.5	200
1	01.01.96	NULL	250
NULL	NULL	2.0	NULL
2	01.01.96	2.5	300
2	NULL	NULL	NULL
2	NULL	3.0	300

```
SELECT sifra, datum, COUNT(*)
FROM isplata
GROUP BY sifra, datum
```

sifra	datum	(count(*))
NULL	NULL	2
NULL	01.01.97	1
1	01.01.95	1
1	01.01.96	1
2	NULL	2
2	01.01.96	1

10.5 Podupiti i NULL vrijednosti

Naročitu pažnju pri korištenju podupita čiji rezultat može sadržavati NULL vrijednosti treba obratiti na uvjete za dohvat koji imaju jedan od ova dva oblika:

```
WHERE expression relationalOperator ALL (subquery)
i
WHERE expression NOT IN (subquery) .
```

Ukoliko je rezultat podupita skup vrijednosti u kojima se nalazi i NULL vrijednost, rezultat evaluacije uvjeta za dohvat u oba gore navedena slučaja će uvijek biti NULL vrijednost, te se n-torka vanjskog upita neće pojaviti u rezultatu.

Primjeri:

ISPLATA				UPLATA			
sifra	datum	koef	iznos	sifra	datum	koef	iznos
NULL	NULL	0.5	50	1	01.01.90	0.8	400
1	01.01.95	1.0	100	NULL	01.01.93	0.9	150
NULL	01.01.97	1.5	200	4	01.01.95	1.5	NULL
1	01.01.96	NULL	250				
NULL	NULL	2.0	NULL				
2	01.01.96	2.5	300				
2	NULL	NULL	NULL				
2	NULL	3.0	300				

```
SELECT * FROM uplata
WHERE iznos > ANY (SELECT iznos FROM isplata)
```

sifra	datum	koef	iznos
1	01.01.90	0.8	400
NULL	01.01.93	0.9	150

```
SELECT * FROM uplata
WHERE iznos > ALL (SELECT iznos FROM isplata)
```

nema zapisa

```
SELECT * FROM uplata
WHERE sifra IN (SELECT sifra FROM isplata)
```

sifra	datum	koef	iznos
1	01.01.90	0.8	400

```
SELECT * FROM uplata
WHERE sifra NOT IN (SELECT sifra FROM isplata)
```

nema zapisa

10.6 Poredak rezultata upita u prisustvu NULL vrijednosti

NULL vrijednost je "manja" pri poretku od svake druge vrijednosti. Zbog toga pri sortiranju rezultata u uzlaznom nizu, NULL vrijednost dolazi prije svih ostalih vrijednosti. Pri sortiranju rezultata u silaznom nizu, NULL vrijednost dolazi iza svih ostalih vrijednosti.

Primjer:

ISPLATA

<u>sifra</u>	<u>datum</u>	<u>koef</u>	<u>iznos</u>
NULL	NULL	0.5	50
1	01.01.95	1.0	100
NULL	01.01.97	1.5	200
1	01.01.96	NULL	250
NULL	NULL	2.0	NULL
2	01.01.96	2.5	300
2	NULL	NULL	NULL
2	NULL	3.0	300

```
SELECT * FROM isplata
ORDER BY sifra, datum
```

<u>sifra</u>	<u>datum</u>	<u>koef</u>	<u>iznos</u>
NULL	NULL	0.5	50
NULL	NULL	2.0	NULL
NULL	01.01.97	1.5	200
1	01.01.95	1.0	100
1	01.01.96	NULL	250
2	NULL	3.0	300
2	NULL	NULL	NULL
2	01.01.96	2.5	300

10.7 NULL konstanta

Ključna riječ NULL može se u INSERT i UPDATE naredbama koristiti da bi se vrijednost nekog atributa postavila na NULL.

```
INSERT INTO mjesto VALUES (99999, NULL)
```

```
UPDATE isplata SET iznos = NULL
WHERE datum IS NULL OR koef < 1.0
```

11. Vanjsko spajanje

Kod prirodnog spajanja se relacije koje se spajaju tretiraju simetrično (jednako su "vrijedne")

STUD		
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>
1	Horvat	10000
2	Kolar	21000
3	Novak	51000

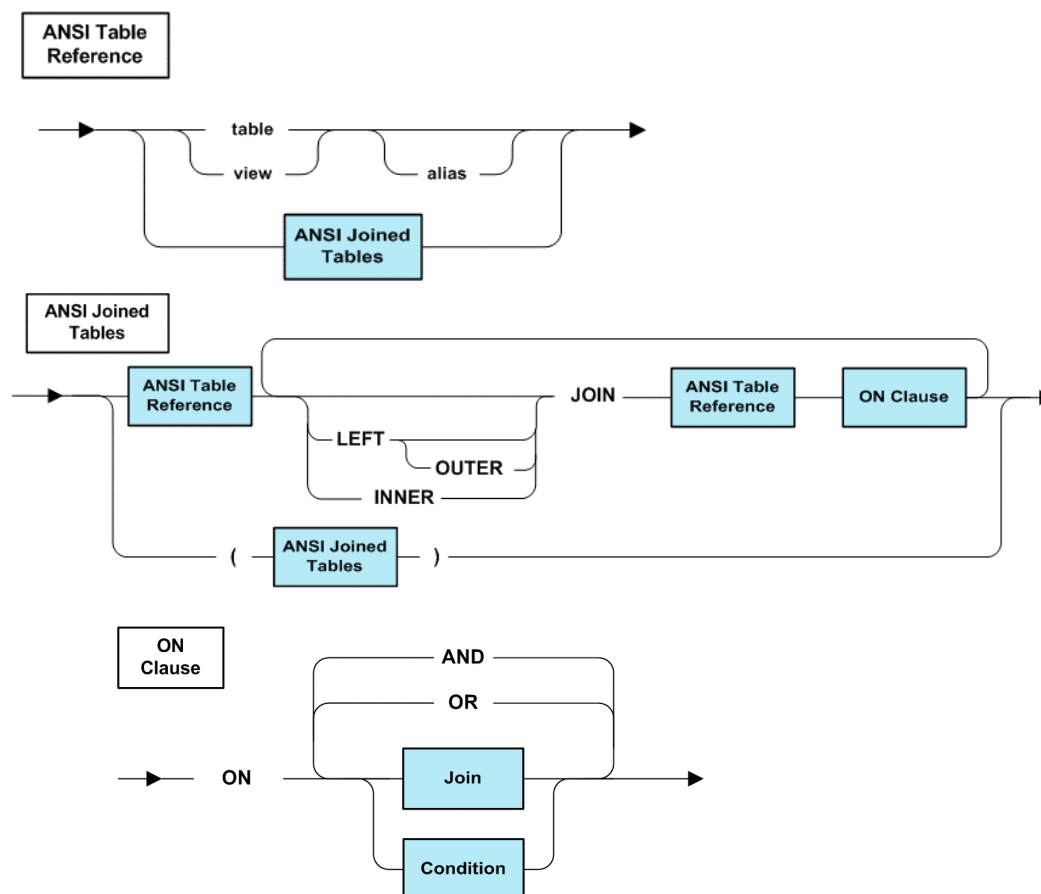
MJESTO	
<u>pbr</u>	<u>nazMjesto</u>
10000	Zagreb
51000	Rijeka
31000	Osijek

```
SELECT stud.*, mjesto.*
FROM stud, mjesto
WHERE stud.pbr = mjesto.pbr
```

Ukoliko za n-torku iz relacije **stud** ne postoji odgovarajuća n-torka iz relacije **mjesto**, dotična se n-torka neće pojaviti u rezultatu. Jednako tako, ako za n-torku iz relacije **mjesto** ne postoji odgovarajuća n-torka u relaciji **stud**, ta n-torka iz relacije **mjesto** se neće pojaviti u rezultatu.

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
1	Horvat	10000	10000	Zagreb
3	Novak	51000	51000	Rijeka

Dio sintaksnog dijagrama za *FROM Clause* (vidjeti dijagram u poglavlju o spajanju relacija) koji sadrži poddijagram *ANSI Table Reference* može se koristiti za definiranje prirodnog spajanja, spajanja uz uvjet, a također i za definiranje vanjskog spajanja.



Sljedeći primjer pokazuje kako se poddijagram *ANSI Table Reference* koristi za definiranje "običnog" prirodnog spajanja. Upit iz prethodnog primjera ovaj je put napisan uz korištenje samo onog dijela dijagrama za *FROM Clause* koji sadrži poddijagram *ANSI Table Reference*:

```
SELECT stud.*, mjesto.*
FROM stud INNER JOIN mjesto
ON stud.pbr = mjesto.pbr
```

U *WHERE* dijelu *SELECT* naredbe se ne pišu uvjeti spajanja, nego samo eventualni uvjeti za selekciju (tzv, *post-join filter*). U sljedećem primjeru će se relacije **stud** i **mjesto** spojiti prema uvjetu napisanom u *ON Clause* dijelu naredbe, a konačni rezultat upita će se dobiti nakon primjene uvjeta selekcije iz *WHERE Clause* dijela naredbe:

```
SELECT stud.*, mjesto.*
FROM stud INNER JOIN mjesto
ON stud.pbr = mjesto.pbr
WHERE stud.prezStud LIKE 'Z%'
```

Prema sintaksnom dijagramu, dopušteno je umjesto *INNER JOIN* napisati samo *JOIN*. Sljedeći upit daje jednak rezultat kao i upit u prethodnom primjeru:

```
SELECT stud.*, mjesto.*
FROM stud JOIN mjesto
ON stud.pbr = mjesto.pbr
WHERE stud.prezStud LIKE 'Z%'
```

Za definiranje *SELECT* naredbi u kojima se koristi samo prirodno spajanje i spajanje uz uvjet i dalje se može koristiti dio sintaksnog dijagrama *FROM Clause* koji ne sadrži poddijagram *ANSI Table Reference*. Primjer s

takvom sintaksom naredbe nalazi se na samom početku ovog poglavlja. Međutim, kad se u naredbi koristi makar jedno vanjsko spajanje, **mora** se koristiti **samo** ona grana dijagrama *FROM Clause* koja sadrži poddijagram *ANSI Table Reference*.

Kod **vanjskog** spajanja relacije se tretiraju nesimetrično. Jedna je **dominantna**, te se njene n-torke pojavljuju u rezultatu bez obzira postoji li odgovarajuća n-torka u **podređenoj** relaciji. U sljedećem upitu **stud** je dominantna, a **mjesto** podređena relacija:

```
SELECT stud.*, mjesto.*
FROM stud LEFT OUTER JOIN mjesto
ON stud.pbr = mjesto.pbr
```

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
1	Horvat	10000	10000	Zagreb
2	Kolar	21000	NULL	NULL
3	Novak	51000	51000	Rijeka

Napomena: u primjerima se za prikaz NULL vrijednosti koristi oznaka NULL. U interaktivnim alatima za obavljanje SQL naredbi se ta vrijednost obično prikazuje kao prazan prostor.

Kod **vanjskog** spajanja relacija koja se spaja može biti:

- **dominantna** (*dominant, preserved*) - njene se n-torke (ukoliko zadovoljavaju eventualno postavljeni uvjet za selekciju u *WHERE* dijelu naredbe) pojavljuju u rezultatu bez obzira postoji li odgovarajuća n-torka u podređenoj relaciji.
- **podređena** (*subservient*). Njezine se n-torke pojavljuju jedino u slučaju kada zadovoljavaju uvjet spajanja (u dijelu naredbe kojim se definiraju uvjeti spajanja, tj. *ON Clause*)

Ukoliko za n-torku dominantne relacije ne postoji odgovarajuća n-torka podređene relacije (tj. n-torka koja zadovoljava uvjet spajanja definiran u *ON Clause* dijelu naredbe), u rezultatu se kao vrijednosti atributa podređene relacije pojavljuju NULL vrijednosti.

Uvjet spajanja iz *ON Clause* dijela naredbe ne utječe na to hoće li se n-torka dominantne relacije pojaviti u rezultatu ili ne - sve n-torke dominantne relacije koje zadovoljavaju uvjet selekcije iz *WHERE Clause* dijela naredbe se pojavljuju u rezultatu. Uvjet spajanja iz *ON Clause* dijela naredbe određuje jedino što će se dogoditi s n-torkama podređene relacije.

Prema sintaksnom dijagramu, moguće je umjesto LEFT OUTER JOIN napisati samo LEFT JOIN. Značenje je jednako. Tako se npr., prethodni primjer može napisati i ovako:

```
SELECT stud.*, mjesto.*
FROM stud LEFT JOIN mjesto
ON stud.pbr = mjesto.pbr
```

11.1 Razlika između uvjeta spajanja i uvjeta selekcije u upitima s vanjskim spajanjem

U uvjetima s vanjskim spajanjem naročitu pozornost treba pokloniti razlici između uvjeta spajanja i uvjeta za selekciju. Uvjeti koji su napisani u *ON Clause* dijelu naredbe su uvjeti spajanja, a uvjeti napisani u *WHERE Clause* dijelu naredbe su uvjeti selekcije (*post-join filter*).

```
SELECT stud.*, mjesto.*
FROM stud LEFT OUTER JOIN mjesto
ON stud.pbr = mjesto.pbr
```

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
1	Horvat	10000	10000	Zagreb
2	Kolar	21000	NULL	NULL
3	Novak	51000	51000	Rijeka

Ukoliko se doda uvjet za selekciju

```
SELECT stud.*, mjesto.*
FROM stud LEFT OUTER JOIN mjesto
ON stud.pbr = mjesto.pbr
WHERE stud.pbr <> 10000
```

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
2	Kolar	21000	NULL	NULL
3	Novak	51000	51000	Rijeka

Ukoliko se uvjet za selekciju iz prethodnog primjera napiše na mjestu uvjeta spajanja

```
SELECT stud.*, mjesto.*
FROM stud LEFT OUTER JOIN mjesto
ON stud.pbr = mjesto.pbr AND stud.pbr <> 10000
```

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>nazMjesto</u>
1	Horvat	10000	NULL	NULL
2	Kolar	21000	NULL	NULL
3	Novak	51000	51000	Rijeka

Primijetite da u prethodnom primjeru uvjetom **stud.pbr <> 10000** nije eliminirana prva n-torka. Prva n-torka nije eliminirana jer je uvjetima iz *ON Clause* dijela naredbe opisan **samo uvjet spajanja, koji kod vanjskog spajanja ne utječe na to hoće li se n-torka dominantne relacije pojaviti u rezultatu ili ne.**

Uvjet selekcije se slobodno može postaviti i na podređenu relaciju

```
SELECT stud.*, mjesto.*
FROM stud LEFT OUTER JOIN mjesto
ON stud.pbr = mjesto.pbr
WHERE mjesto.nazMjesto <> 'Rijeka'
```

STUD				
<u>mbr</u>	<u>prezime</u>	<u>pbr</u>	<u>pbr</u>	<u>naz mjesto</u>
1	Horvat	10000	10000	Zagreb

11.2 Vanjsko spajanje s tri ili više relacija

Tri relacije se mogu spojiti uz upotrebu vanjskog spajanja na tri različita načina:

- A) neka su zadane relacije R, S i T. n-torke iz relacije R vanjski se spajaju s rezultatom prirodnog spajanja relacija S i T

```
SELECT stud.mbrStud, ispit.rbrIspit, nastavnik.mbrNastavnik
FROM stud
      LEFT OUTER JOIN ispit
                        INNER JOIN nastavnik
                        ON ispit.mbrNastavnik = nastavnik.mbrNastavnik
      ON stud.mbrStud = ispit.mbrStud
```

STUD <u>mbrStud</u>	ISPIT <u>rbrIspit</u>	<u>mbrStud</u>	<u>mbrNastavnik</u>	NASTAVNIK <u>mbrNastavnik</u>
1	1	2	1	1
2	2	3	2	2
3	3	3	3	3
4	4	4	4	

U rezultatu će se pojaviti sve n-torke iz relacije **stud**, a uz njih jedna ili više n-torki iz **ispit** i **nastavnik** ukoliko postoji odgovarajući rezultat spajanja relacija **ispit** i **nastavnik**. Za n-torke dominantne relacije **stud**, za koje ne postoji odgovarajuća n-torka nastala spajanjem relacija **ispit** i **nastavnik**, n-torka rezultata se nadopunjava s NULL vrijednostima.

<u>mbrStud</u>	<u>rbrIspit</u>	<u>mbrNastavnik</u>
1	NULL	NULL
2	1	1
3	2	2
3	3	3
4	NULL	NULL

Primijetite da je rezultat sasvim različit od rezultata koji bi se dobio spajanjem tih relacija uz malo promijenjen redoslijed spajanja. Ovdje je prvo na relaciju **stud** vanjski spojena relacija **ispit**, a na dobiveni rezultat je prirodno spojena relacija **nastavnik**.

```
SELECT stud.mbrStud, ispit.rbrIspit, nastavnik.mbrNastavnik
FROM stud
      LEFT OUTER JOIN ispit
      ON stud.mbrStud = ispit.mbrStud
      INNER JOIN nastavnik
      ON ispit.mbrNastavnik = nastavnik.mbrNastavnik
```

<u>mbrStud</u>	<u>rbrIspit</u>	<u>mbrNastavnik</u>
2	1	1
3	2	2
3	3	3

- B) neka su zadane relacije R, S i T. Na dominantnu relaciju R vanjski se spaja relacija S, te se nezavisno o tom spajanju, na dominantnu relaciju R vanjski spaja i relacija T

```
SELECT stud.mbrStud, ispit.rbrIspit, mjesto.pbr
FROM stud
      LEFT OUTER JOIN ispit ON stud.mbrStud = ispit.mbrStud
      LEFT OUTER JOIN mjesto ON stud.pbr = mjesto.pbr
```

STUD	
<u>mbrStud</u>	<u>pbr</u>
1	10000
2	20000
3	31000
4	21000
5	22000

ISPIT	
<u>rbrIspit</u>	<u>mbrStud</u>
1	3
2	4
3	4
4	5
5	5

MJESTO	
<u>pbr</u>	
20000	
51000	
21000	

U rezultatu će se pojaviti sve n-torke iz relacije **stud**, a uz svaku takvu n-torku pojavit će se jedna ili više n-torki iz relacije **ispit** (ukoliko u relaciji **ispit** postoje odgovarajuće n-torke), te jedna ili više n-torki iz relacije **mjesto** (ukoliko postoje odgovarajuće n-torka u relaciji **mjesto**).

<u>mbrStud</u>	<u>rbrIspit</u>	<u>pbr</u>
1	NULL	NULL
2	NULL	20000
3	1	NULL
4	2	21000
4	3	21000
5	4	NULL
5	5	NULL

C) neka su zadane relacije R, S i T. Na dominantnu relaciju R vanjski se spaja relacija S, te se na dobiveni rezultat vanjski spaja i relacija T

```
SELECT mjesto.pbr, stud.mbrStud, ispit.rbrIspit
FROM mjesto
  LEFT OUTER JOIN stud
    ON mjesto.pbr = stud.pbr
  LEFT OUTER JOIN ispit
    ON stud.mbrStud = ispit.mbrStud
```

MJESTO	
<u>pbr</u>	
10000	
20000	
51000	
22000	

STUD	
<u>mbrStud</u>	<u>pbr</u>
1	32000
2	20000
3	20000
4	20000
5	51000
6	22000

ISPIT	
<u>rbrIspit</u>	<u>mbrStud</u>
1	1
2	5
3	5
4	4

U rezultatu će se pojaviti sve n-torke iz relacije **mjesto**, a uz njih jedna ili više n-torke iz **stud** (ukoliko postoje odgovarajuće n-torka u **stud**). Pri tome će se uz svaku n-torku iz relacije **stud** pojaviti i jedna ili više n-torki iz relacije **ispit** (ukoliko postoje odgovarajuće n-torka u relaciji **ispit**).

<u>pbr</u>	<u>mbrStud</u>	<u>rbrIspit</u>
10000	NULL	NULL
20000	2	NULL
20000	3	NULL
20000	4	4
51000	5	2
51000	5	3
22000	6	NULL

Sljedeći primjeri ilustriraju prikazane oblike, A, B, C, vanjskog spajanja. U primjerima se koristi vanjsko spajanje uz uvjet.

ZRAKOPLOV		PILOT		LET	
<u>tip</u>	<u>dolet</u>	<u>prezime</u>	<u>tip</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	JOHNSON	B747	CA-825	7200
DC9	3000	SMITH	AIRBUS	A-224	3300
B727	4500	WALKER	F16	CA-878	4200
AIRBUS	1800				

"obično" spajanje

```
SELECT * FROM zrakoplov, pilot, let
WHERE zrakoplov.tip = pilot.tip
AND zrakoplov.dolet >= let.udaljenost
```

<u>tip</u>	<u>dolet</u>	<u>prezime</u>	<u>tip</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	JOHNSON	B747	A-224	3300
B747	6000	JOHNSON	B747	CA-878	4200

"obično" vanjsko spajanje relacije s rezultatom prirodnog spajanja

```
SELECT *
FROM zrakoplov
INNER JOIN pilot
ON zrakoplov.tip = pilot.tip
LEFT OUTER JOIN let
ON zrakoplov.dolet >= let.udaljenost
```

<u>tip</u>	<u>dolet</u>	<u>prezime</u>	<u>tip</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	JOHNSON	B747	A-224	3300
B747	6000	JOHNSON	B747	CA-878	4200
AIRBUS	1800	SMITH	AIRBUS	NULL	NULL

slučaj A

```
SELECT *
FROM pilot
LEFT OUTER JOIN zrakoplov
INNER JOIN let
ON zrakoplov.dolet >= let.udaljenost
ON pilot.tip = zrakoplov.tip
```

<u>prezime</u>	<u>tip</u>	<u>tip</u>	<u>dolet</u>	<u>brLeta</u>	<u>udaljenost</u>
JOHNSON	B747	B747	6000	A-224	3300
JOHNSON	B747	B747	6000	CA-878	4200
SMITH	AIRBUS	NULL	NULL	NULL	NULL
WALKER	F16	NULL	NULL	NULL	NULL

slučaj B

```
SELECT *
FROM zrakoplov
LEFT OUTER JOIN pilot
ON zrakoplov.tip = pilot.tip
LEFT OUTER JOIN let
ON zrakoplov.dolet >= let.udaljenost
```

<u>tip</u>	<u>dolet</u>	<u>prezime</u>	<u>tip</u>	<u>brLeta</u>	<u>udaljenost</u>
B747	6000	JOHNSON	B747	A-224	3300
B747	6000	JOHNSON	B747	CA-878	4200
DC9	3000	NULL	NULL	NULL	NULL
B727	4500	NULL	NULL	A-224	3300
B727	4500	NULL	NULL	CA-878	4200
AIRBUS	1800	SMITH	AIRBUS	NULL	NULL

slučaj C

```
SELECT *  
  FROM pilot  
    LEFT OUTER JOIN zrakoplov  
      LEFT OUTER JOIN let  
        ON zrakoplov.dolet >= let.udaljenost  
    ON pilot.tip = zrakoplov.tip
```

<u>prezime</u>	<u>tip</u>	<u>tip</u>	<u>dolet</u>	<u>brLeta</u>	<u>udaljenost</u>
JOHNSON	B747	B747	6000	A-224	3300
JOHNSON	B747	B747	6000	CA-878	4200
SMITH	AIRBUS	AIRBUS	1800	NULL	NULL
WALKER	F16	NULL	NULL	NULL	NULL

Dodatak A: Programski alat Relational Object Manager

IBM INFORMIX ROM je programski alat sačinjen od nekoliko modula, koji u grafičkom okruženju omogućavaju kreiranje i rad s objektima baze podataka. ROM uključuje:

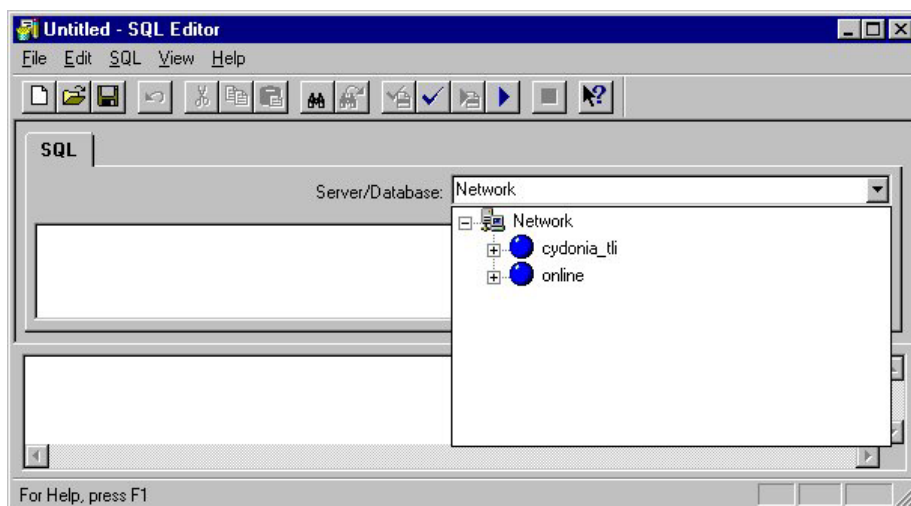
Database Explorer	Glavni modul iz kojeg se mogu pokrenuti i ostali moduli ROM alata. Omogućava pregledavanje, kreiranje te izmjenu objekata u SQL-u.
Table Editor	Kreiranje i izmjena relacijskih shema u bazi podataka.
SQL Editor	Editiranje i obavljanje SQL naredbi i pregled rezultata.
Stored Procedure Editor	Kreiranje i izmjena pohranjenih procedura.
Trigger Editor	Kreiranje i izmjena okidača.
View Editor	Kreiranje i izmjena virtuelnih tablica (pogleda).

U laboratoriju će se koristiti modul **SQL Editor**.

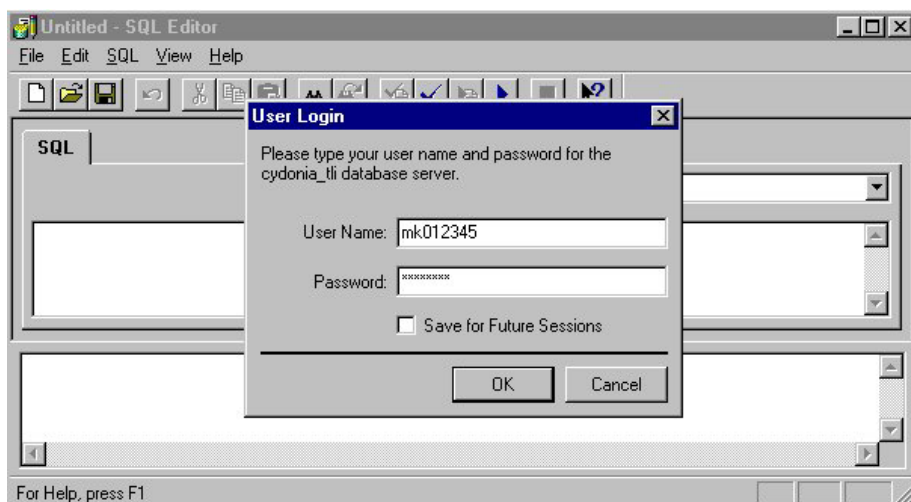
A.1. Pokretanje SQL Editora u programskom alatu ROM

→Start→Programs→Informix Administration Tools→SQL Editor 2.10

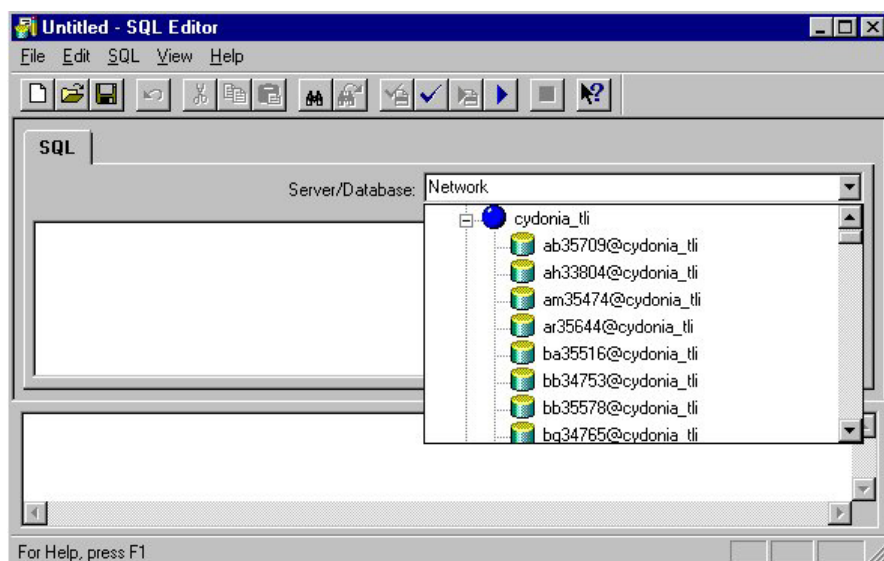
Iz popisa poslužitelja baza podataka koji su tog trenutka poznati računalu klijentu, treba odabrati jedan. Na slici je prikazan popis na kojem se nalaze dva poslužitelja: cydonia_tli i online.



Nakon odabira poslužitelja pojavit će se prozor preko kojeg se obavlja postupak prijave korisnika. Potrebno je upisati korisničko ime i lozinku (*login* i *password*). **Nemojte** označiti opciju "Save for Future Sessions".



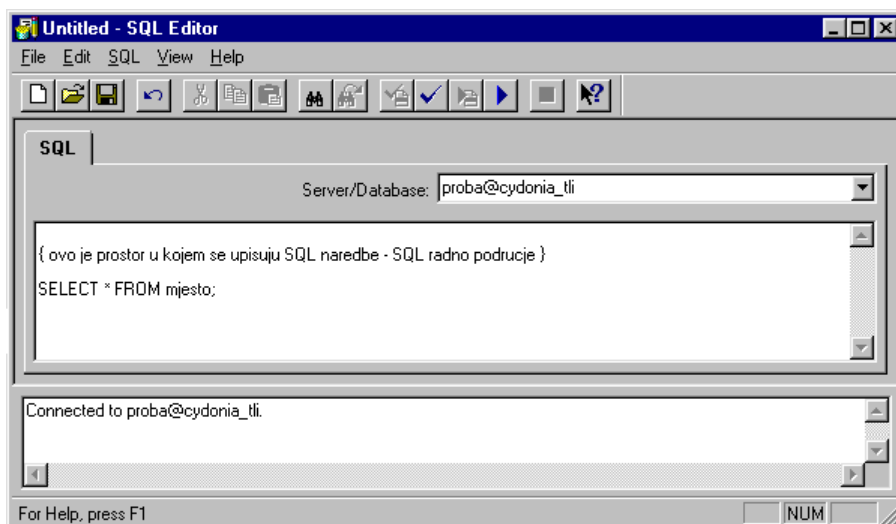
Ukoliko je postupak prijave prošao bez problema, uz ime poslužitelja će se moći prikazati popis baza podataka koje postoje na tom poslužitelju. Ukoliko Vaša baza podataka već postoji, možete je odabrati na popisu ili svojoj bazi podataka možete pristupiti pomoću SQL naredbe DATABASE *imeBazePodataka*. Način na koji se SQL naredbe obavljaju opisan je u poglavlju A.2.







Ukoliko Vaša baza podataka ne postoji na poslužitelju, možete je kreirati obavljanjem SQL naredbe CREATE DATABASE *imeBazePodataka*. Način na koji se SQL naredbe obavljaju opisan je u poglavlju A.2.

A.2. Editiranje i obavljanje SQL naredbi

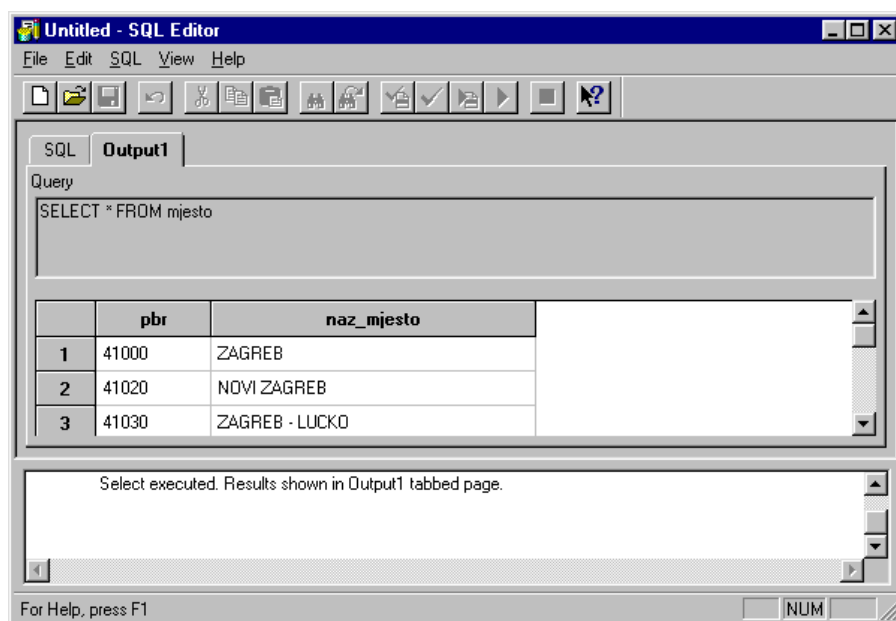
Editiranje i obavljanje SQL naredbi obavlja se pomoću SQL Editor-a



Važnije opcije:

-  provjera ispravnosti sintakse označenih SQL naredbi
-  provjera ispravnosti sintakse svih SQL naredbi upisanih u SQL radnom području (osim onih koje se nalaze unutar komentara)
-  izvođenje označenih SQL naredbi (osim onih koje se nalaze unutar komentara)
-  izvođenje svih SQL naredbi iz radnog područja (osim onih koje se nalaze unutar komentara)

Nakon izvođenja SQL naredbe, informacija o obavljenima naredbama pojavljuje se u statusnom prozoru. Rezultat SELECT naredbe pojavljuje se u posebnom prozoru (npr. prozor *Output1*):



Dodatak B: SQL naredbe za kreiranje probne baze podataka

```
CREATE TABLE zupanija
(  sifZupanija    SMALLINT    NOT NULL
,  nazZupanija    NCHAR(40)   NOT NULL
);

CREATE DISTINCT INDEX zupanijaUnique ON zupanija (sifZupanija);

CREATE TABLE mjesto
(  pbr           INTEGER      NOT NULL
,  nazMjesto     NCHAR(40)    NOT NULL
,  sifZupanija    SMALLINT
);

CREATE DISTINCT INDEX mjestoPbr ON mjesto (pbr);
CREATE INDEX mjestoUnique ON mjesto (nazMjesto);

CREATE TABLE stud
(  mbrStud       INTEGER      NOT NULL
,  imeStud       NCHAR(25)    NOT NULL
,  prezStud      NCHAR(25)    NOT NULL
,  pbrRod        INTEGER
,  pbrStan       INTEGER
,  datRodStud    DATE
,  jmbgStud      CHAR(13)
);

CREATE DISTINCT INDEX studUnique ON stud (mbrStud);

CREATE TABLE orgjed
(  sifOrgjed      INTEGER      NOT NULL
,  nazOrgjed      NCHAR(60)    NOT NULL
,  sifNadorgjed   INTEGER
);

CREATE DISTINCT INDEX orgjedUnique ON orgjed (sifOrgjed);

CREATE TABLE nastavnik
(  sifNastavnik   INTEGER      NOT NULL
,  imeNastavnik   NCHAR(25)    NOT NULL
,  prezNastavnik  NCHAR(25)    NOT NULL
,  pbrStan       INTEGER
,  sifOrgjed      INTEGER      NOT NULL
,  koef          DECIMAL(3,2)  NOT NULL
);

CREATE DISTINCT INDEX nastavnikUnique ON nastavnik (sifNastavnik);

CREATE TABLE pred
(  sifPred        INTEGER      NOT NULL
,  kratPred       CHAR(8)
,  nazPred        NCHAR(60)    NOT NULL
,  sifOrgjed      INTEGER
,  upisanoStud    INTEGER
,  brojSatiTjedno INTEGER
);

CREATE DISTINCT INDEX predUnique ON pred (sifPred);

CREATE TABLE ispit
(  mbrStud       INTEGER      NOT NULL
,  sifPred        INTEGER      NOT NULL
,  sifNastavnik   INTEGER      NOT NULL
,  datIspit      DATE          NOT NULL
,  ocjena        SMALLINT     DEFAULT 1 NOT NULL
);
```



```

CREATE DISTINCT INDEX ispitUnique ON ispit (mbrStud, sifPred, datIspit);

CREATE TABLE dvorana
( oznDvorana CHAR(5) NOT NULL
, kapacitet INTEGER DEFAULT 40
);

CREATE DISTINCT INDEX dvoranaUnique ON dvorana (oznDvorana);

CREATE TABLE rezervacija
( oznDvorana CHAR(5) NOT NULL
, oznVrstaDan CHAR(2) NOT NULL
, sat SMALLINT NOT NULL
, sifPred INTEGER NOT NULL
);

CREATE DISTINCT INDEX rezervacijaUnique
ON rezervacija (oznDvorana, oznVrstaDan, sat);

```

Dodatak C: E-R dijagram probne baze podataka

