UVOD U SQL

Ivica Masar

pso@vip.hr

Zagreb, rujan 2002.

SADRŽAJ

1. UVOD

Na samom početku bi htio objasniti kako sam se odlučio za ovu temu te se zahvaliti onim koji su mi omogućili da napravim ovaj rad. Prvi koraci su se dogodili na poslu, a budući da sam se zaposlio u informatičkoj tvrtci morao sam naučiti dosta različitih stručnih stvari. Najčešće je to bilo uz literaturu na engleskom jeziku, ali polako sam uviđao da dosta toga ima i na hrvatskom jeziku kao tekstualne poduke na internetu. Tu je bilo programiranja na svim programskim jezicima, kriptografija, hacking, cracking, security i slično. Informacije o SQL-u sam našao na stranicama Fakulteta elektrotehnike i računarstva u Zagrebu gdje se obrađivala Informix baza podataka. U knjigama "SQL i relacijski model podataka " izdavača "ZNAK" autora Ratka Vujnovića i "Naučite SQL za 21 dan" koja je prevedena u Jugoslaviji izdavača "Kompjuter Biblioteke" našao sam također dosta korisnih informacija. Tako sam došao do ideje da bi mogao napraviti uvod u SQL i dati ga zainteresiranim na korištenje.

Ovu priliku bi iskoristio i da se zahvalim svom mentoru profesoru Hrvoju Gold koji mi je omogućio da to napravim također i svojim kolegama na poslu koji su mi pomagali u mojim počecima. Trebalo bi napomenuti da će se po potrebi koristiti stručni izrazi u engleskom jeziku, ali smatram da je to normalno kad se radi o temama iz područja informatike.

2. STRUKTURIRANI JEZIK UPITA

SQL kratica zapravo znači " Structured Query Language" što bi se moglo prevesti kao strukturirani jezik za upite. Izgovara se S-Q-L (pojedinačno se izgovori svako slovo ili sequel . Njegova povijest počinje 1970 godine kad je razvijen u IBM Research Laboratory u San Jose-u, California. Do 80-tih godina nije bio komercijalno razvijen zbog slabih karakteristika tadašnjih računala, ali 1981. IBM je predstavio prvi komercijalnu SQL proizvod SQL/DS, iza njega su se pojavili sustavi baza podataka Oracle i Reational Technology. Do 1989 postojalo je oko sedamdesetak različitih verzija SQL sustava. ANSI (American National Standards Institute) i ISO (International Standards Organization) su objavili prve inačice standarda 1986-e odnosno 1987-e. Zatim su skupa prihvatili proširenje standarda 1989 pod nazivom SQL-89 gdje je najvažnije proširenje bilo uvođenje referencijskog integriteta. Onda su 1992 godine objavili SQL-2 ili SQL-92 gdje imamo proširenje standarda u pisanoj formi (broj stranica) više od 4 puta nego u ranijoj verziji. Na kraju objavljen je SQL-99 ili SQL-3 s novim mogućnostima. U pogledu fizičkog zapisa podataka trebalo bi napomenuti da je on različit od logičke strukture ili onog načina na koji mi to vidimo u svojim glavama. Logička organizacija podataka predstavlja organizaciju sa stanovišta korisnika baze podataka te se odnosi na vrste podataka i njihove međusobne logičke veze. Sam SQL kao što ćemo vidjeti opisuje što želimo dobiti kao rezultat, a ne kako doći do toga što ga svrstava u neproceduralne jezike za razliku od npr. C programskog jezika.

SQL je razvijen za rad sa relacijskim bazama podataka za koje dr. Codd 1970 godine iznosi 12 Codd-ovih pravila objavljenih u članku " A Relational Model of Data for Large Shared Data Banks" (Relacijski model podataka za velike baze podataka koje koristi više korisnika). Pravila glase :

0) Relacijski DBMS (Database Management System) – sustav za upravljanje bazama podataka mora biti u mogućnosti da upravlja u potpunosti bazama podataka kroz svoje relacijske mogućnosti.

1) Pravila o informacijama – sve informacije u relacijskoj bazi podataka (uključujući imena tablica i stupaca) predstavljaju se kao vrijednosti u tablici

2) Osiguravanje pristupa – svakoj se vrijednosti u relacijskoj bazi podataka može pristupiti upotrebom kombinacije imena tablice, vrijednosti primarnog ključa i imena stupaca.

3) Sustavna podrška nedefiniranim vrijednostima – DBMS osigurava sustavnu podršku u radu sa nedefiniranim veličinama (nepoznati ili neprimjenjivi tipovi podataka), koji se razlikuju od definiranih vrijednosti i neovisni su

4) Aktivan, uvijek dostupan relacijski katalog – opis baze podataka i njenog sadržaja je predstavljen na logičkom nivou u vidu tabela i može se pretraživati pomoću jezika baze podataka.

5) Razumljiv podjezik podataka – bar jedan podržani jezik mora imati dobro definiranu sintaksu i biti razumljiv. Mora se podržati definicija podataka, upravljanje podacima, pravila integriteta, autorizacija i transakcije.

6) Pravilo za ažuriranje pogleda – svi pogledi koji se teoretski mogu ažurirati, ažuriraju se kroz sustav.

7) Unošenje, ažuriranje i uklanjanje podataka na nivou skupova – DBMS za dobivanje podataka na nivou skupova i za unošenje, ispravak i uklanjanje podataka.

8) Fizička neovisnost podataka – mijenjanje fizičkog zapisa strukture ili metode pristupa ne utječe na aplikacije ili programe

9) Logička neovisnost podataka – koliko je god moguće, promjena strukture tablica ne utječe na aplikacije ili programe

10) Neovisnost integriteta – jezik baze podataka mora osigurati način za definiranje pravila integriteta. Ona moraju biti sačuvana u katalogu, koji je uvijek dostupan i ne može se ignorirati.

11) Neovisnost od distribucije – prva ili ponovna distribucija podataka ne utječe na zahtjeve aplikacije

12) Zaštita podataka – ne smije postojati mogućnost zaobilaženja pravila integriteta definiranih jezikom baze podataka upotrebom jezika koji rade na niskom nivou.

Znači SQL omogućava da tvorimo i promijenimo strukturu baze podataka, dodamo prava korisniku za pristup bazama podataka ili tablicama, da tražimo informacije od baze podataka i da mijenjamo sadržaj baze podataka, zapravo su tu dvije skupine funkcija: DDL (Data Definition Language) funkcije za definiciju podatka čiji je tipičan primjer naredba *CREATE TABLE imeTablice ();* DML (Data Manipulation Language) funkcije za upravljanje podacima gdje kao primjer možemo navesti osnovnu SQL naredbu *SELECT * FROM imeTablice*.

Sam pristup podacima odvija se prema modelu klijent / poslužitelj. To je po Bernardu H. Boar autoru knjige "Implementing Client/server Computing" [(1)] definirano kao :

> " Model rada u kojem je jedna aplikacija podijeljena između više procesa koji komuniciraju (transparentno prema korisničkom kraju) da bi završili procesiranje kao jedan jedinstven zadatak. klijent/poslužitelj model vezuje procese da bi se dobila slika jedinstvenog sustava. Djeljivi resursi su pozicionirani klijenti koji imaju zahtjeve i mogu pristupiti ovlaštenim servisima. Arhitektura je beskonačno rekurzivna; pa poslužitelji mogu postati klijenti i zahtijevati usluge od drugih poslužitelj u mreži, itd."

Procesi komuniciraju pomoću cjevovoda (pipes) ili zajedničke memorije (shared memory) ako se nalaze na istom računalu, te uz pomoć nekog mrežnog protokola (npr. TCP/IP), ako nisu na istom računalu, što znači da nema više fizičkog ograničenja jer se možemo spojiti telefonskom linijom čak ne moramo imati isti operativni sustav već je samo važno da podržava isti mrežni protokol, a zahvaljujući centraliziranom pohranjivanju podataka poboljšan je integritet podataka. Aplikacije su često rađene u drugim okruženjima nego što nam ga je dostavio

proizvođač sustava te se spajanje s poslužiteljem obavlja preko standardnih programskih sučelja kao što su ODBC (Open Database Connectivity) i JDBC (Java Database Connectivity). Možemo napomenuti da postoji i jedna podvrsta JDBC driver-a , a to je JDBC-ODBC most. Samo instaliranje JDBC-a sastoji se od kopiranja odgovarajućih datoteka na tvrdi disk (to su najčešće *.jar datoteke). Dalje slijede primjeri koda za JDBC

učitavanje driver-a:

```
Class.forName("jdbc.DriverXYZ");
```

uspostavljanje veze:

```
Connection con = DriverManager.getConnection(url,
"myLogin", "myPassword");
```

3. PRAVILA PISANJA KODA

SQL ne pravi razliku između malih i velikih slova, što znači da su slijedeće dvije naredbe jednake:

select prezime from osoba where ime = 'Pero'

ili

SELECT prezime FROM osoba WHERE ime = 'Pero'

Radi lakšeg čitanja koda, a tim načinom će biti i sve dalje napisano preporuča se da ključne riječi (naredbe) budu napisane velikim slovima, svi ostali elementi malim slovima. U nekim bazama niz znakova (string) mora biti napisan kao što je u bazi. Znači u gornjim naredbama nije isto ako piše 'Pero' ili 'PERO', ali isto ne mora vrijediti za Microsoft SQL.

Komentari su tekst koji upišemo kao podsjetnik, a koji se neće izvršiti. Imamo ih dvije vrste:

za samo jedan red

- - ovo je komentar

tj. oznaka za komentar je - - ,a iza slijedi tekst komentara ili

komentar kroz više redova

/

ovo je

također komentar, ali se

proteže kroz više

*redova */*

za Informix umjesto /* */ imamo vitičaste zagrade {}

Imena objekata u SQL-u (imena tablica, stupaca, pogleda) prave se tako da prvi znak mora biti a - z, A – Z ili podvučeno (underscore) _ , te u nekim slučajevima @ i #. U MS SQL imena mogu biti preko 100 znakova. Znakovi iza prvog mogu biti pisani u Unicode kodu, decimalni brojevi ili znakovi @, $, _, #, ali ne mogu biti ključne riječi. Ključne riječi za SAP DB možemo vidjeti u dodatku ključne riječi. Razmak i ostali specijalni znakovi se mogu koristiti, ali nije preporučljivo npr.

*SELECT * FROM "Nova adresa";*
*SELECT * FROM [Nova adresa]*

Takve nizove treba označiti sa " " ili [ ].

4. BAZA PODATAKA

Baza podataka je objekt koji sadrži tablice i druge objekte radi pospremanja i obrade podataka. Za kreiranje baze najčešće se koristi naredba :
Primjer 1:

CREATE DATABASE imeBaze;

1992 SQL ANSI standard (SQL-92) ne uključuje CREATE DATABASE naredbu, umjesto nje on koristi CREATE SCHEMA naredbu za opis dijela baze koju određeni korisnik koristi. Obično se baza podataka sastoji od više schema. Ipak većina komercijalnih inačica SQL-a podražava naredbu CREATE DATABASE, iza imena baze podataka mogu slijediti još dodaci kao što su NAME (logičko ime baze podataka), FILENAME (ime i staza datoteke baze podataka pod kojim ona postoji na tvrdom disku), SIZE(početna veličina datoteke za podatke u MB ili kB), MAXSIZE (najveća veličina koju baza podataka smije automatski uzeti, zadaje se u MB ili kB, a može biti i UNLIMITED), FILEGROWTH (vrijednost koraka širenja (rasta) datoteke, a zadaje se u MB, kB ili postotcima %), LOG ON(zadaje se veličina datoteke dnevnika transakcija i lokaciju, ako nije zadana biti će 25% veličine svih datoteka za podatke, njeno ime će sustav sam generirati i biti će u istom direktoriju kao i datoteke podataka), FOR LOAD (zbog kompatibilnosti), FOR ATTACH (za pridruživanje datoteka baze podataka koje su bile formirane na drugom poslužitelju). Ove dodatne opcije nisu standardizirane te bi trebalo provjeriti u dokumentaciji koje se opcije mogu primijeniti u pojedinom sustavu upravljanja bazama podataka. Samo kreiranje baze ne uključuje nužno da je sada koristite pa

se moramo spojiti na tu bazu podataka sa naredbama USE, DATABASE ili CONNECT ili će se najvjerojatnije moći spojiti uz pomoć nekog grafičkog sučelja npr. u Informixu pri radu sa SQL Editorom ili u MS SQL-u u radu sa SQL Server Query Analyzerom imamo okvir s padajućim popisom (combo box) u kojem biramo uz pomoć miša na koju se bazu želimo spojiti, dok se to isto ručno upisuje u SAP-ovom SQL Studiu.
Primjer 2:

USE mobitel

The command(s) completed successfully.

Pri dizajnu baze podataka trebali bi imati na umu sigurnost, prostor na disku, brzinu pretraživanja, dobivanja rezultata, ažuriranja (nekad je dobro koristiti umjesto naredbe UPDATE kombinaciju DELETE i INSERT), brzinu spajanja više tablica radi dobivanja podataka i RDBMS-ove mogućnosti privremenih tablica. Idući primjer pokazuje kreiranje (*CREATE*) i brisanje (*DROP*) novonastale baze podataka.
Primjer 3:

CREATE DATABASE proba;
DROP DATABASE proba;

Nešto drukčije radi se kreiranje baze za SAP, ali najvažnijim smatram da se postavi UNICODE baza zbog mogućnosti spremanja hrvatskih znakova u bazu. Primjer datoteke za kreiranje baze imamo u dodatku pod KREIRANJE BAZE ZA SAP.

5. TIPOVI PODATAKA

Pri kreiranju tablica određujemo nazive stupaca te tip podatka koji će biti spremljen. To bi mogli usporediti sa inicijalizacijom varijable u programiranju. Tipovi podataka su:

Cjelobrojni:

- **bit** podatak koji je 1 ili 0
- **int** (**integer**) koji iznosi od -2^{31} (-2,147,483,648) do $2^{31}-1$ 2,147,483,647) pohranjen u 4 byte-a
- **smallint** cijeli broj pohranjen u 2 byte-a; 2^{15} (-32,768) do 2^{15} - 1 (32,767)
- **tinyint** podatak od 0 - 255

Decimalni:

- **decimal** ili **numeric** $-(10)^{38}-1$ do $10^{38}-1$.

Primjer je decimal(15, 3);. Prva znamenka označava ukupan broj znamenki, a druga broj decimalnih mjesta iza decimalne točke.

Novac:

- **money** tip podatka je isti kao i decimal . Razlika je u ispisu.

-2^{63} (-922,337,203,685,477.5808) do 2^{63} - 1 (+922,337,203,685,477.5807)

- **smallmoney** 214,748.3648 do +214,748.3647

kod novčanog tipa podatka podaci se čuvaju sa četiri decimalna mjesta

Pomični zarez:

- **float** Floating -1.79E + 308 do 1.79E + 308.
- **real** -3.40E + 38 do 3.40E + 38.

Datumi:

- **datetime** 1.Siječanj, 1753, do 31.Prosinca, 9999 uz točnost od 3.33 milisekunde
- **smalldatetime** 1.Siječanj, 1900, do 6.Lipnja, 2079 uz točnost od minute

Nizovi znakova:

- **char (character)** znakovni niz npr. char (9) u bazi će podatak zauzimati 9 znakova bez obzira na unošenu duljinu što znači da može doći do kraćenja ili nadopune .Maksimalno 8000 znakova.
- **nchar** (national char) Spremaju se znakovi koji spadaju u Unicode. Maksimalne dužine 4000 znakova.
- **text** sprema tekstualne podatke . Može sadržavati 2,147,483,647 znakova.
- **varchar** promjenjiva dužina (u bazu se sprema trenutna dužina podatka) ne Unicode znakova. Maksimalne dužine 8000 znakova.
- **nvarchar** (national char varying) promjenjiva dužina Unicode znakova. Može sadržavati 4000 znakova.

Binarni

- **binary** binarni podatak maksimalne duljine 8000 bajtova
- **varbinary** binarni podatak promjenjive dužine. Maksimalne dužine 8000 bajtova.
- **image** binarni podatak promjenjive dužine , maksimalne dužine 2,147,483,647 bajtova.

Usporedba tipova podataka kod 6 baza podataka prikazana je u Tablici 1.

TIP	ASA	ASE	MS SQL	ORACLE	INFOR.	SAP
znak	*char*	*char*	*char*	*char*	*char*	*char*
znak promj. dužine	*varchar*	*varchar*	*varchar*	*varchar2*	*varchar*	*varchar, long (varchar)*
datum, vrijeme	*datetime, smalldate-time, timestamp*	*datetime ,small-date-time*	*datetime, small-date-time*	*date, timestamp*	*datetime*	*time-stamp*
datum	*date*	-	-	-	*date*	*date*
vrijeme	*time*	-	-	*time*	*interval*	*time*
cjelobroj.	*int, smallint, tinyint*	*int, smallint, tinyint*	*int, smallint, tinyint*	*int, smallint, number*	*int, smallint*	*int, smallint*
decimalni	*dec, numeric*	*dec, numeric*	*dec, numeric*	*dec, numeric, number*	*dec, numeric*	*dec, fixed*
pomični zarez	*float, real, double precision*	*float, real, double prec-ision*	*float, real, double precision*	*float, real, double precision number*	*float, smallfloat real, double precision dec*	*float*
novac	*money, small-money*	*money, small-money*	*money, small-money*	-	*money*	-

Tablica 1. Usporedba tipova podataka kod 6 baza podataka

6. KREIRANJE TABLICA

Tablice predstavljaju dvodimenzionalne matrice čiji redovi predstavljaju naziv i svojstva objekata pohranjenih u tablicu, a stupci svojstva objekata izražena odgovarajućim tipom podatka. Uz pomoć jedne n-torke opisali smo jedan objekt. Zapravo bi to mogli i opisati kao tablice u Wordu kroz koje će dalje i biti predstavljeni primjeri eksploatacije mobitela. Primjer :

maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Perić	Gajeva 3	Zagreb
0302982383818	Ivan	Ivić		Split
0305972383915	Marko	Marić	Lavova 67	Zadar

Vidimo da je to objekt baze podataka u kojem se čuvaju podaci. Naredba koju ćemo koristiti za kreiranje tablica glasi *CREATE TABLE imeTablice* te u skladu sa bazom imamo i naredbu *DROP DATABASE imeTablice* , koja je suprotna prethodnoj naredbi, pa s njom bespovratno uništavamo podatke, strukture tablice ili privremene tablice. Idući primjeri biti će kreiranja tablica kroz koje ćemo objasniti SQL.

Primjer baze će biti eksploatacija mobitela i knjiženje vremena posudbe i vraćanja. Ono će biti provedeno kroz tri tablice: mobitel, osoba i posudi.
Primjer 4:

```
CREATE TABLE osoba
(
   maticni NVARCHAR(15),
   ime NVARCHAR(15) NOT NULL,
```

```
    prezime NVARCHAR(15) NOT NULL,
    ulica NVARCHAR(25),
    mjesto NVARCHAR(15) DEFAULT 'Zagreb'
    PRIMARY KEY (maticni)
  );

CREATE TABLE mobitel
(
    sifra NVARCHAR(15),
    proizvodi NVARCHAR(15) NOT NULL,
   model NVARCHAR(15) NOT NULL,
   tezina INT,
   visina INT,
   sirina INT,
   debljina INT,
   UNIQUE (proizvodi, model)
   PRIMARY KEY (sifra)
  );
CREATE TABLE posudi
(
   broj INT IDENTITY (1, 1) NOT NULL,
  osoba  NVARCHAR(15),
  mobitel NVARCHAR(15),
  uzeo DATETIME,
  vratio DATETIME,
  -- napomena NVARCHAR(25),
   FOREIGN KEY (osoba) REFERENCES osoba,
  -- FOREIGN KEY (mobitel) REFERENCES mobitel
 );
```

Kad ove naredbe izvršimo u SQL Editoru kao rezultat ćemo dobiti tri tablice i kao što je prije napomenuto u slučaju da ih želimo izbrisati služimo se sa naredbom *DROP* npr. *DROP TABLE posudi;*
Kroz analizu navoda strukture objekata u bazi možemo vidjeti da u zadnjoj tablici posudi imamo dva komentara. Oni su stavljeni da se vidi kako izgleda komentar i kako izgleda puna tablica. Znači u ovom kodu taj dio iako ispravan neće se izvoditi pa ćemo pokazati i kako naknadno izmijeniti strukturu tablice (nakon što je kreirana). Naknadna izmjena se najčešće radi jer se kod prvobitne izgradnje tablice nije uzelo u obzir sve što bi trebalo ili je došlo da nekakvih zahtjeva za promjenama aplikacije i baze podataka. Izmjena se vrši naredbom *ALTER TABLE.* Sad ćemo dodati jedan stupac tipa nvarchar i jedan strani ključ (*FOREIGN KEY*) koji će biti objašnjen malo kasnije.
Primjer 5:

```
ALTER TABLE  posudi ADD FOREIGN KEY (mobitel)
        REFERENCES mobitel;
ALTER TABLE posudi ADD napomena NVARCHAR(25);
```

U tablicama osoba i mobitel susrećemo *NOT NULL.* Iz samog naziva vidimo da su to stupci u kojim mora biti nešto upisano, jasno je da upotreba ove naredbe ovisi o dizajnu naše tablice odnosno o strukturi tablice koju smo mi zamislili tako da će se dalje zapravo objašnjavati dizajn tablice i zašto su nekim stupcima određene dodatne karakteristike. Za stupce ime i prezime nužno je da budu upisani neki podaci bez čega inače ta tablica ne bi imala smisla, adresu sam odredio da je poželjna, ali ne i nužna kao i mjesto koje će u slučaju da ništa ne upišemo poprimiti vrijednost Zagreb ...*DEFAULT 'Zagreb'* i još nam ostaje stupac maticni koja je određena kao osnovni ključ (*PRIMARY KEY (maticni)*). Osnovni

ključ bi trebao zadovoljiti kriterije da bude jedinstven u čitavoj tablici (nemamo dva ista) i da se ostali podaci iz tog reda ne ponavljaju u nekom drugom redu jer bi došlo do ponavljanja podataka što treba izbjeći.

maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Perić	Gajeva 3	Zagreb
0302982383818	Ivan	Ivić		Split
0305972383915	Marko	Marić	Lavova 67	Zadar

Vidimo da se broj u stupcu maticni ne ponavlja znači da je prvi uvjet zadovoljen, a on jednoznačno određuje osobu tj. nema osobe koja ima dva matična broja pa smo zadovoljili i drugi uvjet. Pravilno postavljen osnovni ključ odmah zadovoljava uvjete, matični broj je još bolji jer mu je zadnja znamenka kontrolni broj po modulu 11 pa programski možemo provjeriti i da li je pravilno upisan. Ipak gledajući sa stanovišta jednostavnosti i privatnosti možemo pitati zašto ne ime i prezime? Prvo bi trebalo napomenuti da se osnovni ključ može protezati i kroz više stupaca i onda vrijede isti uvjeti, ali ne smijemo zaboraviti da više ljudi može imati ista imena i vidimo da više nemamo jedinstveni osnovi ključ, a tu je dodatna okolnost da se ime i prezime mogu mijenjati (npr. udaja) čime on postaje promjenjiv što nije karakteristika osnovnog ključa. U ovom slučaju možemo uzeti da je izabrani ključ prirodan, dok u slučaju druge tablice mobitel imamo umjetni ključ koji se sastoji od dva slova imena proizvođača i modela mobitela. Još nam ostaje za objasniti strani ključ (*FOREIGN KEY*). Strani ključ predstavlja vrijednosti stupca u jednoj tablici koje se poklapaju sa vrijednošću primarnog ključa u drugoj odnosno pokazuje na točno određeni red u drugoj tablici. Npr. ako pogledamo osobu u tablici posudi vidimo matični broj iz tablice osoba iz

kojeg možemo uzeti ime, prezime itd. U tablici mobitel susrećemo ključnu riječ *UNIQUE*. Ona nam kaže da vrijednosti u poljima proizvodi i model se ne smiju ponoviti više u niti jednom redu. Stavljena je zato što imamo umjetni osnovni ključ te bi neko tko krivo upiše taj osnovni ključ ponovo mogao registrirati isti mobitel pa smo ovako malo postrožili kontrolu unosa. Na kraju možemo zamijetiti ključnu riječ *IDENTITY* odnosno kod SAP DB-a *DEFAULT SERIAL* koja nam povećava polje broj u tablici pri svakom unosu novih podataka za jedan i pritom dobivamo jednu vrstu ključa koja razlikuje svaki red. Pitanje je zašto smo ga stavili tu ? Kad bi neko htio obrisati neki red te tablice mi bi to odmah vidjeli, čak da obriše čitavu tablicu naredbom *DELETE* . Pri idućem unosu novih podataka pod brojem bi bila upisana vrijednost plus jedan od prošlog najvećeg broja u našem slučaju znači kad bi obrisali tablicu sa trenutnih pet redova i unijeli samo jedan red on bio bio pod brojem šest, a ta vrijednost se ne može mijenjati naredbom *UPDATE*. Sad će biti prikazani podaci sa kojim ćemo raditi i objašnjavati daljnje naredbe.

TABLICA OSOBA:

maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Perić	Gajeva 3	Zagreb
0302982383818	Ivan	Ivić		Split
0305972383915	Marko	Marić	Lavova 67	Zadar
1212972383944	Ivan	Dundov	Obala 4	Zagreb
1717985383542	Ivan	Pos	NULL	Zagreb

TABLICA MOBITEL:

sifra	proizvodi	model	tezina	visina	sirina	debljina
no3310	nokia	3310	133	113	48	22
no3330	nokia	3330	133	113	48	22
soz7	sony	z7	95	91	50	25
mov60	motorola	v60	109	85	45	25
soj70	sony	j70	92	133	45	22

TABLICA POSUDI:

broj	osoba	mobitel	uzeo	vratio	napomena
1	0102968383911	no3310	01.01.2002	30.01.2002	
2	0302982383818	soz7	15.01.2002	02.02.2002	
3	0102968383911	no3310	03.03.2002	15.05.2002	
4	1212972383944	mov60	15.02.2002	15.07.2002	
5	0102968383911	no3330	01.06.2002	01.10.2002	

Da su tablice kreirane na MS SQL-u možemo jednostavno provjeriti uz pomoć sistemskih tablica i naredbe *SELECT* koja je obrađena u idućem poglavlju..

Primjer 6:

SELECT name FROM SysObjects WHERE type ='U'

```
name
----------
osoba
dtproperties
mobitel
posudi
```

7. OSNOVNE NAREDBE SQL-a

7.1 NAREDBA SELECT

Osnovna naredba u SQL-u je *SELECT izraz FROM imeTablice* . U prijevodu *IZABERI izraz IZ imeTablice*. U naredbi riječ *izraz* zamjenjujemo sa imenima stupaca koje želimo vidjeti ili u slučaju da želimo vidjeti sve sa *. Možemo pokraj imena stupca napisati novo ime stupca pod kojim ga želimo vidjeti u izvještaju, ali u tablici ostaje sve po starom. Te možemo upisati neki matematički izraz ili tekst unutar navodnika, a u slučaju da imamo više parametara odvajamo ih zarezima.
Primjer 7:

*SELECT 'broj' broj, 100 * 2 daljina, mjesto grad, * FROM osoba*

broj daljina	grad	maticni	ime	prezime	ulica	mjesto
broj 200	Zagreb	0102968383911	Pero	Peric	Gajeva 3	Zagreb
broj 200	Split	0302982383818	Ivan	Ivic		Split
broj 200	Zadar	0305972383915	Marko	Maric	Lavova 67	Zadar
broj 200	Zagreb	1212972383944	Ivan	Dundov	Obala 4	Zagreb
broj 200	Zagreb	1717985383542	Ivan	Pos	NULL	Zagreb

(5 row(s) affected)

Iduća *SELECT* naredba se razlikuje kod MS SQL-a i SAP-a. Vidimo u prethodnom primjeru da svaki stupac ima svoj naziv. No nekad može biti potrebno da u nekom izvještaju spojimo dva stupca tablice u jedan stupac izvještaja. Spajanje se vrši u MS SQL-u sa znakom + dok se ista radnja u SAP-u radi sa znakom naziva pipe || (Alt Gr + W). Primjer će biti izvaditi ime i prezime u stupac pod nazivom naziv.

Primjer 8:

SELECT ime + ' ' + prezime naziv
FROM osoba

naziv

Pero Peric
Ivan Ivic
Marko Maric
Ivan Dundov
Ivan Pos

(5 row(s) affected)

SELECT ime || ' ' || prezime naziv
FROM osoba

NAZIV
Pero Peric
Ivan Ivic
Marko Maric
Ivan Dundov
Ivan Pos

Možemo još spomenuti i ključnu riječ *TOP x* koja nam daje samo prvih x redova. Što je korisno pri radu na velikim tablicama kad često moramo provjeravati rezultate naših izraza.
Primjer 9:

SELECT TOP 2 ime + ' ' + prezime naziv FROM osoba

naziv

Pero Peric
Ivan Ivic

(2 row(s) affected)

Postoji još jedna mogućnost korištenja naredbe *SELECT* bez dodatka *FROM* jer taj podatak ne mora biti spremljen u tablicu. Primjer traženja podatka sadašnjeg vremena:

Primjer 10:

SELECT GETDATE() 'datum i sat';

```
datum i sat
--------------------------
2002-08-05 16:15:15.167

(1 row(s) affected)
```

Idući koristan dodatak naredbi je uvjet WHERE koji kaže koje redove želimo vidjeti jer u većini prethodnih primjera imamo pokazane sve redove.

Za početak možemo vidjeti primjere kroz operatore usporedbe :

= jednako, > veće od, < manje od, >= veće ili jednako, <= manje ili jednako i != ili < > za različito od. Idući primjer će pokazati sve mobitele u bazi koji su lakši od 100 grama (*tezina <= 100*).

Primjer 11:

SELECT proizvodi, model, tezina FROM mobitel WHERE tezina <= 100

```
proizvodi      model          tezina
-------------- -------------- -----------
sony           j70            92
sony           z7             95

(2 row(s) affected)
```

7.1.1. LOGIČKI OPERATORI AND, OR i NOT

Daljnji korak bi bio upotreba logičkih operatora tako da možemo još više proširiti uvjete za pretraživanje. Logički operatori su *AND, OR* i *NOT*. Kod složenijih izraza bi trebalo voditi računa o prioritetima. Hijerarhija primjene operatora glasi:

zagrada (),
dijeljenje / i množenje * ,
zbrajanje + i oduzimanje - ,
NOT (ne),
AND (i),
OR (ili).

Sada već možemo usporediti težinu mobitela i volumen te naći mobitel koji je lakši od 100 grama (*tezina <= 100*) i (*AND*) ima manji volumen od 120000 mm^3 (*visina*sirina*debljina < 120000*).
Primjer 12:

```
SELECT proizvodi, model, tezina, visina*sirina*debljina 'Volumen mm3'
        FROM mobitel
       WHERE tezina <= 100 AND visina*sirina*debljina < 120000
```

proizvodi	model	tezina	Volumen mm3
sony	z7	95	113750

(1 row(s) affected)

U slučaju da uvjet iza WHERE stavimo u NOT zagradu dobili bi suprotnu vrijednost izvještaja odnosno samo one mobitele koji nisu navedeni u prethodnom primjeru što je i normalno jer uvjet sada glasi *oni koji nisu.*

Primjer 13:

```
SELECT proizvodi, model, tezina, visina*sirina*debljina 'Volumen mm3'
        FROM mobitel  WHERE
            NOT (tezina <= 100 AND visina*sirina*debljina < 120000)
```

proizvodi	model	tezina	Volumen mm3
motorola	v60	109	95625
nokia	3310	133	119328
nokia	3330	133	119328
sony	j70	92	131670

(4 row(s) affected)

I na kraju pri korištenju logičkog operatora *OR* kao izlaz dobivamo sve mobitele jer svi zadovoljavaju makar jedan od uvjeta da je lakši od 100 grama (*tezina <= 100*) ili (*OR*) ima manji volumen od 120000 mm^3 (*visina*sirina*debljina < 120000*).

Primjer 14:

```
SELECT proizvodi, model, tezina, visina*sirina*debljina 'Volumen mm3'
     FROM mobitel
    WHERE tezina <= 100 OR visina*sirina*debljina < 120000
```

proizvodi	model	tezina	Volumen mm3
motorola	v60	109	95625
nokia	3310	133	119328
nokia	3330	133	119328
sony	j70	92	131670
sony	z7	95	113750

(5 row(s) affected)

Normalno je da se ovi nabrojani operatori mogu primijeniti i na ostale vrste podataka kao što su datum (*DATE*) ili znakovi (*VARCHAR*).

7.1.2 IN i BETWEEN

Ako želimo definirati u uvjetu raspon imamo dvije mogućnosti. Jedna je da radimo sa operatorima usporedbe (<, >, = ...) , a druga je da koristimo ključnu riječ *BETWEEN*. Znači ako hoćemo naći mobitel težine između (*BETWEEN*) 90 i 100 grama imamo dva različita koda i isti izlaz.
Primjer 15:

```
SELECT  proizvodi, model, tezina FROM mobitel
            WHERE tezina > 90 AND tezina < 100;
                        ili
SELECT proizvodi, model, tezina FROM mobitel
            WHERE tezina BETWEEN 90 AND 100;
```

proizvodi	model	tezina
sony	j70	92
sony	z7	95

(2 row(s) affected)

Iz prijašnjih primjera slijedi da bi *NOT BETWEEN* kao suprotna naredba dao suprotno rješenje odnosno tri mobitela koja nisu navedena u gornjem primjeru.

Primjer 16:

proizvodi	model	tezina
motorola	v60	109
nokia	3310	133
nokia	3330	133

(3 row(s) affected)

Kad želimo naći podatak kojem je uvjet da se poklapa sa jednom od vrijednosti u listi koristimo ključnu riječ *IN* . To bi mogli dobro prikazati kroz primjer imena i prezimena svih osoba koje žive u Zadru ili Zagrebu. Imamo tri načina koji će nam dati jedno te isto rješenje.

Primjer 17:

S*ELECT ime, prezime FROM osoba*

WHERE mjesto IN ('Zagreb', 'Zadar');

ili

SELECT ime, prezime FROM osoba WHERE mjesto >= 'Zadar';

ili

SELECT ime, prezime FROM osoba

WHERE mjesto = 'Zagreb' OR mjesto = 'Zadar';

```
ime            prezime
-------------- ---------------
Pero           Peric
Marko          Maric
Ivan           Dundov
Ivan           Pos

(4 row(s) affected)
```

Kao što je prije rečeno možemo to riješiti kroz operatore usporedbe jer oni vrijede i za nizove znakova te jedan niz znakova može biti jednak ili veći od nekog drugog. A drugi je način da nabrojimo listu i kažemo da tražimo ime i prezime osobe koja je iz mjesta koja su nabrojano u (*IN*) listi (*('Zagreb', 'Zadar')*). Lista se nalazi uvijek u zagradama i njezini elementi su odvojeni zagradama. I jedan primjer iz prakse, napraviti adresar osoba koje imaju upisanu adresu ili napisati posebno u svakom redu ime i prezime; ulicu; mjesto te idući podatak odvojiti praznim redom.

Primjer 18:

```
SELECT ime + ' ' + prezime + '
' + ulica + '
' + mjesto + '
' + '
' adresar
FROM osoba WHERE ulica NOT IN ('')
```

adresar

Pero Peric
Gajeva 3
Zagreb

Marko Maric
Lavova 67
Zadar

Ivan Dundov
Obala 4
Zagreb

(3 row(s) affected)

Važno je kao prijelaz u novi red (između jednostrukih navodnika) stisnuti ENTER. Ovako se može imati baza korisnika podijeljena u različite grupe (profile) te automatizirati štampanje adresa.

7.1.3. LIKE

Nekad se može dogoditi da tražimo npr. neko ime koje počinje kao ... ili telefonski broj sadrži sigurno ove znamenke za redom. Tako zadane uvjete je malo teže dobiti kroz prošle primjere, ali zato služi ključna riječ *LIKE.* Ako se vratimo na prethodni primjer imena i prezimena svih osoba koje žive u Zadru ili Zagrebu i dodamo ili nekom sličnom mjestu (*mjesto LIKE 'Za%'*) te još uvjet da im ime počinje nekako na I (Ivan, Ivica, Ivana, Ivo...) (*ime LIKE 'I%'*), možemo napisati slijedeći kod:

Primjer 19:

SELECT ime, prezime, mjesto FROM osoba
WHERE mjesto LIKE 'Za%' AND ime LIKE 'I%'

```
ime          prezime       mjesto
------------ ------------- ---------------
Ivan         Dundov        Zagreb
Ivan         Pos           Zagreb

(2 row(s) affected)
```

U kodu se javlja tzv. joker znak % koji predstavlja niz bilo kakvih znakova. U primjeru je % na kraju (*LIKE 'Za%'*), ali isto tako on se može nalaziti na početku (*LIKE '%reb'*) ili u sredini izraza koji tražimo (*LIKE 'Za%eb'*), a mora ih biti jedan ili više. Postoji još jedan joker znak. To je podvučeno _ (underscore ili underbar). On nam predstavlja zamjenu za bilo koji znak, ali samo jedan znak. Npr. ako tražimo ime osobe koja živi u gradu koji počinje sa Za , a naziv grada ima pet znakova, možemo skratiti listu ponuđenih rezultata jer unutra neće biti Zagreb kroz kod :
Primjer 20:

SELECT ime, prezime, mjesto FROM osoba
WHERE mjesto LIKE 'Za___'

```
ime          prezime       mjesto
------------ ------------- ---------------
Marko        Maric         Zadar

(1 row(s) affected)
```

No postavlja se pitanje što u slučaju kad tražimo neki izraz koji u sebi imaju joker znak % ili _ i kao takvi predstavljaju nešto drugo nego nam je potrebno u ovom slučaju. Tad koristimo takozvani Escape znak koji kaže da znak koji slijedi iza njega je joker znak, ali mi tražimo baš taj znak u zadanoj tablici. Npr. ... *LIKE '%@%%' ESCAPE '@'* traži bilo što gdje se nalazi znak %. U tablici osoba nalazi se stupac matični u koji upisujemo JMBG taj stupac nam je tipa *NVARCHAR(15)* što znači da mi tu možemo upisati i slova, a to nikako ne odgovara prijašnjoj tvrdnji da tu trebaju biti samo brojevi pa je potrebno ograničiti unos podataka. U SQL-u imamo naredbu *CONSTRAINT* (ograničenje) kojom možemo opisati kako podatak treba izgledati.

Primjer 21:

```
ALTER TABLE osoba ADD CONSTRAINT kriviJMBG CHECK
        (maticni like '[0,1,2,3][0-9][0-1][0-9][0,9]%' )
```

Ako poznajemo strukturu JMBG-a znamo da se na početku nalazi datum rođenja, a to možemo malo uobličiti naredbom *LIKE*. Prvi znak može biti 0 ili 1 ili 2 ili 3 jer mjesec ima maksimalno 31 dan, drugi znak je bilo koja znamenka od 0 do 9. Treća znamenka označava prvi broj u oznaci mjesec koji može biti samo 0 ili 1 jer imamo 12 mjeseci itd. Na kraju je dodan joker znak % radi kraćeg koda koji kaže da tu može biti bilo što iako je bolje staviti da su tu samo znamenke. Za primjer ćemo probati popuniti tablicu sa osobom koja je rođena 41.02.1968 godine.

Primjer 22:

```
INSERT INTO osoba VALUES ('4102968383912','?','?','?','?')
```

Server: Msg 547, Level 16, State 1, Line 1
INSERT statement conflicted with COLUMN CHECK constraint '**kriviJMBG**'.
The conflict occurred in database 'mobitel', table 'osoba', column 'maticni'.
The statement has been terminated.

Za kraj možemo ponovo nabrojiti kad se služimo dosta sličnim naredbama jednako =, *IN, LIKE*. Jednako koristimo kad tražimo da se nešto poklapa sa jednom vrijednošću bilo koje vrste podataka. *IN* služi kad moramo uspoređivati sa više vrijednosti te ne moramo koristiti seriju *OR* i na kraju *LIKE* služi kad tražimo nekakav uzorak.

7.1.4. IS NULL

Na početku smo opisali stupac ulica u tablici osoba kao takav da nije nužno upisati podatak. Ipak poželjno je znati za koga nemamo podatke u tablici, a upravo to nam daje *IS NULL*.
Primjer 23:

```
SELECT ime, prezime FROM osoba
        WHERE ulica IS NULL
```

```
ime             prezime
--------------- ---------------
Ivan            Pos

(1 row(s) affected)
```

Ipak postoji razlika između NULL i praznog skupa znakova te moramo staviti još uvjet koji kaže da ulica može biti prazan skup znakova ".

Primjer 24:

SELECT ime, prezime FROM osoba
WHERE ulica =" OR ulica IS NULL

ime	prezime
Ivan	Ivic
Ivan	Pos

(2 row(s) affected)

7.1.5. ORDER BY

Uz korištenje ove ključne riječi možemo dobiti da nam rezultat bude čitljiviji, odnosno možemo rezultat sortirati po nekom redoslijedu u idućem primjeru neka to bude težina i model.
Primjer 25:

SELECT proizvodi, model, tezina FROM mobitel
ORDER BY tezina , model

proizvodi	model	tezina
sony	j70	92
sony	z7	95
motorola	v60	109
nokia	3310	133
nokia	3330	133

(5 row(s) affected)

Vidimo da smo dobili sortiran rezultat po težini od najmanje prema najvećoj, a u slučaju iste težine sortirali smo po modelu. Također postoji i naredba za suprotno sortiranje od najtežeg do najlakšeg mobitela (*DESC*) te idućem primjeru sortiramo po tom uvjetu težinu i po suprotnom uvjetu model tj. rastućem nizu (*ASC*). Ako se ne spomene izričito način sortiranja podrazumijeva se rastući niz.

Primjer 26:

```
SELECT proizvodi, model, tezina FROM mobitel
        ORDER BY tezina DESC, model ASC
```

proizvodi	model	tezina
nokia	3310	133
nokia	3330	133
motorola	v60	109
sony	z7	95
sony	j70	92

Isti rezultat možemo dobiti i skraćenim pisanjem koda tako da imena stupaca iza *GROUP BY* ključne riječi zamijenimo sa rednim brojem pozicije u *SELECT*-u. kao što pokazuje idući primjer.

Primjer 27:

```
SELECT proizvodi, model, tezina FROM mobitel
        ORDER BY 3 DESC, 2 ASC
```

proizvodi	model	tezina
nokia	3310	133
nokia	3330	133
motorola	v60	109
sony	z7	95
sony	j70	92

Zanimljiv je primjer kad bi željeli sortirati po modelu koji je tipa *NVARCHAR* što znači da sad nisu u pitanju samo brojevi već imamo podatke koji počinju sa slovima, ali i sa brojevima. Rezultat takvog sortiranja vidimo u slijedećem primjeru.

Primjer 28:

SELECT proizvodi, model, tezina FROM mobitel ORDER BY 2

proizvodi	model	tezina
nokia	3310	133
nokia	3330	133
sony	j70	92
motorola	v60	109
sony	z7	95

(5 row(s) affected)

Kad imamo samo brojeve ili samo slova jasno je kako će biti sortirani ti nizovi međutim kad je sve to izmiješano postavlja se pitanje kako će sad to biti sortirano? Tu možemo pozvati u pomoć jednu proceduru u MS SQL-u koja će nam dati redoslijed niza po kojem se vrši sortiranje.

Primjer 29:

EXEC SP_HELPSORT

Unicode data sorting

Locale ID = 1050

case insensitive, kana type insensitive, width insensitive

Sort Order Description

Character Set = 1, iso_1

ISO 8859-1 (Latin-1) - Western European 8-bit character set.

Sort Order = 52, nocase_iso

Case-insensitive dictionary sort order for use with several We

stern-European languages including English, French, and German

. Uses the ISO 8859-1 character set.

Characters, in Order

! " # $ % & ’ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ‘ { | }

~ ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾

¿ × ÷ 0 1 2 3 4 5 6 7 8 9 A=a À=à Á=á Â=â Ã=ã Ä=ä Å=å Æ=æ B=b C

=c Ç=ç D=d E=e È=è É=é Ê=ê Ë=ë F=f G=g H=h I=i Ì=ì Í=í Î=î Ï=ï J

=j K=k L=l M=m N=n Ñ=ñ O=o Ò=ò Ó=ó Ô=ô Õ=õ Ö=ö Ø=ø P=p Q=q R=r S

=s ß T=t U=u Ù=ù Ú=ú Û=û Ü=ü V=v W=w X=x Y=y Ý=ý ÿ Z=z Ð=ð Þ=þ

Ipak može se pojaviti jedna zanimljiva situacija, a to je da u znakovnom polju imamo brojeve koje treba sortirati po veličini. U našim primjerima tablica nema takvog jer je to ipak iznimka koju možemo oponašati tako

da iza sortiranja (*ORDER BY*) cjelobrojni znak pretvorimo u znakovni podatak (*CAST(visina AS VARCHAR)*).
Primjer 30:

```
SELECT visina FROM mobitel
    ORDER BY CAST(visina AS VARCHAR)
```

```
visina
-----------
113
113
133
85
91

(5 row(s) affected)
```

Da je u pitanju cjelobrojna vrijednost brojevi bi bili normalno poredani od manjeg prema najvećem dok tu na prvi pogled nema nikakvog smisla. Kad ovo gledamo kao znak 1 je u nizu sortiranja ispred 8 ili 9, a uspoređivanje počinje od prvog znaka pa će bilo koliko velik broj upisan kao znak koji počinje sa 1 uvijek biti ispred bilo čega što počinje sa nekom drugom znamenkom ma kako malen bio jer tu brojeve moramo gledati kao znakove što je onda jasnije jer ne može riječ na B biti ispred one koja počinje sa A zato što ima manje znakova u sebi. Najbrže rješenje je to pretvoriti u brojeve pomoću naredbe *CAST* ali ne smijemo zaboraviti da nam jedno slovo greškom upisano može izbaciti tada grešku tipa:

```
Server: Msg 245, Level 16, State 1, Line 1
Syntax error converting the varchar value 'xxx'
to a column of data type int.
```

još gore je ako taj stupac mora biti znakovni što je u praksi moguće. To rješavamo tako da moramo obrisati praznine ako postoje (*LTRIM*) na početak svakog broja dodati nule (*'00' +*) i skraćujemo znakovni niz tako da svi budu iste duljine (*RIGHT*)
Primjer 31:

```
SELECT  visina,  '00' +
        LTRIM( CAST(visina AS VARCHAR))  'prije kraćenja',
        RIGHT ('00' +
        LTRIM (CAST(visina AS VARCHAR)), 3) 'vrijednost usporedbe'
        FROM mobitel ORDER BY RIGHT ('00' +
                LTRIM (CAST(visina AS VARCHAR)), 3)
```

visina	prije kraćenja	vrijednost usporedbe
85	0085	085
91	0091	091
113	00113	113
113	00113	113
133	00133	133

(5 row(s) affected)

Sad uspoređujemo 085, 091, 113, a znak 0 je ispred 1 pri sortiranju i sve ispada kao da smo sortirali brojčane vrijednosti. U ovim slučajevima iza naredbe *ORDER BY* imali smo nekakve izraze što znači da tu ne mora biti samo naziv ili broj stupca.

7.1.6. DISTINCT i ALL

Ako bi željeli vidjeti sve proizvođače mobitela iz prethodnog teksta je jasno je da bi napisani kod glasio:
Primjer 32:

SELECT proizvodi FROM mobitel

```
proizvodi
---------------
motorola
nokia
nokia
sony
sony

(5 row(s) affected)
```

Ipak zbunjujuće je to što se neki proizvođači ponavljaju pa nismo sigurni koji su to ili koliko ih ima. Zapravo gornja naredba je skraćeni ispis ovog programskog koda:
Primjer 33:

SELECT ALL proizvodi FROM mobitel

Koji daje isti rezultat kao gore navedeni. Sada je jasnije da smo tražili da nam ispiše sve proizvođače koji se nalaze u tablici, a ključna riječ koja nam neće dati duplikate u rezultatu je DISTINCT.

Primjer 34:

SELECT DISTINCT proizvodi FROM mobitel

proizvodi
---
motorola
nokia
sony

(3 row(s) affected)

8.1. FUNKCIJE SAKUPLJANJA

U funkcije sakupljanja spadaju *SUM* (suma svih vrijednosti), *AVG* (prosječna vrijednost), *COUNT* (broj redova dobivenih rezultata), *MAX* (maksimalna vrijednost u izrazu), *MIN* (minimalna vrijednost u izrazu), *STDEV* (standardna devijacija), *VAR* (varijanca). Vrijedi općenito za sve vrste funkcija da bi trebalo konzultirati upute da znamo koje su nam dostupne. Ako bolje pogledamo vrijednost pod stupcem *broj* vidimo da je tu broj pet. On nam kaže nad koliko redova su provedene ove funkcije jer funkcije sakupljanja bi se trebale vršiti nad dva ili više redova. Objašnjenje je jednostavno u slučaju da radimo nad jednim redom ili nula kada neće doći do greške, ali je besmisleno računati sumu ili srednju vrijednost jednog reda. Parametar koji dolazi iza funkcije nalazi se u zagradama.

Primjer 35:

```
SELECT 'TEŽINA' , SUM(tezina) 'UKUPNA', AVG (tezina) 'PROSJEK',
        COUNT(tezina) 'BROJ', MAX (tezina) 'MAX',
       MIN(tezina) 'MIN', ROUND(STDEV(tezina), 0) 'DEV',
       ROUND(VAR (tezina), 0) 'VAR' FROM mobitel
```

	UKUPNA	PROSJEK	BROJ	MAX	MIN	DEV	VAR
TEŽINA	562	112	5	133	92	20.0	395.0

(1 row(s) affected)

8.2. MATEMATIČKE FUNKCIJE

Matematičke funkcije su *ABS* (apsolutna vrijednost), *ACOS* (arkus kosinus), *ASIN* (arkus sinus), *ATAN* (arkus tangens), *CEILING* (najmanja cjelobrojna vrijednost veća od vrijednosti zadanog izraza), *COS* (kosinus), *COT* (kotangens), *FLOOR* (najveća cjelobrojna vrijednost manja od vrijednosti zadanog izraza), *LOG* (logaritam baze 2), *LOG10* (logaritam baze 10), *PI* (3,14), *POWER* (potenciranja), *RADIANS* (pretvara stupnjeve u radijane), *RAND* (generator slučajnih brojeva), *ROUND* (zaokruživanje decimalnih brojeva na zadanu preciznost), *SIGN* (predznak izraza kao izlaz daje nula, ako je parametar nula ili jedan ako je parametar veći od nule ili minus jedan ako je parametar manji od nule), *SIN* (sinus), *SQRT* (drugi korijen), *TAN* (tangens). Parametar koji dolazi iza funkcije nalazi se u zagradama, ako ga funkcija traži inače imamo prazne zagrade kao kod *RAND()*.

Primjer 36:

```
SELECT ROUND(RAND()*100, 0) 'SLUČAJAN',
       SIGN(-33) 'NEGATIVAN', LOG10(10) 'LOG',
       COS(0) 'COS(0)', ROUND(PI(), 2) 'PI'
```

```
SLUČAJAN    NEGATIVA N     LOG      COS(0)     PI
----------  ----------  --------  ----------  ----
12.0            -1        1.0       1.0      3.14

(1 row(s) affected)
```

ili još jedan primjer sa preostalim funkcijama

Primjer 37:

SELECT '5.3 ' BROJ, CEILING(5.3) 'CEILING', FLOOR(5.3) 'FLOOR',
POWER (5.3, 3)'NA 3-u', SQRT(5.3) 'KORIJEN'

```
BROJ    CEILING    FLOOR    NA 3-u     KORIJEN
------  ---------- ---------  ---------- -----------------
5.3            6          5      148.9  2.3021728866442674

(1 row(s) affected)
```

Gornji primjeri obrađuju ključnu riječ *SELECT* bez dodatka *FROM*, normalno je da smo mogli koristiti taj dodatak i podatke za obradu matematičkih funkcija uzimati iz tablica.

Primjer 38:

SELECT TEZINA, SQRT(tezina) 'KORIJEN',
POWER(tezina, 2)'KVADRAT', SIGN(tezina) 'PREDZNAK'
FROM mobitel WHERE proizvodi ='Sony'

```
TEZINA     KORIJEN                                KVADRAT     PREDZNAK
---------- ------------------------------------------------- ----------- -----------
92         9.5916630466254382                        8464           1
95         9.7467943448089631                        9025           1

(2 row(s) affected)
```

Ili možemo koristiti te funkcije iza uvjeta *WHERE* npr. u primjeru ćemo pogledati mobitele čiji je drugi korijen veći od deset.

Primjer 39:

SELECT proizvodi, model, tezina, SQRT(tezina) 'korijen'
FROM mobitel WHERE SQRT(tezina) > 10

proizvodi	model	tezina	korijen
motorola	v60	109	10.440306508910551
nokia	3310	133	11.532562594670797
nokia	3330	133	11.532562594670797

(3 row(s) affected)

U nekim gornjim uvjetima izvršili smo *SELECT* bez dodatka *FROM*, no to ne radi na svim bazama. Što ćemo jasno vidjeti u slijedećem primjeru.
Primjer 40:

MS SQL

SELECT PI() 'PI'

PI

3.1415926535897931

(1 row(s) affected)

SAP DB

SELECT PI() "PI"

Database error occured:
Native error: -5015
Syntax error or access violation

U drugom slučaju ipak moramo navesti dodatak FROM i ime neke postojeće tablice i onda će to biti u redu.

Primjer 41:

SELECT PI() "PI" FROM osoba

PI
3,14159265358979
3,14159265358979
3,14159265358979
3,14159265358979
3,14159265358979

No sad imamo pet redova odnosno onoliko koliko imamo ukupno redova u tablici osoba, što opet nije odgovarajuće rješenje i moramo koristiti neke agregatne funkcije.
Primjer 42:

SELECT DISTINCT(PI()) "PI" FROM osoba

PI
3,14159265358979

U slučaju da želimo nekakve testne uzorke redova u tablici posudi na primjer svaki drugi red uzeti također koristimo matematičku funkciju, to je % modulo, ona daje ostatak cjelobrojnog dijeljenja. Nama je za svaki red karakterističan njegov broj pa možemo gledati ostatak dijeljenja tog broja sa dva i ako je nula ispisati ga odnosno u ovom slučaju ispisujemo svaki drugi red.

Primjer 43:

*SELECT * FROM posudi WHERE broj % 2 = 0*

broj	osoba	mobitel	uzeo	vratio	napomena
2	0302982383818	soz7	2002-01-15 00:00:00.000	2002-02-02	
4	1212972383944	mov60	2002-02-15 00:00:00.000	2002-07-15	

(2 row(s) affected)

8.3. FUNKCIJE ZA RAD NA NIZOVIMA ZNAKOVA

LEFT (izdvaja znakove počevši od lijevog kraja), *LEN* (daje dužinu znakovnog niza), *LOWER* (sva slova znakovnog niza pretvara u mala slova), *LTRIM* (uklanja znakove razmaka na početku znakovnog niza), *REPLACE* (zamjenjuje u zadanom nizu znakova određeni dio drugim dijelom), *RIGHT* (izdvaja znakove počevši od desnog kraja niza), *RTRIM* (uklanja eventualne razmake na kraju znakovnog niza), *SUBSTRING* (daje zadani broj znakova počevši također od zadanog mjesta u znakovnom nizu), *UPPER*(sva slova znakovnog niza pretvara u velika slova), *SOUNDEX*(daje Soundex kod znakovnog niza. Dva niza koja zvuče slično, a pišu se različito daju jednake kodove)

Primjer 44:

```
SELECT   LEN(ime) + LEN(prezime)  'slova u nazivu LEN()',
            SOUNDEX(ime) 'SOUNDEX(Pero)',
            SOUNDEX(SUBSTRING(prezime, 1, 4))  'SOUNDEX(Peri)',
            LEFT(prezime, 4) 'LEFT(prezime, 4)' ,
            SUBSTRING(prezime, 1, 4) 'SUBSTRING(prez, 1, 4)'
      FROM  osoba WHERE maticni ='0102968383911'
```

```
slova u nazivu LEN() SOUNDEX(Pero) SOUNDEX(Peri) LEFT(prezime, 4) SUBSTRING(prez, 1, 4)
-------------------- ------------- ------------- ---------------- ------------------------------------------------------------
9                    P600          P600          Peri             Peri

(1 row(s) affected)
```

Funkcije *UPPER* i *LOWER* su zanimljive sa stanovišta pretraživanja.

U bazi možemo imati neko polje npr. ime upisano na različite načine sa gledišta velikih i malih slova (IVAN, Ivan, ivan ili greškom i druge kombinacije). Ako upišemo

Primjer 45:

*SELECT * FROM osoba WHERE ime ='IVAN'*

nećemo dobiti nijedan rezultat vani iako se nama neke stvari podrazumijevaju da su jednake pa ovako koristeći standardno pretraživanje možemo dobiti samo dio rezultata. Tu nam u pomoć uskaču ove funkcije jedna ili druga tako da dobiveni podatak promijenimo i onda uspoređujemo sa našim nizom znakova ili isto kroz primjer. Znači znakovni niz ime pretvorimo u velika slova *UPPER(ime)* i tek onda uspoređujemo sa nekom drugom vrijednošću kojoj su normalno svi znakovi velika slova *='IVAN'*.

Primjer 46:

*SELECT * FROM osoba WHERE UPPER(ime) ='IVAN'*

MATICNI	IME	PREZIME	ULICA	MJESTO
0302982383818	Ivan	Ivic		Split
1212972383944	Ivan	Dundov	Obala 4	Zagreb
1717985383542	Ivan	Pos	?	Zagreb

8.4. FUNKCIJE ZA RAD SA DATUMIMA

Za MS SQL server imamo ove funkcije *DATEADD*(datumu dodaje određeno vrijeme), *DATEDIFF*(daje broj datumskih dijelova zadanog tipa između dva tipa), *DATENAME*(izdvaja tekstualna imena), *DATEPART*(daje dio datuma zadanog tipa iz ulaznog tipa), *DAY*(daje dan zadanog datuma), *GETDATE*(daje trenutno vrijeme i datum), *MONTH*(daje mjesec zadanog datuma), *YEAR* (daje godinu zadanog datuma). U funkcijama za rad sa datumima susrećemo argumente koji će biti objašnjeni u slijedećoj tablici.

Argument	Značenje
yy ili *yyy*	godina
qq ili *q*	tri mjeseca
mm ili *m*	mjesec
wk ili *ww*	nedjelja
dy ili *y*	dan u godini
dd ili *d*	dan
hh	sat
mi ili *n*	minut
ss ili *s*	sekunda
ms	milisekunda

Tablica : argumenti za rad sa funkcijama datuma

Primjer 47:

```
SELECT uzeo, vratio,  DATEDIFF(d, uzeo, vratio) 'razlika',
        DATEADD(d, 3, vratio) 'vratio +3 dana'
    FROM posudi WHERE broj = 1
```

```
uzeo                        vratio                      razlika     vratio +3 dana
--------------------------- --------------------------- ----------- --------------------------
2002-01-01 00:00:00.000     2002-01-30 00:00:00.000     29          2002-02-02

(1 row(s) affected)
```

Primjer 48:

```
SELECT CAST(GETDATE() AS VARCHAR) + ';      Danas je ' +
         CAST(DATEPART(d, GETDATE()) AS VARCHAR) +  ' dan ' +
         DATENAME(MONTH, GETDATE()) +  '-a ' +
         CAST(YEAR(GETDATE()) AS VARCHAR) + ' godine, ' +
        CAST(DATEPART(hh, GETDATE()) AS VARCHAR)+  ' sati i ' +
        CAST(DATEPART(n, GETDATE()) AS VARCHAR) +  ' minute'
       ' SQL upit za datum i vrijeme, a izlaz je rečenica'
```

```
 SQL upit za datum i vrijeme, a izlaz je rečenica
-----------------------------------------------------------------------------------------------
kol 10 2002 12:53PM;      Danas je 10 dan kolovoz-a 2002 godine, 12 sati i 53 minute

(1 row(s) affected)
```

9. GROUP BY ... HAVING

Ako bi željeli vidjeti iz tablice posudi tko je koliko dana držao mobitel napisali bi smo :
Primjer 49:

```
SELECT  osoba,
        SUBSTRING(CAST(uzeo AS VARCHAR), 1, 11) 'početak',
        SUBSTRING(CAST(vratio AS VARCHAR), 1, 11) 'kraj',
        DATEDIFF(d,uzeo, vratio) 'ukupno'
    FROM posudi
```

```
osoba              početak         kraj          ukupno
-------------- ----------- ----------- --------------------
0102968383911   sij  1 2002     sij 30 2002    29
0302982383818   sij 15 2002    vel  2 2002   18
0102968383911   ožu  3 2002   svi 15 2002   73
1212972383944   vel 15 2002   srp 15 2002   150
0102968383911   lip  1  2002    lis  1  2002    122

(5 row(s) affected)
```

No vidimo da se neke osobe ponavljaju i da bi nam ipak korisniji rezultat bio kad bi imali u rezultatu osobu i koliko je ukupno posudila dana mobitel (npr. kad bi morali naplatiti uslugu), tj. morali bi nekako grupirati izlaz po osobi (*GROUP BY osoba*) što je i svrha nove naredbe.

Primjer 50:

```
SELECT  osoba,
        SUM(DATEDIFF(d,uzeo, vratio)) 'ukupno dana',
        SUM(DATEDIFF(d,uzeo, vratio))*13 'platiti kuna',
       COUNT(osoba) 'posudio puta'
    FROM posudi
    GROUP BY osoba
```

osoba	ukupno dana	platiti kuna	posudio puta
0102968383911	224	2912	3
0302982383818	18	234	1
1212972383944	150	1950	1

(3 row(s) affected)

Ali vidimo velike razlike u danima korištenja te bi možda trebalo dati neku nagradu onima koji su koristili mobitel više od 120 dana (*HAVING SUM(DATEDIFF(d,uzeo, vratio)) > 120*). Njima možemo dati popust od 50% (*SUM(DATEDIFF(d,uzeo, vratio))*13*0.5 'platiti kuna'*). Ključna riječ *HAVING* nam je vrsta uvjeta koja određuje koji će od dobivenih redova biti prikazani, a njen rezultat je logički tip.

Primjer 51:

```
SELECT  osoba,
        SUM(DATEDIFF(d,uzeo, vratio)) 'ukupno dana',
        SUM(DATEDIFF(d,uzeo, vratio))*13*0.5 'platiti kuna',
        COUNT(osoba) 'posudio puta'

FROM posudi
    GROUP BY osoba
   HAVING SUM(DATEDIFF(d,uzeo, vratio)) > 120
```

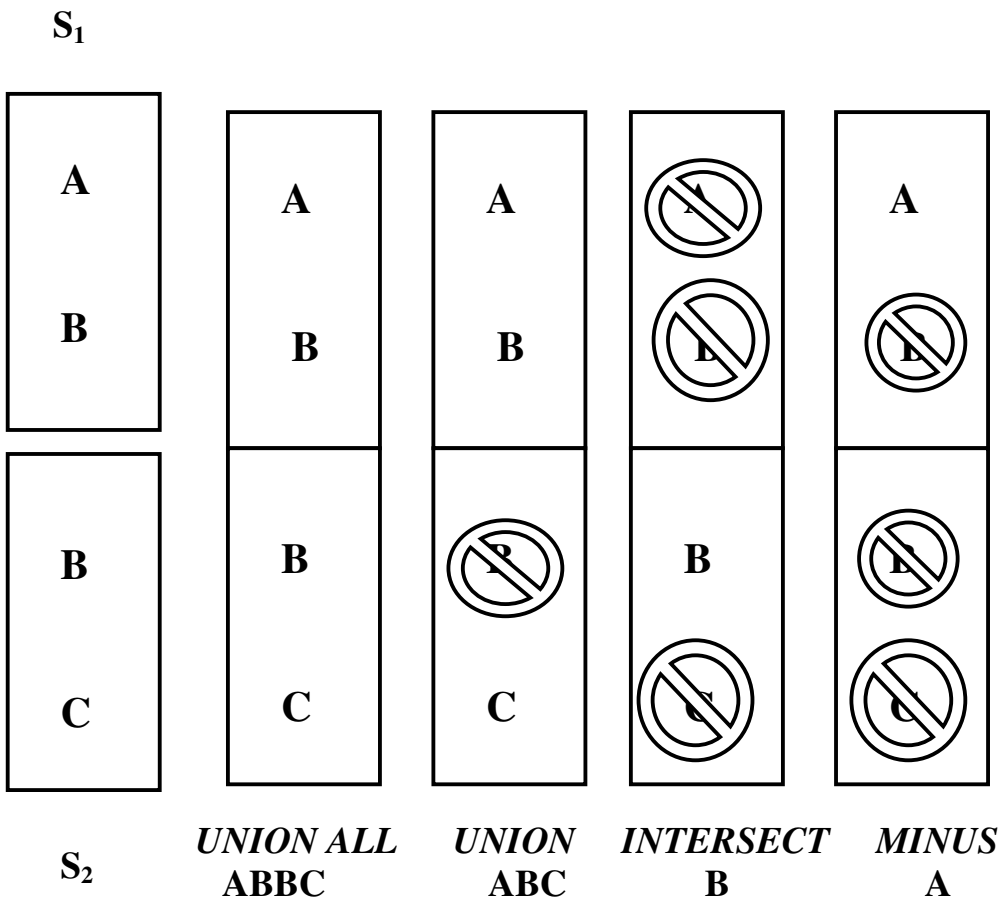
```
osoba          ukupno dana  platiti kuna  posudio puta
-------------- ----------- ------------- ------------ -----------------
0102968383911  224           1456.0        3
1212972383944  150            975.0        1

(2 row(s) affected)
```

Ključna riječ *GROUP BY* ponaša se kao *ORDER BY* pa nema potrebe za sortiranjem rezultata, a dodatak *HAVING* nam omogućava da koristimo agregatne funkcije kod uspoređivanja.

10. OPERACIJE SA SKUPOVIMA

SQL je baziran na teoriji skupova. Naredbe sa skupovima su *UNION, UNION ALL, INTERSECT, MINUS*. Ovdje za razliku od spajanja radimo sa rezultatima nekih izraza, a ne sa tablicama. Prvo ćemo grafički prokazati te operacije.



SKUP S_1 ={ A; B }

SKUP S_2 ={ B; C }

S_1 *UNION ALL* S_2 = { A; B; B; C }

S_1 *UNION* S_2 ={ A; B; C }

S_1 *INTERSECT* S_2 = { B }

S_1 *MINUS* S_2 = { A }

Prva slika predstavlja dva zasebna skupa. Jedan skup ima elemente A i B, a drugi B i C. Kasnije slike prikazuju spojene skupove preko zajedničkih elemenata, ispod su dopisane operacije i rješenja operacija na tim skupovima. *UNION ALL* nam daje sve elemente skupa s tim da se duplikati ponavljaju onoliko puta koliko ih ima. *UNION* također vraća sve elemente skupova s tim da se duplikati ne ponavljaju više puta, sve su vrijednosti jedinstvene. *INTERSECT* nam vraća zajedničke elemente skupova tj. samo duplikate koji su jedinstveni, ne ponavljaju se. *MINUS* daje redove iz prvog izraza koji nisu prisutni u rezultatu drugog izraza. *MINUS* i *INTERSECT* nisu uvijek uključeni tako da ćemo ovisno o bazi možda morati koristiti drugi kod da bi dobili isto.

Primjer 52:

SELECT ime FROM osoba

UNION ALL

SELECT ime FROM osoba

UNION ALL

SELECT 'Ante'

Pero

Ivan

Marko

Ivan

Ivan

Pero

Ivan

Marko

Ivan

Ivan

Ante

(11 row(s) affected)

Kao što je prije navedeno kod *UNION ALL* dobijemo u rezultatu sve redove bez ikakvih oduzimanja za razliku od slijedećeg primjera gdje smo samo zamijenili *UNION ALL* sa *UNION* -om pa vani dobivamo samo različite elemente i ne vidimo koji se ponavljaju. Zapravo je prvi *UNION* nepotreban jer radimo na istoj tablici tako da je to beskorisno spajanje potpuno isti rezultat dobili bi da smo izvršili samo drugi *UNION*.

Primjer 53:

SELECT ime FROM osoba

UNION

SELECT ime FROM osoba

UNION

SELECT 'Ante'

ime

Ante

Ivan

Marko

Pero

(4 row(s) affected)

I u zadnjem primjeru vidimo koji su to zajednički elementi, ali ne i koliko se puta ponavljaju.

Primjer 54:

SELECT ime FROM osoba

INTERSECT

SELECT ime FROM osoba

IME

Ivan

Pero

Marko

11. NAREDBA UPDATE

Svrha ove ključne riječi je promjena vrijednosti u postojećem redu ili redovima. Prvo ću pokazati kako promijeniti vrijednosti čitavog stupca.
Za primjer ćemo staviti u stupac napomena tablice posudi staviti 'vraćen ispravan'.
Primjer 55:

UPDATE posudi SET napomena ='vraćen ispravan'

Rezultat našeg rada vidimo kroz:
Primjer 56:

SELECT osoba, mobitel, napomena FROM posudi

```
osoba              mobitel             napomena
-------------- -------------- ------------------------
0102968383911   no3310         vracen ispravan
0302982383818    soz7          vracen ispravan
0102968383911   no3310         vracen ispravan
1212972383944   mov60          vracen ispravan
0102968383911   no3330         vracen ispravan

(5 row(s) affected)
```

A ako koristimo uvjet možemo promijeniti samo određeni broj kolona npr. dodati u ulicu napomenu da *'nema podatka'* za čovjeka koji nema adresu u tablici osoba.

Primjer 57:

UPDATE osoba SET ulica ='Nema podatka'
WHERE ulica ='' OR ulica IS NULL

Jednostavno sad možemo pregledati koje su to osobe.
Primjer 58:

*SELECT * FROM osoba WHERE ulica ='Nema podatka'*

maticni	ime	prezime	ulica	mjesto
0302982383818	Ivan	Ivic	Nema podatka	Split
1717985383542	Ivan	Pos	Nema podatka	Zagreb

(2 row(s) affected)

Pri radu s ovom naredbom moramo znati da kolonu osnovnog ključa ne možemo mijenjati ako se taj ključ spominje u vezanoj tablici mobitel.

12. NAREDBA DELETE

Kad su podaci nepotrebni moramo ih izbrisati. Oni koji su navikli raditi u Windows OS trebali bi pripaziti jer neće biti nikakvih dodatnih pitanja tipa jeste li sigurni? Općenito kad sam radio u editorima ta je naredba uvijek stajala kao komentar – *DELETE FROM WHERE..* da je ne bi izvršio slučajno. Pa bi preporučio da se prvo uvjet stavi u *SELECT* tako da vidimo što ćemo obrisati ili da radimo sa *BEGIN TRANSACTION, COMMIT* i *ROLLBACK*. Transakcija označava jednu cjelinu posla čiji početak označavamo sa *BEGIN TRANSACTION* i kad završimo tu cjelinu provjerimo da li je sve u redu, ako je u redu izvršimo *COMMIT* i promjene koje smo napravili na bazi postaju važeće ili ako nešto nije onako kako smo zamislili izvršimo *ROLLBACK* i svi podaci su isti kao prije izvršenja početka transakcije.

Naredba *DELETE* ne može obrisati vrijednost iz jednog polja, već uklanja cijeli red, moramo paziti da nam se ne javi problem zbog relacije osnovni/strani ključ i na kraju ova naredba briše samo redove ne i tablicu.

Primjer brisanja tablice osoba koja je vezana na tablicu posudi.

Primjer 59:

DELETE FROM osoba

Server: Msg 547, Level 16, State 1, Line 1
DELETE statement conflicted with COLUMN REFERENCE
constraint 'FK__posudi__osoba__3B75D760'. The conflict occurred in database 'mobitel', table 'posudi', column 'osoba'.
The statement has been terminated.

Ovaj problem rješavamo tako da prvo obrišemo tablicu posudi u kojoj postoji ključ koji je vezan na tablicu osoba. Znači čitavu tablicu brišemo tako da navedemo samo ime tablice.
Primjer 60:

DELETE FROM posudi;
DELETE FROM osoba;
*SELECT * FROM osoba;*

(5 row(s) affected)
(5 row(s) affected)

maticni	ime	prezime	ulica	mjesto

(0 row(s) affected)

Kao i dosad koristeći uvjet *WHERE* možemo obrisati samo neke redove u tablicama npr. sve osobe koje su iz Zadra.
Primjer 61:

*SELECT * FROM osoba WHERE mjesto = 'Zadar';*
DELETE FROM osoba WHERE mjesto = 'Zadar';
*SELECT * FROM osoba WHERE mjesto = 'Zadar';*

maticni	ime	prezime	ulica	mjesto
0305972383915	Marko	Maric	Lavova 67	Zadar

(1 row(s) affected)
(1 row(s) affected)

maticni	ime	prezime	ulica	mjesto

0 row(s) affected)

13. NAREDBA INSERT

Ključna riječ *INSERT* unosi podatke u tablicu red po red. Vrijednosti koje unosimo moraju biti istog tipa podataka kao polja u koja unosimo podatke, veličina podatka mora odgovarati veličini kolone (ne možemo unijeti 20 znakova u polje koje je nekog tipa duljine 10 znakova i podaci u *VALUES* moraju odgovarati stupcima u listi stupaca. Za primjere ćemo uzeti kod iz dodatka za kreiranje i popunjavanje tablica.
Primjer 62:

```
INSERT INTO osoba
        VALUES ('1717985383542', 'Ivan', 'Pos', NULL, 'Zagreb');
```

ili isti kod napisan na drugi način
Primjer 63:

```
INSERT INTO osoba (maticni, ime, prezime, ulica, mjesto)
        VALUES ('1717985383542', 'Ivan', 'Pos', NULL, 'Zagreb');
```

To su dvije osnovne varijante. Prva da ne navedem imena stupaca u koje stavljamo podatke, ali se vrijednosti iza *VALUES* moraju poklapati sa tipom podatka i brojem stupaca. Znači prvi podatak ide u prvi stupac i tako dalje. U slučaju da imamo manje podataka nego stupaca u tablici dobivamo poruku o grešci :

```
Insert Error: Column name or number of supplied values
        does not match table definition.
```

U drugoj varijanti ne trebamo unositi sve stupce već samo one koje su navedeni iza imena tablice, no tu trebamo voditi računa da polje koje je *NOT NULL* mora biti upisano. Polje koje ima karakteristiku *IDENTITY* može nam praviti probleme pri unosu, a u polju *UNIQUE* treba pripaziti da dva puta ne unosimo istu vrijednost. Primjer je jako sličan gornjem drugom primjeru samo treba uočiti da ne postoji stupac ulica.
Primjer 64:

```
INSERT INTO osoba (maticni, ime, prezime, mjesto)
        VALUES ('1717985383543', 'Ivan', 'Pos', 'Zagreb');
```

Postoji još jedna vrsta, a to je *INSERT...SELECT* čije značenje je jasno iz prijašnjih objašnjenja. Unosimo podatke koje dobivamo uz pomoć *SELECT-a.* Vidjeti ćemo to na malo vjerojatnom primjeru da u tablicu osoba moramo unijeti vrijednosti iz tablice mobitel jer moramo imati samo jednu tablicu za osobu i mobitel. Sad nam sifra iz mobitela prelazi u maticni koji je također osnovni ključ, u stupce ime i prezime stavljamo proizvodi i model, dok u znakovna polja ulica i mjesto stavljamo tezinu i visinu odnosno sirinu i debljinu koje pretvaramo u tip *NVARCHAR* jer bi inače izgubili vrijednosti ta dva parametra koja bi završila u jednom stupcu zbrojena što nam nikako ne odgovara.
Primjer 65:

```
INSERT INTO osoba
    SELECT sifra, proizvodi, model,
            CAST(tezina AS  NVARCHAR) + '-' +
            CAST(visina AS NVARCHAR),
            CAST(sirina AS NVARCHAR) + '-' +
            CAST(debljina AS NVARCHAR) FROM mobitel
```

maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Peric	Gajeva 3	Zagreb
0302982383818	Ivan	Ivic		Split
0305972383915	Marko	Maric	Lavova 67	Zadar
1212972383944	Ivan	Dundov	Obala 4	Zagreb
1717985383542	Ivan	Pos	NULL	Zagreb
mov60	motorola	v60	109-85	45-25
no3310	nokia	3310	133-113	48-22
no3330	nokia	3330	133-113	48-22
soj70	sony	j70	92-133	45-22
soz7	sony	z7	95-91	50-25

(10 row(s) affected)

ovo je izlaz koji bi dobili da nismo koristili *CAST*

Primjer 66:

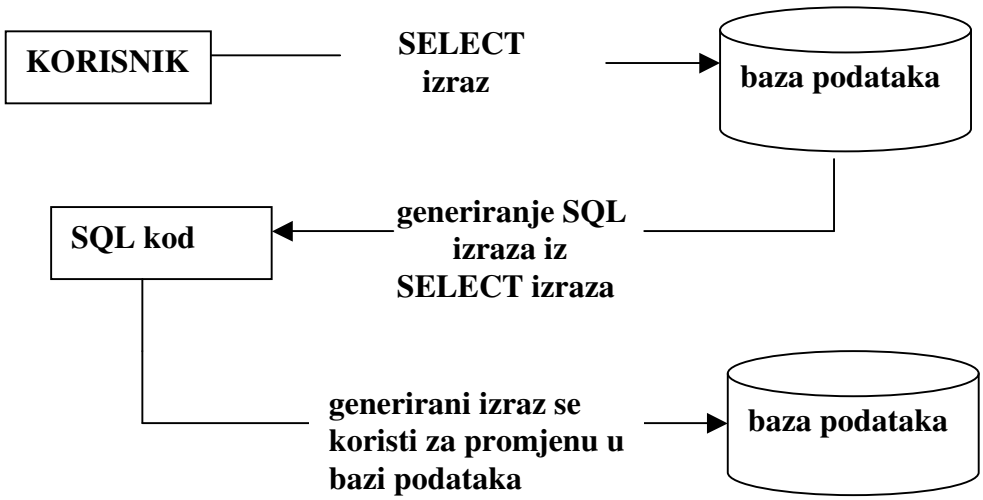
maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Peric	Gajeva 3	Zagreb
0302982383818	Ivan	Ivic		Split
0305972383915	Marko	Maric	Lavova 67	Zadar
1212972383944	Ivan	Dundov	Obala 4	Zagreb
1717985383542	Ivan	Pos	NULL	Zagreb
mov60	motorola	v60	194	70
no3310	nokia	3310	246	70
no3330	nokia	3330	246	70
soj70	sony	j70	225	67
soz7	sony	z7	186	75

(10 row(s) affected)

Nove redove moramo obrisati, a to radimo na sljedeći način:
Primjer 67:

DELETE FROM osoba WHERE maticni LIKE '%o%'

Jedan od načina učitavanja podataka među različitim bazama je i SQL koji piše SQL. Napraviti ćemo *SELECT* koji će nam dati kao izlaz SQL naredbu *INSERT* koju možemo postupkom kopiranja i lijepljenja staviti u dio prozora za izvršavanje koda te je izvršiti. Sve će biti malo jasnije na slici i primjeru.



Slika : Proces generiranja SQL-a iz baze podataka
Primjer 68:

SELECT "INSERT INTO osoba VALUES

('" + maticni + " ','" + ime + "','" + prezime

+ "','" + ulica +" ','" + mjesto +" ')" FROM osoba

```
---------------------------------------------------------------------------------------------------
INSERT INTO osoba VALUES ('0102968383911 ',' Pero','Peric','Gajeva 3 ','Zagreb ')
INSERT INTO osoba VALUES ('0302982383818 ',' Ivan','Ivic',' ','Split ')
INSERT INTO osoba VALUES
        ('0305972383915 ',' Marko','Maric','Lavova 67 ','Zadar ')
INSERT INTO osoba VALUES
        ('1212972383944 ',' Ivan','Dundov','Obala 4 ','Zagreb ')
NULL

(5 row(s) affected)
```

Dobiveni dio trebamo samo kopirati i izvršiti, ali ako bolje pogledamo vidimo da tu nemamo peti red. Uzrok je to što vrijednost *NULL* zbrojena sa bilo čim daje izlaz *NULL.* To dobro vidimo u trećem redu slijedećeg primjera gdje *'Ovo je adresa : ' + NULL* daje *NULL.*

Primjer 69:

```
SELECT 'Ovo je adresa : ' + ulica  'Zbrajam 2 znakovna  niza'
        FROM osoba WHERE ime ='Ivan'
```

```
Zbrajam 2 znakovna  niza
----------------------------------------
Ovo je adresa :
Ovo je adresa : Obala 4
NULL

(3 row(s) affected)
```

Taj problem rješavamo koristeći funkciju *ISNULL.*

Primjer 70:

SELECT 'Ovo je adresa : ' + ISNULL(ulica,"")
FROM osoba WHERE ime ='Ivan'

Ovo je adresa :
Ovo je adresa : Obala 4
Ovo je adresa :

(3 row(s) affected)

U dodatku se nalazi procedura za izvoz podataka MS SQL koju je napisao Alexander Chigrik[7] koju sam malo promijenio zbog nedostatka koji je objašnjen gore. Ona se izvršava tako da u navodnicima upišemo naziv tablice koju želimo dobiti i pozovemo proceduru:
Primjer 71:

EXEC DataAsInsCommand 'osoba';

kao rezultat daje:

```
SELECT REPLACE ('INSERT INTO osoba VALUES (' + '''' +
    ISNULL(maticni,"null") + '''' + ',' + '''' + ISNULL(ime,"null") + '''' + ',' + '''' +
    ISNULL(prezime,"null") + '''' + ',' + '''' +ISNULL(ulica,"null") + '''' + ',' + '''' +
    ISNULL(mjesto,"null") + ''''+ ")", ''"null"'' , 'null') FROM a
```

```
INSERT INTO osoba
        VALUES ("0102968383911","Pero","Peric","Gajeva 3","Zagreb")
INSERT INTO osoba
        VALUES ("0302982383818","Ivan","Ivic","","Split")
INSERT INTO osoba
        VALUES ("0305972383915","Marko","Maric","Lavova 67","Zadar")
INSERT INTO osoba
        VALUES ("1212972383944","Ivan","Dundov","Obala 4","Zagreb")
INSERT INTO osoba
        VALUES ("1717985383542","Ivan","Pos",null,"Zagreb")
----
GO
```

14. PODUPITI

Podupit je izraz koji svoj rezultat prenosi kao argument drugom izrazu što znači da u kodu imamo zapravo više zasebnih linija ispravnog koda. Pokazati ću to kroz primjer ispisa imena i prezimena svih osoba koje imamo u tablici osoba, a nemamo ih u tablici posudi jer nam samo zauzimaju prostor zbog toga što nikad nisu posudili niti jedan mobitel te ih možemo slobodno izbrisati jer ih kasnije lako u slučaju potrebe možemo ponovo dodati. Rekao sam da je to više naredbi pa ću prvo pokazati te dvije naredbe i njihove rezultate odvojeno.
Primjer 72:

SELECT ime, prezime FROM osoba

```
ime             prezime
--------------- ---------------
Pero            Peric
Ivan            Ivic
Marko           Maric
Ivan            Dundov
Ivan            Pos

(5 row(s) affected)
```

SELECT osoba FROM posudi

```
osoba
---------------
0102968383911
0302982383818
0102968383911
```

1212972383944
0102968383911

(5 row(s) affected)

Sad bi trebalo ta dva različita koda spojiti u jedan uz uvjet da dobijemo vani samo ona imena i prezimena koja nisu u (*NOT IN*) tablici posudi.
Primjer 73:

```
SELECT  ime, prezime FROM osoba
        WHERE maticni NOT IN (SELECT osoba FROM posudi)
```

```
ime             prezime
--------------- ---------------
Marko           Maric
Ivan            Pos

(2 row(s) affected)
```

Također možemo podupite izvršiti na istoj tablici. Dobar primjer za to je kad bi željeli izdvojiti samo zadnja posuđivanja svake osobe. U ovom slučaju moramo zamijetiti da zadnji datum posudbe za svaku osobu posebno karakterizira najveća vrijednost u polju broj pa možemo te vrijednosti izdvojiti i napraviti normalan *SELECT* koji će gledati da vrijednost njegovog polja broja bude u skupu koji je napravio drugi izraz.
Primjer 74:

```
SELECT * FROM posudi WHERE broj IN
        (SELECT MAX(broj) FROM posudi GROUP BY osoba)
```

broj	osoba	mobitel	uzeo	vratio	napomena
2	0302982383818	soz7	2002-01-15 00:00:00.000	2002-02-02	
4	1212972383944	mov60	2002-02-15 00:00:00.000	2002-07-15	
5	0102968383911	no3330	2002-06-01 00:00:00.000	2002-10-01	

(3 row(s) affected)

15. SPAJANJA

Podaci koje spremamo u bazu se obično nalaze u više tablica jer bi spremanje sveg potrebnog u jednu tablicu vrlo brzo popunilo prostor predviđen za podatke. Iz toga logično slijedi da moramo imati nekakva spajanja tablica da bi dobili određene podatke koji se inače nalaze u različitim tablicama . Dobar primjer za to je tablica posudi u kojoj se nalazi samo matični broj osobe koja je posudila mobitel, a mi ne znamo koja je to osoba ako ne pogledamo u tablicu osoba. Spajanje se može izvršiti na dva načina *FROM / WHERE* ili *JOIN* . Možda malo čudno ali tablice osim što spajamo sa drugim možemo spojiti i sa samom sobom (self-join) samo-spajanje. Obično se to koristi u složenijim upitima, a koristimo sinonime za imena tablica(*FROM mobitel tablica1, mobitel tablica2*) da bi mogli točno odrediti što se odnosi na koju tablicu. U idućem primjeri sinonimi za tablicu mobitel su tablica1 i tablica2.
Primjer 75:

```
SELECT tablica1.proizvodi + ' ' +
        tablica1.model 'mobitel', tablica1.tezina 'težina',
        tablica2.proizvodi + ' ' +
        tablica2.model 'lakši ili jednak', tablica2.tezina 'težina'
  FROM mobitel tablica1, mobitel tablica2
 ORDER BY 2
```

mobitel	težina	lakši ili jednak	težina
sony j70	92	motorola v60	109
sony j70	92	nokia 3310	133
sony j70	92	nokia 3330	133
sony j70	92	sony j70	92
sony j70	92	sony z7	95
sony z7	95	motorola v60	109
sony z7	95	nokia 3310	133
sony z7	95	nokia 3330	133
sony z7	95	sony j70	92
sony z7	95	sony z7	95
motorola v60	109	motorola v60	109
motorola v60	109	nokia 3310	133
motorola v60	109	nokia 3330	133
motorola v60	109	sony j70	92
motorola v60	109	sony z7	95
nokia 3310	133	motorola v60	109
nokia 3310	133	nokia 3310	133
nokia 3310	133	nokia 3330	133
nokia 3310	133	sony j70	92
nokia 3310	133	sony z7	95
nokia 3330	133	motorola v60	109
nokia 3330	133	nokia 3310	133
nokia 3330	133	nokia 3330	133
nokia 3330	133	sony j70	92
nokia 3330	133	sony z7	95

(25 row(s) affected)

U ovom samo-spajanju tablice mobitel dobili smo 25 redova. Rezultat je poznat pod imenom Kartezijev produkt što znači da smo svakom redu prve tablice pridružili sve do jedan red druge tablice pa budući da tablica

ima 5 redova dobili smo 5 * 5 redova na izlazu. U slučaju da se spajaju različite tablice Kartezijev produkt bi iznosio broj redova prve tablice * broj redova druge tablice. Na prvi pogled možda beskorisno, ali uz uvjet *WHERE* možemo dobiti u prvom stupcu rezultata sve mobitele, a u trećem sve one koji su lakši ili jednaki (*WHERE tablica1.tezina >= tablica2.tezina*). Tu koristimo izraz veće ili jednako zbog toga što ne bi bez jednako dobili najlakši mobitel jer nema lakšeg od njega.
Primjer 76:

```
SELECT tablica1.proizvodi  + ' ' +
        tablica1.model 'mobitel', tablica1.tezina 'težina',
       tablica2.proizvodi  + ' ' +
       tablica2.model 'lakši ili jednak', tablica2.tezina 'težina'
  FROM  mobitel tablica1, mobitel tablica2
 WHERE  tablica1.tezina >= tablica2.tezina
 ORDER BY 2
```

mobitel	težina	lakši ili jednak	težina
sony j70	92	sony j70	92
sony z7	95	sony j70	92
sony z7	95	sony z7	95
motorola v60	109	motorola v60	109
motorola v60	109	sony j70	92
motorola v60	109	sony z7	95
nokia 3310	133	motorola v60	109
nokia 3310	133	nokia 3310	133
nokia 3310	133	nokia 3330	133
nokia 3310	133	sony j70	92
nokia 3310	133	sony z7	95
nokia 3330	133	motorola v60	109

nokia 3330	133	nokia 3310	133
nokia 3330	133	nokia 3330	133
nokia 3330	133	sony j70	92
nokia 3330	133	sony z7	95

(16 row(s) affected)

Napredniji primjer bi bio problem numeriranja zadanog izlaza. Uz pomoć uspoređivanja znakovnih podataka (*WHERE tablica1.prezime >= tablica2.prezime*) možemo prebrojiti koliko ima osoba ispred te ispisati taj broj (*SELECT COUNT(*) 'Redni broj'*). Prvi izlaz je Dundov, a veći ili jednak je onda samo taj isti podatak što znači da ih ukupno imamo jedan te ispišemo taj broj, drugi izlaz je Ivić od kojeg su veći ili jednaki Dundov i Ivić znači imamo ih 2 te ispišemo i taj broj pod nazivom Redni broj i tako dalje.
Primjer 77:

```
SELECT COUNT(*) 'Redni broj',tablica1.prezime, tablica1.ime
     FROM osoba tablica1, osoba tablica2
     WHERE tablica1.prezime >= tablica2.prezime
     GROUP BY tablica1.prezime, tablica1.ime
     ORDER BY 1
```

Redni broj	prezime	ime
1	Dundov	Ivan
2	Ivic	Ivan
3	Maric	Marko
4	Peric	Pero
5	Pos	Ivan

(5 row(s) affected)

Za ovaj kod vodimo računa o tome da bi vrijednost u uvjetu trebala imati karakteristiku da je to jedinstven podatak.
Za primjer *JOIN* naredbe raditi ćemo sa privremenom tablicom tako da možemo upisati nove podatke a da na postojećim ne mijenjamo ništa i jer možemo zaobići ograničenja osnovni/primarni ključ.
Primjer 78:

```
SELECT * INTO #proba FROM posudi
INSERT INTO #proba
    VALUES('4102968383911', '?', '', '', '?');
INSERT INTO #proba
    VALUES(NULL, '??', '', '', '??');
```

Prvo ću obraditi *INNER JOIN* . Kad dobijemo Kartezijev produkt ovo spajanje daje redove čiji podaci se poklapaju u zadanim stupcima tj. one koji imaju partnera u drugoj tablici ili po spajanju uvjetima to bi odgovaralo (*WHERE osoba.maticni = #proba.osoba*)
Primjer 79:

```
SELECT osoba.ime, osoba.prezime, #proba.mobitel
      FROM osoba
      INNER JOIN  #proba
      ON osoba.maticni = #proba.osoba
```

```
ime           prezime        mobitel
------------- -------------- --------------
Pero          Peric          no3310
Ivan          Ivic           soz7
Pero          Peric          no3310
```

```
Ivan        Dundov      mov60
Pero        Peric       no3330

(5 row(s) affected)
```

Iduća spajanja su *OUTER* i mogu biti *FULL, LEFT* ili *RIGHT*. Budući da neki ljudi nisu nikad posudili mobitel nemaju par u tablici posudi te u prethodnom primjeru nisu prikazani, ali ne bi bilo loše da možemo i njih vidjeti. Upravo to radimo sa (*LEFT OUTER JOIN*) koji kaže da prikažemo i one koji se nalaze u lijevoj tablici, a nemaju svog para u drugoj tablici. U ovom slučaju to su Marko Marić i Ivan Pos koji nisu posudili nikada mobitel pa umjesto šifre mobitela imamo *NULL.*
Primjer 80:

```
SELECT osoba.ime, osoba.prezime, #proba.mobitel
        FROM osoba
        LEFT OUTER JOIN  #proba
        ON osoba.maticni = #proba.osoba
```

```
ime         prezime      mobitel
----------- ------------ ----------
Pero        Peric        no3310
Pero        Peric        no3310
Pero        Peric        no3330
Ivan        Ivic         soz7
Marko       Maric        NULL
Ivan        Dundov       mov60
Ivan        Pos          NULL

(7 row(s) affected)
```

Suprotan slučaj je *RIGHT OUTER JOIN* koji nam daje podatke iz desne tablice koji nemaju para u lijevoj tablici. Znači imamo mobitele ? i ?? koje su posudile osobe kojih nema u tablici osoba te u izlazu umjesto imena i prezimena koji se nalaze u toj tablici imamo *NULL.*
Primjer 81:

```
SELECT osoba.ime, osoba.prezime, #proba.mobitel
        FROM osoba
        RIGHT OUTER JOIN  #proba
        ON osoba.maticni = #proba.osoba
```

```
ime             prezime         mobitel
--------------- --------------- ---------------
NULL            NULL             ?
NULL            NULL             ??
Pero            Peric           no3310
Ivan            Ivic            soz7
Pero            Peric           no3310
Ivan            Dundov          mov60
Pero            Peric           no3330

(7 row(s) affected)
```

Na kraju sasvim logički slijedi *FULL OUTER JOIN* koji predstavlja spojena prijašnja dva spajanja.
Primjer 82:

```
SELECT osoba.ime, osoba.prezime, #proba.mobitel
        FROM osoba
        FULL OUTER JOIN  #proba
        ON osoba.maticni = #proba.osoba
```

```
ime            prezime        mobitel
-------------- -------------- --------------
NULL           NULL             ??
Pero           Peric          no3310
Pero           Peric          no3310
Pero           Peric          no3330
Ivan           Ivic           soz7
Marko          Maric          NULL
Ivan           Dundov         mov60
Ivan           Pos            NULL
NULL           NULL             ?

(9 row(s) affected)
```

16. POGLEDI

Pogledi (*views*) su kao i tablice sastavljeni od polja, redova, naziva kolona. Ipak za razliku od tablica oni ne sadrže podatke. Pogledi se zasnivaju na tablicama te su i slične ključne riječi kojima ih stvaramo (*CREATE VIEW*) i uništavamo (*DROP VIEW*). Sadržaj pogleda osvježava se u trenutku izvođenja upita na osnovama sadržaja objekta nad kojim je pogled definiran pa se svaka promjena odmah vidi u pogledu koji je vezan sa tablicom na kojoj se dogodila promjena. Obično ih koristimo zbog jedne od ovih dvije funkcije. Prva je da bi zaštitili podatke koji nisu da ih svatko gleda. Npr. možemo iz tablice osoba maknuti stupce maticni i ulicu tako da osiguramo neku privatnost tim osobama, a onda svim zaposlenim možemo omogućiti da vide taj pogled dok samo ovlašteni mogu gledati u tablicu.
Primjer 83:

```
CREATE VIEW osobaPogled (ime, prezime, mjesto) AS
        SELECT ime, prezime, mjesto FROM osoba
```

The command(s) completed successfully.

```
SELECT * FROM osobaPogled;
```

ime	prezime	mjesto
Pero	Peric	Zagreb
Ivan	Ivic	Split
Marko	Maric	Zadar
Ivan	Dundov	Zagreb

Ivan Pos Zagreb

(5 row(s) affected)

DROP VIEW osobaPogled;

The command(s) completed successfully.

Drugi razlog korištenja pogleda je da nam podaci budu spremljeni na pregled u pogodnom obliku. U tablici posudi osoba i mobitel su navedeni kroz matični broj odnosnu neku šifru mobitela što je nečitljivo, ali ne znači da je netočno jer organizacija podataka smislena za ljude ne mora biti najpogodnija za spremanje podataka u bazu. Primjer je znači tablica posudi s tim da piše ime i prezime osobe koja je posudila mobitel te model i proizvođač mobitela.
Primjer 84:

```
CREATE VIEW posudiPogled ( naziv, mobitel, uzeo, vratio) AS
        SELECT osoba.ime + ' ' + osoba.prezime,
        mobitel.proizvodi + ' ' + mobitel.model,
        SUBSTRING(CAST(posudi.uzeo AS VARCHAR), 1, 11),
        SUBSTRING(CAST(posudi.vratio AS VARCHAR), 1, 11)
   FROM osoba, mobitel, posudi
   WHERE osoba.maticni = posudi.osoba
         AND mobitel.sifra = posudi.mobitel
```

The command(s) completed successfully.

*SELECT * FROM posudiPogled*

naziv	mobitel	uzeo	vratio
Pero Peric	nokia 3310	sij 1 2002	sij 30 2002
Ivan Ivic	sony z7	sij 15 2002	vel 2 2002
Pero Peric	nokia 3310	ožu 3 2002	svi 15 2002
Ivan Dundov	motorola v60	vel 15 2002	srp 15 2002
Pero Peric	nokia 3330	lip 1 2002	lis 1 2002

(5 row(s) affected)

DROP VIEW posudiPogled

The command(s) completed successfully.

Zbog svojih specifičnosti pogledi imaju neka ograničenja :

- ne možemo upotrijebiti DELETE izraz nad pogledima koji su kreirani nad više tablica
- ne može se upotrijebiti INSERT iskaz ukoliko svi stupci koji imaju karakteristiku NOT NULL u tablici nisu u pogledu
- ukoliko unosimo ili mijenjamo podatke kroz pogled spajanja, svi slogovi moraju pripadati istoj tablici
- ukoliko koristimo DISTINCT prilikom kreiranja pogleda ne možemo mijenjati ili unositi redove u pogled
- ne možemo mijenjati stupac koji je rezultat izraza ili funkcije

17. PRIVREMENE TABLICE

Privremene tablice su specijalna vrsta tablica. One postoje samo za vrijeme dok smo prijavljeni na server. U SAP-u se takve tablice prepoznaju po vlasniku *TEMP.* ispred imena tablice dok u MS-u moramo dodati znak # (ljestve ispred imena tablice). Kreiramo i uništavamo ih standardnim naredbama *CREATE TABLE*, *DROP TABLE.* Međutim postoji i njihovo automatsko uništavanje kad se odjavimo ili prekinemo pristup bazi podataka. Za razliku od pogleda naknadne promjene u nekakvim temeljnim tablicama neće se odraziti u ovim tablicama. Koristimo ih za pohranjivanje rezultata nekakvih kompliciranih izraza, a te rezultate mislimo koristiti u kasnijim izrazima upita ili kad je potrebno nešto napraviti u više odvojenih koraka. Tipičan primjer je kreiranje tablice iste kao neka postojeća u našem slučaju tablica osoba. Te sad možemo na njoj obavljati različite upite, brisanja, računanja i na kraju usporediti s nečim drugim.

Primjer 85:

```
CREATE TABLE #privremenaOsoba
(
   maticni NVARCHAR(15),
   ime NVARCHAR(15) NOT NULL,
   prezime NVARCHAR(15) NOT NULL,
   ulica NVARCHAR(25),
   mjesto NVARCHAR(15) DEFAULT 'Zagreb'
 );
```

The command(s) completed successfully.

Vidimo da pri *SELECT-u* pokazuje sve karakteristike standardnih tablica. Budući da je nismo punili podacima naša privremena tablica (*#privremenaOsoba*) je prazna.
Primjer 86:

*SELECT * FROM #privremenaOsoba*

```
maticni      ime          prezime      ulica                  mjesto
------------ ------------ ------------ ---------------------- ----------

(0 row(s) affected)
```

Ako bi otvorili novi prozor za pisanje naredbi i pokušali pristupiti našoj tablici vidjeli bi da je takva tablica dostupna samo sesiji u kojoj je kreirana jer bi dobili grešku tj. kad bi vratili fokus na prethodni prozor opet bi normalno vidjeli tablicu i mogli bi raditi na njoj.
Primjer 87:

*SELECT * FROM #privremenaOsoba*

```
Server: Msg 208, Level 16, State 1, Line 1
Invalid object name '#privremenaOsoba'.
```

Standardan način uništavanja tablice.

DROP TABLE #privremenaOsoba

```
The command(s) completed successfully.
```

Gornji primjer je kreirao privremenu tablicu istovjetnu kao i postojeća tablica osoba pa u drugom primjeru pokazujem jednostavniji način da se

napravi isto, a ujedno je i popunimo sa podacima iz te tablice što vidimo po odgovoru servera koji kaže da se rezultat odnosi na pet redova (ukupan broj redova u tablici osoba). Ali u privremenoj tablici više ne vrijede definicije ključeva, sad bi nju mogli popuniti sa jednim korisnikom koji ima matični broj isti kao neki koji je upisan.
Primjer 88:

*SELECT * INTO #privremenaOsoba FROM osoba*
(5 row(s) affected)

*SELECT * FROM #privremenaOsoba*

maticni	ime	prezime	ulica	mjesto
0102968383911	Pero	Peric	Gajeva 3	Zagreb
0302982383818	Ivan	Ivic		Split
0305972383915	Marko	Maric	Lavova 67	Zadar
1212972383944	Ivan	Dundov	Obala 4	Zagreb
1717985383542	Ivan	Pos	NULL	Zagreb

(5 row(s) affected)

Kao što sam naveo ranije za SAP se sintaksa malo razlikuje što je očito iz idućeg primjera čija svrha je ista kao u gornjem primjeru (kreiranje i popunjavanje).
Primjer 89:

*CREATE TABLE TEMP.privremenaOsoba AS SELECT * FROM osoba*
*SELECT * FROM TEMP.privremenaOsoba*
DROP TABLE TEMP.privremenaOsoba

18. UVOZ – IZVOZ PODATAKA

Uvoz ili izvoz podataka radimo da bi prebacili podatke iz sustava koji nisu povezani, a mogu biti istih ili različitih baza. U MS SQL-u to radimo pomoću DTS Import/Export Wizard-a (Data Transformation Services). To je program *dstwiz* koji možemo pokretati i sa komandne linije gdje sa određenim parametrima određujemo njegove funkcije, ipak to je grafičko okruženje i vrlo je intuitivno tako da neću objašnjavati već ću pokazati najčešće oblike izvezenih podataka u ovom slučaju tablice osoba. Uočavamo da postoje graničnici koji ograđuju podatke. To su znakovi "," zarez ili "|" okomita crta. Jasno je da se znak koji ograničava podatak ne smije pojaviti u samom podatku.

"0102968383911","Pero","Peric","Gajeva 3","Zagreb"
"0302982383818","Ivan","Ivic","","Split"
"0305972383915","Marko","Maric","Lavova 67","Zadar"
"1212972383944","Ivan","Dundov","Obala 4","Zagreb"
"1717985383542","Ivan","Pos","","Zagreb"

ili

0102968383911|Pero|Peric|Gajeva 3|Zagreb
0302982383818|Ivan|Ivic||Split
0305972383915|Marko|Maric|Lavova 67|Zadar
1212972383944|Ivan|Dundov|Obala 4|Zagreb
1717985383542|Ivan|Pos||Zagreb

ili

0102968383911,Pero,Peric,Gajeva 3,Zagreb
0302982383818,Ivan,Ivic,,Split
0305972383915,Marko,Maric,Lavova 67,Zadar
1212972383944,Ivan,Dundov,Obala 4,Zagreb
1717985383542,Ivan,Pos,,Zagreb

Za SAP DB moramo takve stvari raditi sa komandne linije uz pomoć programa *repmcli* . Pa ću pokazati i jedan takav primjer.

E:\>*repmcli -d TST1 -u DBA,DBA -b c:\export1.txt*

Opened connection to REPLICATION SERVER at node local host.

REPLICATION SERVER Log File:

'E:\Program Files\sapdb\indep_data\wrk\repserver.log'

User DBA connected to database TST1 on local host.

DATAEXTRACT * FROM dba.osoba

  OUTFILE 'osoba.txt'

Successfully executed

Tražili smo iza prijave na bazu izvršenje naredbi u *c:\export1.txt* koje glase:

DATAEXTRACT * FROM dba.osoba

  OUTFILE 'osoba.txt'

Tu smo dobili podatke u datoteci osoba.txt koji izgledaju kao što je navedeno u prvom gornjem obliku. Podaci su ograničeni navodnicama i razdvojeni zarezom. Kad bi htjeli učitati te podatke morali bi prvo obrisati tablicu osoba jer ne možemo imati dva ista osnovna ključa u jednoj tablici, a to radim naredbom :

DELETE FROM osoba

Server: Msg 547, Level 16, State 1, Line 1

DELETE statement conflicted with TABLE REFERENCE constraint 'FK__posudi__0697FACD'. The conflict occurred in database 'mobitel', table 'posudi'.

The statement has been terminated.

koja javlja grešku. Greška je nastala zbog relacije osnovni/strani ključ u tablicama osoba/posudi. Tablica posudi ima podatke koji se odnose na

tablicu osoba. Znači prvo moramo obrisati tablicu posudi i onda tek osoba. Kad smo to napravili izvršimo u DOS prozoru:

E:\>*repmcli -d TST1 -u DBA,DBA -b c:\naredba.txt*

```
Opened connection to REPLICATION SERVER at node local host.
REPLICATION SERVER Log File:
'E:\Program Files\sapdb\indep_data\wrk\repserver.log'
User DBA connected to database TST1 on local host.
DATALOAD TABLE "OSOBA"
"MATICNI" 1 CHAR
"IME" 2 CHAR
"PREZIME" 3 CHAR
"ULICA" 4 CHAR DEFAULT NULL
"MJESTO" 5 CHAR DEFAULT NULL
INFILE 'osoba.txt' COMPRESSED ASCII
DECIMAL '/ /./'
DATE INTERNAL
TIME INTERNAL
TIMESTAMP INTERNAL
NULL '?                '
BOOLEAN 'TRUE/FALSE'
Successfully executed
```

Datoteka *c:\naredba.txt* izgleda ovako:

```
DATALOAD TABLE "OSOBA"
"MATICNI" 1 CHAR
"IME" 2 CHAR
"PREZIME" 3 CHAR
"ULICA" 4 CHAR DEFAULT NULL
"MJESTO" 5 CHAR DEFAULT NULL
```

INFILE 'osoba.txt' COMPRESSED ASCII

DECIMAL '/ /./'

DATE INTERNAL

TIME INTERNAL

TIMESTAMP INTERNAL

NULL '? '

BOOLEAN 'TRUE/FALSE'

19. PROCEDURE

Procedura je upit koji se čuva u bazi podataka. Pisanje započinjemo sa *CREATE PROCEDURE imeProcedure AS programskiKod,* a pozivamo je sa *EXEC imeProcedure.* Može biti bez ili sa ulaznim parametrima kao što ćemo vidjeti u slijedećim primjerima. Prednosti su da smanjuje promet kroz mrežu jer se upit ne šalje mrežom već je uskladišten na poslužitelju, pošto se nalaze na poslužitelju prošle su postupak prevođenja pa za njih postoji u memoriji plan izvršavanja te se brže izvršavaju za razliku od upita koji se svaki put prvo prevodi (poslužitelj gleda ima li u upitu spajanja tablica itd.) i na kraju budući da je upit na poslužitelju u slučaju potrebe za nekakvom izmjenom mijenjamo ga samo na jednom mjestu. U idućem primjeru koristimo proceduru bez ulaznih parametara, a kao rezultat dobivamo prazna mjesta u tablici posudi, odnosno ona koja je neko izbrisao. To vidimo kroz stupac broj koji se automatski povećava za jedan i dodjeljuje idućem unosu sa tim da se praznine ne dopunjavaju te imamo prazninu ako nam je neko pokušao izbrisati neki unos. Radimo sa transakcijama tako da na tablici posudi na kraju ništa nije promijenjeno.

Primjer 90:

```
CREATE PROCEDURE rupica AS
    SELECT posudi.broj + 1  "početak",
        (SELECT MIN (broj - 1) FROM posudi  tablica
              WHERE tablica.broj > posudi.broj AND NOT broj-1
              IN (SELECT broj FROM posudi)) "kraj rupe"
    FROM posudi
    WHERE NOT posudi.broj + 1 IN (SELECT broj FROM posudi)
```

BEGIN TRANSACTION

DELETE FROM posudi WHERE broj=2

DELETE FROM posudi WHERE broj=3

DELETE FROM posudi WHERE broj=4

EXEC rupica

ROLLBACK

početak kraj rupe

---------- ----------

2 4

6 NULL

(2 row(s) affected)

Zanimljivo bi bilo pokazati kako bi rezultat izgledao u slučaju da je drugi *DELETE* bio u komentaru (*--DELETE FROM posudi WHERE broj=3*)

početak kraj rupe

---------- ----------

2 2

4 4

6 NULL

(3 row(s) affected)

Vidimo da je onda početak i kraj isti što je normalno jer izostaje samo jedan broj. Na samom kraju imamo prvi idući broj u našem slučaju 6 jer imamo 5 redova te budući da nema kraja stoji oznaka *NULL.* Treba objasniti kako se došlo do ovih brojeva. Ako nemamo broj 4 u listi od 5

pri prolasku programskog koda kroz broj 3 uzmemo vrijednost 4 (*posudi.broj + 1*) i gledamo da li se nalazi u listi postojećih brojeva (*NOT posudi.broj + 1 IN (SELECT broj FROM posudi)*) ako ga nema dobili smo početak rupe u našem nizu jer u tablici posudi nemamo iduću cjelobrojnu vrijednost. Isto vrijedi i za (*posudi.broj - 1*) osim što onda dobivamo kraj. Idući primjer pokazuje proceduru sa ulaznim parametrima. Zadatak joj je da kad unesemo šifru mobitela ispiše ime i prezime zadnje osobe koja je koristila taj mobitel.

Primjer 91:

```
CREATE PROCEDURE zadnji
 @moby NVARCHAR(15)
 AS
SELECT ime, prezime FROM osoba WHERE maticni =
(SELECT TOP 1 osoba FROM posudi
WHERE mobitel = @moby ORDER BY vratio DESC)

EXEC zadnji 'soz7'
```

```
ime             prezime
--------------- ---------------
Ivan            Ivic
(1 row(s) affected)
```

Vidimo da se ulazni parametar definira kao

@ nazivParametra tipPodatka i kasnije normalno kao takav koristi u kodu. Posebna vrsta procedura su okidači jer se pokreću automatski. Oni nisu dio 1992 ISO-ANSI SQL-a već su dodatak bazama. Ograničavaju unos, promjenu i brisanje podataka.

20. ZAKLJUČAK

Moderan način poslovanja je nezamisliv bez računala i baza podataka. Ipak u popularnoj informatičkoj štampi nema puno mjesta za baze podataka i SQL zbog nezainteresiranosti korisnika za to područje. Ono što možemo naći su osnove, tako da se lako stekne mišljenje da SQL i nije jezik. Na mom početku se upravo to i dogodilo. Gledajući naredbe SQL-a (UPDATE, SELECT, DELETE, INSERT) pomislio sam da sa tim jezikom ne mogu puno napraviti . Dobivajući razne zadatke učio sam više o SQL-u i sve uspio napraviti u njemu. Uz osnove, trudio sam se pokazati i takve naprednije mogućnosti u ovom radu kao npr. numeriranje ili usporedba brojeva kao znakovnih podataka. Jedna od otežavajućih okolnosti u radu s bazama podataka su svakako dijakritički znakovi o kojima moramo voditi računa pri kreiranju baze i dodjeljivanju tipova podataka. Format datuma također predstavlja specifičnost jer moramo odrediti način na koji se upisuje i prikazuje (redoslijed mjeseca, dana i godine). Kod nas je standardan način upisivanja dan, mjesec i godina. Drugi jezik čije bi osnove trebali znati za rad s bazama podataka na Internetu je XML (eXtensible Markup Language). To je jezik koji pretendira na područje WEB-a i baza podataka postaviti nove standarde, a mnoge baze ga počinju podržavati. Na kraju mogu reći da su baze podataka važan dio automatizacije poslovanja koje običan korisnik nikad neće primjetiti dok bi programeri i administratori trebali biti dobro upoznati sa tom tematikom.

21. DODATAK:

21.1. KONFIGURACIJA

Radio sam na svom računalu slijedećih karakteristika :

procesor – INTEL Celeron Tualatin 1300 MHz, 256 kb cache, FC PGA 2, BUS 100 MHz
matična ploča – EP-3VSA2 VIA Apollo Pro133T AGPSet
RAM – 256 MB
video kartica – Voodoo Banshee 16 MB
monitor – ProVista 17 "
tvrdi diskovi
– IBM 10 GB, 7.200 o/min., 2 MB cache
– MAXTOR 40 GB, 5.400 o/min., 2 MB cache
te ostali standardni periferni uređaji.

Operativni sustav : – Win 2000 Professional

Baze podataka : – MS SQL Server 7.0
– SAP DB 7.30

Alati za rad : – SAP SQL Studio
– MS SQL Server Query Analyzer

21.2. KLJUČNE RIJEČI SAP DB :

ABS, ABSOLUTE, ACOS, ADDDATE, ADDTIME

ALL, ALPHA, ALTER, ANY, ASCII

ASIN, ATAN, ATAN2, AVG

BINARY, BIT, BOOLEAN, BYTE

CEIL, CEILING, CHAR, CHARACTER, CHECK

CHR, COLUMN, CONCAT, CONNECTED, CONSTRAINT

COS, COSH, COT, COUNT, CROSS

CURDATE, CURRENT, CURTIME

DATABASE, DATE, DATEDIFF, DAY, DAYNAME

DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, DBYTE, DEC

DECIMAL, DECODE, DEFAULT, DEGREES, DELETE

DIGITS, DIRECT, DISTINCT, DOUBLE

EBCDIC, EXCEPT, EXISTS, EXP, EXPAND

FIRST, FIXED, FLOAT, FLOOR, FOR

FROM, FULL

GRAPHIC, GREATEST, GROUP

HAVING, HEX, HOUR

IFNULL, IGNORE, INDEX, INITCAP, INNER

INSERT, INT, INTEGER, INTERNAL, INTERSECT

INTO

JOIN

KEY

LAST, LCASE, LEAST, LEFT, LENGTH

LFILL, LINK, LIST, LN, LOCALSYSDBA

LOCATE, LOG, LOG10, LONG, LONGFILE

LOWER, LPAD, LTRIM

MAKEDATE, MAKETIME, MAPCHAR, MAX, MBCS

MICROSECOND, MIN, MINUTE, MOD, MONTH

MONTHNAME

NATURAL, NCHAR, NEXT, NOROUND, NO

NOT, NOW, NULL, NUM, NUMERIC

OBJECT, OF, ON, ORDER

PACKED, PI, POWER, PREV, PRIMARY

RADIANS, REAL, REFERENCED, REJECT, RELATIVE

REPLACE, RFILL, RIGHT, ROUND, ROWID

ROWNO, RPAD, RTRIM

SECOND, SELECT, SELUPD, SERIAL, SET

SHOW, SIGN, SIN, SINH, SMALLINT

SOME, SOUNDEX, SPACE, SQRT, STAMP

STATISTICS, STDDEV, SUBDATE, SUBSTR, SUBSTRING

SUBTIME, SUM, SYSDBA

TABLE, TAN, TANH, TIME, TIMEDIFF

TIMESTAMP, TIMEZONE, TO, TOIDENTIFIER,

TRANSACTION

TRANSLATE, TRIM, TRUNC, TRUNCATE

UCASE, UID, UNICODE, UNION, UPDATE

UPPER, USER, USERGROUP, USING, UTCDIFF

VALUE, VALUES, VARCHAR, VARGRAPHIC, VARIANCE

WEEK, WEEKOFYEAR, WHERE, WITH

YEAR

ZONED

21.3. KREIRANJE BAZE ZA SAP

Da bi se pokazalo kreiranje baze podataka za SAP slijedi prikaz datoteke u kojoj definiramo svojstva nove baze podataka.

```
@echo off
set PATH=E:\Program Files\sapdb\indep_prog\bin;E:\Program
Files\sapdb\indep_prog\pgm;%PATH%
set INSTPATH="E:\Program Files\sapdb\depend"
rem IME BAZE PODATAKA
set SID=TST1
rem MJESTO GDJE ĆE BITI SPREMLJENE DATOTEKE BAZE
set DATA=E:\data
rem START COMMUNICATION SERVER
x_server stop
x_server start
rem STOP I DROP EVENTUALNO POSTOJEĆE BAZE
dbmcli -d %SID% -u dbm,dbm db_offline > NUL
dbmcli -d %SID% -u dbm,dbm db_drop > NUL
rem KREIRANJE NOVE BAZE PODATAKA
dbmcli -R %INSTPATH% db_create %SID% dbm,dbm
rem KREIRANJE DIREKTORIJA ZA DATOTEKE BAZE
md %DATA% > NUL
md %DATA%\%SID% > NUL
rem  POSTAVLJAMO PARAMETRE BAZE
echo param_rmfile > param.tmp
echo param_startsession >> param.tmp
echo param_init OLTP >> param.tmp
echo param_put LOG_MODE SINGLE >> param.tmp
echo param_put CAT_CACHE_SUPPLY 300 >> param.tmp
echo param_put DATA_CACHE 2500 >> param.tmp
echo param_put MAXDATADEVSPACES 5 >> param.tmp
echo param_put MAXDATAPAGES 1024000 >> param.tmp
echo param_put RESTART_SHUTDOWN AUTO >> param.tmp
```

```
echo param_put _UNICODE YES >> param.tmp
echo param_checkall >> param.tmp
echo param_commitsession >> param.tmp
echo param_adddevspace 1 SYS  %DATA%\%SID%\DISKS01   F >> param.tmp
echo param_adddevspace 1 DATA %DATA%\%SID%\DISKD0001 F 2560 >>
param.tmp
echo param_adddevspace 1 LOG  %DATA%\%SID%\DISKL001  F 1024 >>
param.tmp
type param.tmp | dbmcli -d %SID% -u dbm,dbm
rem POKREĆEMO BAZU
dbmcli -d %SID% -u dbm,dbm db_start
rem INICIJALIZACIJA
echo util_connect dbm,dbm > param.tmp
echo util_execute init config >> param.tmp
echo util_activate dba,dba >> param.tmp
type param.tmp | dbmcli -d %SID% -u dbm,dbm
rem UČITAJ TABLICE SUSTAVA
dbmcli -d %SID% -u dbm,dbm load_systab -u dba,dba -ud domain
rem KREIRANJE KORISNIKA
echo sql_connect dba,dba > param.tmp
echo sql_execute CREATE USER test PASSWORD test DBA NOT EXCLUSIVE >>
param.tmp
type param.tmp | dbmcli -d %SID% -u dbm,dbm
echo medium_put data %DATA%\%SID%\datasave FILE DATA 0 8 YES >
param.tmp
echo medium_put auto %DATA%\%SID%\autosave FILE AUTO >> param.tmp
echo util_connect dbm,dbm >> param.tmp
echo backup_save data >> param.tmp
echo autosave_on >> param.tmp
type param.tmp | dbmcli -d %SID% -u dbm,dbm
del param.tmp
```

21.4. PROCEDURA ZA IZVOZ PODATAKA MS SQL

```
/*
Version: SQL Server 7.0/2000
Created by: Alexander Chigrik
Upgrade: PSO
*/
CREATE PROC DataAsInsCommand (
  @TableList nvarchar (200))
AS
SET NOCOUNT ON
DECLARE @position int, @exec_str nvarchar (2000), @TableName nvarchar (30)
DECLARE @name nvarchar(128), @xtype int, @status tinyint, @IsIdentity tinyint
SELECT @TableList = @TableList + ','
SELECT @IsIdentity = 0
SELECT @position = PATINDEX('%,%', @TableList)
WHILE (@position <> 0)
  BEGIN
    SELECT @TableName = SUBSTRING(@TableList, 1, @position - 1)
    SELECT @TableList = STUFF(@TableList, 1, PATINDEX('%,%', @TableList),
'')
    SELECT @position = PATINDEX('%,%', @TableList)
    SELECT @exec_str = 'DECLARE fetch_cursor CURSOR FOR '
      + 'SELECT name, xtype, status FROM syscolumns WHERE id = object_id('''
      + @TableName + ''')'
    EXEC (@exec_str)
    OPEN fetch_cursor
    FETCH fetch_cursor INTO @name, @xtype, @status
    IF (@status & 0x80) <> 0
      BEGIN
        SELECT @IsIdentity = 1
        SELECT 'SET IDENTITY_INSERT ' + @TableName + ' ON'
        SELECT 'GO'
```

```
    END
   SELECT @exec_str = "SELECT REPLACE ('INSERT INTO " + @TableName +
" VALUES (' + "
   --text or ntext
   IF (@xtype = 35) OR (@xtype = 99)
      SELECT @exec_str = @exec_str + ""'None yet'""
   ELSE
   --image
   IF (@xtype = 34)
      SELECT @exec_str = @exec_str + "'' + '0xFFFFFFFF' + "''
   ELSE
   --smalldatetime or datetime
   IF (@xtype = 58) OR (@xtype = 61)
      SELECT @exec_str = @exec_str + ' + ""'' + ' + ' + CONVERT(nvarchar,' +
@name + ',101)' + ' + ""''
   ELSE
   --nvarchar or char or nvarchar or nchar
   IF (@xtype = 167) OR (@xtype = 175) OR (@xtype = 231) OR (@xtype = 239)
      SELECT @exec_str = @exec_str + ""'' + ISNULL(' + @name + ',"null") + ""''
   ELSE
   --uniqueidentifier
   IF (@xtype = 36)
      SELECT @exec_str = @exec_str + ' + ""'' + ' + ' + CONVERT(nvarchar(255),' +
@name + ')' + ' + ""''
   ELSE
   --binary or varbinary
   IF (@xtype = 173) OR (@xtype = 165)
      SELECT @exec_str = @exec_str + "'' + '0x0' + "''
   ELSE
      SELECT @exec_str = @exec_str + 'ISNULL(CONVERT(nvarchar,' + @name +
'), "null")'
   WHILE @@FETCH_STATUS <> -1
    BEGIN
      FETCH fetch_cursor INTO @name, @xtype, @status
```

```
        IF (@@FETCH_STATUS = -1) BREAK
        IF (@status & 0x80) <> 0
         BEGIN
           SELECT @IsIdentity = 1
           SELECT 'SET IDENTITY_INSERT ' + @TableName + ' ON'
           SELECT 'GO'
         END
        --text or ntext
        IF (@xtype = 35) OR (@xtype = 99)
          SELECT @exec_str = @exec_str + ' + '',''' + ' + ''''None yet''''
        ELSE
        --image
        IF (@xtype = 34)
          SELECT @exec_str = @exec_str + ' + ',' + ' + ''' + '0xFFFFFFFF' + '''
        ELSE
        --smalldatetime or datetime
        IF (@xtype = 58) OR (@xtype = 61)
          SELECT @exec_str = @exec_str + ' + ','''' + ' + ''''' + ' + ' +
CONVERT(nvarchar,' + @name + ',101)' + ' + ''''''
        ELSE
        --nvarchar or char or nvarchar or nchar
        IF (@xtype = 167) OR (@xtype = 175) OR (@xtype = 231) OR (@xtype = 239)
          SELECT @exec_str = @exec_str + ' + '',''' + ' + ''''' + ISNULL(' + @name +
',''null'') + ''''''
        ELSE
        --uniqueidentifier
        IF (@xtype = 36)
          SELECT @exec_str = @exec_str + ' + ','''' + ' + ''''' + ' + ' +
CONVERT(nvarchar(255),' + @name + ')' + ' + ''''''
        ELSE
        --binary or varbinary
        IF (@xtype = 173) OR (@xtype = 165)
          SELECT @exec_str = @exec_str + ' + ',' + ' + ''' + '0x0' + '''
        ELSE
```

```
        SELECT @exec_str = @exec_str + ' + ","' + ' +
ISNULL(CONVERT(nvarchar,' + @name + '), "null")'
     END
   CLOSE fetch_cursor
   DEALLOCATE fetch_cursor
   SELECT @exec_str = @exec_str + '+ ")", ' + "'" + '"null"' + "'" + ' , ' + "'" + 'null' +
"'" + ') FROM ' + @TableName
   SELECT @exec_str
   EXEC(@exec_str)
   SELECT 'GO'
   IF @IsIdentity = 1
     BEGIN
       SELECT @IsIdentity = 0
       SELECT 'SET IDENTITY_INSERT ' + @TableName + ' OFF'
       SELECT 'GO'
     END
  END
```

21.5. PROGRAMSKI KOD ZA KREIRANJE I POPUNJAVANJE TABLICA (MS SQL)

```
CREATE TABLE osoba
(
   maticni NVARCHAR(15),
   ime NVARCHAR(15) NOT NULL,
   prezime NVARCHAR(15) NOT NULL,
   ulica NVARCHAR(25),
   mjesto NVARCHAR(15) DEFAULT 'Zagreb'
  PRIMARY KEY (maticni)
 );

CREATE TABLE mobitel
(
 sifra NVARCHAR(15),
 proizvodi NVARCHAR(15) NOT NULL,
model NVARCHAR(15) NOT NULL,
tezina INT,
visina INT,
sirina INT,
debljina INT
 PRIMARY KEY (sifra),
UNIQUE (proizvodi, model)
 );
```

```
CREATE TABLE posudi
(
broj INT IDENTITY (1, 1) NOT NULL,
osoba  NVARCHAR(15),
mobitel NVARCHAR(15),
uzeo DATETIME,
vratio DATETIME,
napomena NVARCHAR(25),
 FOREIGN KEY (osoba) REFERENCES osoba,
FOREIGN KEY (mobitel) REFERENCES mobitel
 );
```

(SAP DB)

```
CREATE TABLE osoba
(
   maticni VARCHAR(15) UNICODE,
   ime VARCHAR(15) UNICODE NOT NULL,
   prezime VARCHAR(15) UNICODE NOT NULL,
   ulica VARCHAR(25) UNICODE,
   mjesto VARCHAR(15) UNICODE DEFAULT 'Zagreb',
  PRIMARY KEY (maticni)
 )

CREATE TABLE mobitel
(
 sifra VARCHAR(15) UNICODE,
 proizvodi VARCHAR(15) UNICODE NOT NULL,
model VARCHAR(15) UNICODE NOT NULL,
```

```
tezina INT,
visina INT,
sirina INT,
debljina INT,
UNIQUE (proizvodi, model),
 PRIMARY KEY (sifra)
  )

CREATE TABLE posudi
(
broj INT DEFAULT SERIAL NOT NULL ,
osoba  VARCHAR(15) UNICODE,
mobitel VARCHAR(15) UNICODE,
uzeo DATE,
vratio DATE,
napomena VARCHAR(25) UNICODE,
 FOREIGN KEY (osoba) REFERENCES osoba,
 FOREIGN KEY (mobitel) REFERENCES mobitel
 )
```

--TABLICA OSOBA:

```
INSERT INTO osoba
        VALUES('0102968383911', 'Pero', 'Perić', 'Gajeva 3', 'Zagreb');
INSERT INTO osoba VALUES('0302982383818', 'Ivan', 'Ivić', '', 'Split');
INSERT INTO osoba
        VALUES('0305972383915', 'Marko', 'Marić', 'Lavova 67', 'Zadar');
INSERT INTO osoba
        VALUES('1212972383944', 'Ivan', 'Dundov', 'Obala 4', 'Zagreb');
INSERT INTO osoba
```

VALUES('1717985383542', 'Ivan', 'Pos', NULL, 'Zagreb');

--TABLICA MOBITEL:

INSERT INTO MOBITEL
VALUES('no3310', 'nokia', '3310', '133', '113', '48', '22');
INSERT INTO MOBITEL
VALUES('no3330', 'nokia', '3330', '133', '113', '48', '22');
INSERT INTO MOBITEL VALUES('soz7', 'sony', 'z7', '95', '91', '50', '25');
INSERT INTO MOBITEL
VALUES('mov60', 'motorola', 'v60', '109', '85', '45', '25');
INSERT INTO MOBITEL
VALUES('soj70', 'sony', 'j70', '92', '133', '45', '22');

--TABLICA POSUDI MS SQL:

INSERT INTO posudi
VALUES('0102968383911', 'no3310', '01.01.2002', '01.30.2002', '');
INSERT INTO posudi
VALUES('0302982383818', 'soz7', '01.15.2002', '02.02.2002', '');
INSERT INTO posudi
VALUES('0102968383911', 'no3310', '03.03.2002', '05.15.2002', '');
INSERT INTO posudi
VALUES('1212972383944', 'mov60', '02.15.2002', '07.15.2002', '');
INSERT INTO posudi
VALUES('0102968383911', 'no3330', '06.01.2002', '10.01.2002', '');

--TABLICA POSUDI SAP DB:

INSERT INTO posudi (osoba, mobitel, uzeo, vratio, napomena)
VALUES('0102968383911', 'no3310', '2002-01-01', '2002-01-30', '')

INSERT INTO posudi (osoba, mobitel, uzeo, vratio, napomena)
VALUES('0302982383818', 'soz7', '2002-01-15', '2002-02-02', '')

INSERT INTO posudi (osoba, mobitel, uzeo, vratio, napomena)
VALUES('0102968383911', 'no3310', '2002-03-03', '2002-05-15', '')

INSERT INTO posudi (osoba, mobitel, uzeo, vratio, napomena)
VALUES('1212972383944', 'mov60', '2002-02-15', '2002-07-15', '')

INSERT INTO posudi (osoba, mobitel, uzeo, vratio, napomena)
VALUES('0102968383911', 'no3330', '2002-06-01', '2002-10-01', '')

21.6. FUNKCIJE MOBITELA KOJE SE KORISTE U EKSPLOATACIJI

proizvođač	model	težina [mm]	visina [mm]	debljina [mm]	širina [mm]
Motorola	v60	109	85	45	25
Nokia	3310	133	113	48	22
Nokia	3330	133	113	48	22
Sony	j70	92	133	45	22
Sony	z7	95	91	50	25

21.7. SINTAKSA OSNOVNIH NAREDBI ZA MS SQL

21.7.1. NAREDBA SELECT

```
SELECT statement ::=
   <query_expression>
   [ ORDER BY { order_by_expression | column_position [ ASC | DESC
] }
      [,...n]    ]
   [ COMPUTE
      { { AVG | COUNT | MAX | MIN | SUM } (expression) } [,...n]
      [ BY expression [,...n] ]
   ]
   [ FOR BROWSE ]
   [ OPTION (<query_hint> [,...n]) ]

<query expression> ::=
   { <query specification> | (<query expression>) }
   [UNION [ALL] <query specification | (<query expression>) [...n] ]

<query specification> ::=
   SELECT [ ALL | DISTINCT ]
      [ {TOP integer | TOP integer PERCENT} [ WITH TIES] ]
      <select_list>
   [ INTO new_table ]
   [ FROM {<table_source>} [,...n] ]
   [ WHERE <search_condition> ]
   [ GROUP BY [ALL] group_by_expression [,...n]
      [ WITH { CUBE | ROLLUP } ]
   ]
   [ HAVING <search_condition> ]
```

21.7.2. NAREDBA UPDATE

```
UPDATE
      {
       table_name WITH ( <table_hint_limited> [...n])
      | view_name
      | rowset_function_limited
      }
      SET
      {column_name = {expression | DEFAULT | NULL}
      | @variable = expression
      | @variable = column = expression } [,...n]

   {{[FROM {<table_source>} [,...n] ]

      [WHERE
         <search_condition>] }
      |
      [WHERE CURRENT OF
      { { [GLOBAL] cursor_name } | cursor_variable_name}
      ] }
      [OPTION (<query_hint> [,...n] )]

<table_source> ::=
   table_name [ [AS] table_alias ] [ WITH ( <table_hint> [,...n]) ]
   | view_name [ [AS] table_alias ]
   | rowset_function [ [AS] table_alias ]
   | derived_table [AS] table_alias [ (column_alias [,...n] ) ]
   | <joined_table>

<joined_table> ::=
   <table_source> <join_type> <table_source> ON <search_condition>
   | <table_source> CROSS JOIN <table_source>
   | <joined_table>

<join_type> ::=
   [ INNER | { { LEFT | RIGHT | FULL } [OUTER] } ]
   [ <join_hint> ]
   JOIN

<table_hint_limited> ::=
   {    INDEX(index_val [,...n])
      | FASTFIRSTROW
      | HOLDLOCK
```

```
      | PAGLOCK
      | READCOMMITTED
      | REPEATABLEREAD
      | ROWLOCK
      | SERIALIZABLE
      | TABLOCK
      | TABLOCKX
   }

<table_hint> ::=
   {    INDEX(index_val [,...n])
      | FASTFIRSTROW
      | HOLDLOCK
      | NOLOCK
      | PAGLOCK
      | READCOMMITTED
      | READPAST
      | READUNCOMMITTED
      | REPEATABLEREAD
      | ROWLOCK
      | SERIALIZABLE
      | TABLOCK
      | TABLOCKX
      | UPDLOCK
   }

<query_hint> ::=
   {    { HASH | ORDER } GROUP
      | { CONCAT | HASH | MERGE     } UNION
      | {LOOP | MERGE | HASH} JOIN
      | FAST number_rows
      | FORCE ORDER
      | MAXDOP
      | ROBUST PLAN
      | KEEP PLAN
   }
```

21.7.3. NAREDBA DELETE

```
DELETE
   [FROM ]
      {
       table_name WITH ( <table_hint_limited> [...n])
       | view_name
       | rowset_function_limited
       }

      [ FROM {<table_source>} [,...n] ]

   [WHERE
      { <search_condition>
      |   { [ CURRENT OF
            {
               { [ GLOBAL ] cursor_name }
               | cursor_variable_name
            }
         ]
      }
   ]
   [OPTION (<query_hint> [,...n])]

<table_source> ::=
   table_name [ [AS] table_alias ] [ WITH ( <table_hint> [,...n]) ]
   | view_name [ [AS] table_alias ]
   | rowset_function [ [AS] table_alias ]
   | derived_table [AS] table_alias [ (column_alias [,...n] ) ]
   | <joined_table>

<joined_table> ::=
    <table_source> <join_type> <table_source> ON <search_condition>
   | <table_source> CROSS JOIN <table_source>
   | <joined_table>

<join_type> ::=
   [ INNER | { { LEFT | RIGHT | FULL } [OUTER] } ]
   [ <join_hint> ]
   JOIN

<table_hint_limited> ::=
   {    INDEX(index_val [,...n])
      | FASTFIRSTROW
```

```
    | HOLDLOCK
    | PAGLOCK
    | READCOMMITTED
    | REPEATABLEREAD
    | ROWLOCK
    | SERIALIZABLE
    | TABLOCK
    | TABLOCKX
  }

<table_hint> ::=
  {   INDEX(index_val [,...n])
    | FASTFIRSTROW
    | HOLDLOCK
    | NOLOCK
    | PAGLOCK
    | READCOMMITTED
    | READPAST
    | READUNCOMMITTED
    | REPEATABLEREAD
    | ROWLOCK
    | SERIALIZABLE
    | TABLOCK
    | TABLOCKX
    | UPDLOCK
  }

<query_hint> ::=
  {   {   HASH | ORDER } GROUP
    | { CONCAT | HASH | MERGE } UNION
    | FAST number_rows
    | FORCE ORDER
    | MAXDOP
    | ROBUST PLAN
    | KEEP PLAN
  }
```

21.7.4. NAREDBA INSERT

```
INSERT [INTO]
   {
    table_name WITH ( <table_hint_limited> [...n])
    | view_name
    | rowset_function_limited
    }

   {    [(column_list)]
      { VALUES ( {    DEFAULT
                  |   NULL
                  |   expression
                   }[,...n]
         )
      | derived_table
      | execute_statement
      }
   }
   | DEFAULT VALUES


<table_hint_limited> ::=
   {    INDEX(index_val [,...n])
      | FASTFIRSTROW
      | HOLDLOCK
      | PAGLOCK
      | READCOMMITTED
      | REPEATABLEREAD
      | ROWLOCK
      | SERIALIZABLE
      | TABLOCK
      | TABLOCKX
   }
```

21.8. KRATICE KORIŠTENE U RADU

DBMS (Database Management System)
sustav za upravljanje bazama podataka

DDL (Data Definition Language)
funkcije za definiciju podatka

DML (Data Manipulation Language)
funkcije za upravljanje podacima

ODBC (Open Database Connectivity)
JDBC (Java Database Connectivity)
programska sučelja

MS Microsoft

RDBMS (Relational Database Management System)
relacijski sustav za upravljanje bazama podataka

SQL (Structured Query Language)
strukturirani jezik upita

XML (eXtensible Markup Language)
proširivi jezik za označavanje

22. LITERATURA

1. Judith S. Bowman, Sandra L. Emerson, Marcy Darnovsky
"The Practical SQL Handbook, Fourth Edition"
Addison-Wesley, 2001

2. Ryan K. Stephens, Ronald R. Plew, Bryan Morgan, Jeff Perkins
"Teach Yourself SQL in 21 Days, Second Edition"
Sams, 1997

3. Mike Gunderloy, Joseph L. Jorden
"Mastering SQL Server 2000"
Sybex, Inc., 2000

4. SQL
ZPM – FER : SQL – 2001

5. Tere' Parnell, Christopher Null
"Network Administrator's Reference"
Osborne/McGraw-Hill, 2000

6. James Hoffman
"Introduction to Structured Query Language"
version 4.66, 2001, e - book

7. http://www.mssqlcity.com/Scripts/scrImpExp.htm