- **Odredba GROUP BY**

GROUP BY odredba logički deli tabelu na grupe n-torki tako da u okviru jedne grupe sve n-torke imaju istu vrednost zadate kolone. Ovim se omogućuje da funkcije za dobijanje sumarnih informacija budu primenjene na svaku ovakvu grupu posebno, umesto na celu tabelu. Prouzrokuje dobijanje jednog rezultujućeg reda za svaku razilčtu vrednost kolone po kojoj se vrši grupisanje.

U istu grupu ulaze n-torke koje imaju jednake vrednosti atributa i svi atributi koji se nalaze u listi za selekciju, a koji nisu argumenti agregatnih funkcija, moraju biti navedeni u GROUP BY delu naredbe.

Za svaki predmet ispisati prosečne poene :

⊙ grupirati po predmetu
⊙ za svaku grupu izračunati AVG(Ispitni poeni)
⊙ za svaku grupu formirati po jednu n-torku s vrednošću atributa Predmet i izračunatim prosekom
⊙ obaviti operaciju preimenovanja.

Sintaksa ove odredbe ima sledeći izgled:

```
SELECT [DISTINCT] kolona [,kolona...]
FROM tabela
[WHERE uslov-selekcije]
[GROUP BY izraz {,izraz}]
ORDER BY (izraz|pozicija)[ASC | DESC]
```

Primer:

- Naći minimalni, srednji i maksimalni broj poena kao i ukupan broj studenata na svakom smeru:

```
SELECT MIN (Ispitni_poeni), AVG (Ispitni_poeni),
       MAX (Ispitni_poeni), COUNT (*), SmerID
       FROM Smer GROUP BY SmerID;
```

MIN(Ispit_poeni)	AVG(Ispit_poeni)	MAX(Ispit_poeni)	COUNT(*)	SmerID#
67,5	74,875	85	4	1
67	68,167	69	3	2
72,5	76,333	79	3	3

Grupisanje se može vršiti po više kolona. U tom slučaju svaka različita postojeća kombinacija vrednosti kolona čini grupu. SQL naredbe mogu sadržati aritmetičke izraze sastavljene od imena kolona i konstantnih vrednosti povezanih aritmetičkim operatorima ("+", "*","-", "/").

Primeri:

- Koliki je srednji broj poena studenata sa elektronike:

```
SELECT AVG   (Ispitni_poeni + Dodatni_poeni)
FROM Student
WHERE  Smer = 'Elektronika';

AVG(Ispitni_poeni + Dodatni_poeni)
_____________________________________
                                74,66
```

- Izračunati broj studenata sa različitog smera koji slušaju isti predmet:

```
SELECT PredID,   Smer,   COUNT(*)
FROM Student
GROUP BY PredID,   Smer;
```

PredID	Smer	Count
4	Informatika	3
5	Informatika	1
6	Elektronika	1
7	Menadžment	2
8	Menadžment	2
9	Elektronika	1

Kada se koristi GROUP BY u okviru WHERE odredbe onda se redovi koji ne zadovoljavaju uslov u okviru WHERE odredbe eliminišu pre grupisanja. WHERE i GROUP BY odredba mogu se koristiti zajedno, pri čemu WHERE odredba uvek ide pre GROUP BY odredbe.

Primeri:

- Treba prikazati smer, srednju aritmetičku vrednost i broj studenata čiji su Ispitni poeni veći od 60. Sintaksa ove odredbe ima sledeći izgled:

```
SELECT Smer, AVG   (Ispitni_poeni), COUNT (*)
FROM Student
 WHERE Ispitni_poeni > 60
 GROUP BY Smer;
```

Smer	avg	count
Informatika	79	3
Elektronika	68	3
Menadžment	85	4

- Odrediti srednji broj poena studenata za svaki smer. Pri računanju ne uzimati u obzir studente čija je šifra predmeta 4:

```
SELECT SmerID,   AVG   (Ispitni_poeni)
FROM Student
WHERE PredID  NOT  IN  4
GROUP BY SmerID;
```

SmerID	AVG(Ispitni_poeni)
1	76
2	68
3	75,75

- **HAVING odredba**

Grupisanje se može vršiti po više kolona, i tada svaka različita kombinacija kolona predstavlja jednu grupu. U okviru dobijenih grupa mogu se uvoditi dodatni uslovi za selekciju primenom odredbe HAVING (koji imaju). Određuje kriterijume za selekciju grupa koje su prethodno specificirane GROUP BY odredbom.

Primer:

- Prikazati koje predmete sluša više od 1 studenta na svakom smeru:

```
SELECT SMER, PredID, COUNT(*)
FROM STUDENT
GROUP BY SMER, PREDID
HAVING COUNT(*)>1;
```

Smer	PredID	Count
Menadžment	8	2

1.1.1 Upiti spajanja više tabele istovremeno

Kada su nam potrebni podaci iz dve ili više tabela, tada se koristi povezivanje (JOIN) n-torke različitih tabela koristeći zajedničke atribute. U skladu sa relacionom teorijom podržani su različiti tipovi spajanja:

- spajanje izjednačavanjem
- spoljno levo spajanje i
- spoljno desno spajanje.

Operacija spajanja može se zahtevati:

- implicitno - zadavanjem uslova spajanja u WHERE odredbi upita.
- eksplicitno - navođenjem tipa i uslova spajanja u FROM odredbi upita -SQL-92 i SQL:1999 standard, pored implicitnog, omogućuju i eksplicitno navođenje tipa spajanja.

Koristeći zapis sa tačkama da biste izbegli dvosmislenost, ispred imena kolona možete pisati imena tabela. Međutim, kako su u tabelama imena kolona različita, ovo nije bilo neophodno.

- **Spajanje izjednačavanjem**

U uslovu spajanja upotrebljava se znak jednakosti za izjednačavanje vrenosti po kolonama iz različitih tabela, i ovo spajanje se naziva spajanje izjednačavanjem. Potpuno spajanje (ekvispajanje, equijoin) uvek daje rezultujuću tabelu sa dve identične kolone, odnosno kolone sa identičnim sadržajem.

Korišćenjem klasične sintakse spajanje izjednačavanjem tabela Student i Predmet može se prikazati sledećom SQL naredbom:

```
SELECT [kolone]
FROM tabela_1, tabela_2
WHERW uslov_spajanja;
```

Primer:

- Prikaži za svakog studenta ime, smer i podatke o predmetu koji sluša.

```
SELECT Student, Smer, Student.PredID,
       Predmet.PredID, Predmet, Profesor
FROM Student, Predmet
WHERE Student.PredID = Predmet.PredID;
```

Student	Smer	PredID	PredID	Predmet	Profesor
Marko Marić	Menadžment	7	7	Operativni sistemi	Milica Arsić
Dragan Perić	Elektronika	9	9	Digitalna elektronika	Nenad Perić
Nevena Simić	Menadžment	8	8	Marketing	Janko Jović
Milica Tasić	Informatika	4	4	Fizika	Lazar Ilić
Milan Ilić	Informatika	5	5	Informacioni sistemi	Željka Marić
Zoran Marić	Elektronika	4	4	Fizika	Lazar Ilić
Sanja Lukić	Elektronika	6	6	Programski jezici	Nikola Lazić
Mirko Spasić	Menadžment	4	4	Fizika	Lazar Ilić
Tamara Dasić	Menadžment	8	8	Marketing	Janko Jović
Željko Spasić	Informatika	7	7	Operativni sistemi	Milica Arsić

Da bi se uklonile identične kolone spajanje može se prikazati sledećom SQL naredbom:

```
SELECT Student.*, Predmet, Profesor
FROM Student, Predmet
WHERE Student.PredID = Predmet.PredID;
```

Eksplicitnim navođenjem odredbe za unutrašnje (INNER) spajanje izjednačavanjem kao što se koristi u Access-u sintaksa je sledeća:

```
SELECT [kolone]
FROM tabela_1 INNER JOIN tabela_2
ON uslov_spajanja;
```

Primer:

- Prikaži za svakog studenta ime, smer i podatke o predmetu I profesoru koji sluša

```
SELECT Student.*, Predmet, Profesor
FROM Student INNER JOIN Predmet
ON Student.PredID = Predmet.PredID;
```

Ako se i uslov spajanja i uslov selekcije u nalaze istom upitu tada se upit realizuje klasičnom sintaksom u WHERE odredbi.

Primer:

- Za svakog informatičara prikaži ime, prezime i profesora kod kojeg sluša predmet.

```
SELECT Student, Profesor
FROM Student, Predmet
WHERE Student.PredID = Predmet.PredID
   AND Smer = 'Informatika';
```

Student	Profesor
Milica Tasić	Lazar Ilić
Milan Ilić	Željka Marić
Željko Spasić	Milica Arsić

Ako se upit realizuje eksplicitnim navođenjem operacije spajanja u FROM odredbi, u WHERE odredbi ostaće samo uslov selekcije.

```
SELECT [kolone]
FROM tabela_1 INNER JOIN tabela_2
ON uslov_spajanja;
```

Primer:

- Za svakog informatičara prikaži ime, prezime i profesora kod kojeg sluša predmet.

```
SELECT Student, Profesor
FROM Student INNER JOIN Predmet
ON Student.PredID = Predmet.PredID
   WHERE Smer = 'Informatika';
```

⊙ SQL-92 i SQL:1999 standard omogućuju eksplicitno navođenje operacije spajanja u FROM odredbi bez INNER na jedan od sledeća dva načina:

```
SELECT [kolone]
FROM tabela_1 JOIN tabela_2
ON uslov_spajanja;
```

Primer:

- Prikaži za svakog studenta ime, smer i podatke o predmetu koji sluša.

```
SELECT *
   FROM Student JOIN Predmet
        ON Student.PredID = Predmet.PredID;
```

Drugi način kada se spajanje vrši po svim kolonama sa identičnim nazivima u obe tabele.

```
SELECT [kolone]
FROM tabela_1 JOIN tabela_2
USING uslov_spajanja;
```

Primer:

```
SELECT *
FROM Student JOIN Predmet
        USING (PredID);
```

Ili na drugi način:

```
SELECT Student,  Profesor
FROM Student JOIN Predmet
        USING (PredID)
  WHERE Smer =  'Informatika';
```

- **Spoljno levo spajanje**

Levo spoljno spajanje (LEFT OUTER) omogućuje uključivanje u rezultujuću tabelu svih redova tabele sa leve strane JOIN odredbe tako što se praznim redom proširuje tabela sa desne strane.
SQL naredba sa eksplicitnim navođenjem spoljnog (OUTER) spajanja bila bi sledećeg izgleda.

Primer:

```
SELECT Student, Smer, Predmet
FROM Student LEFT OUTER JOIN Predmet
ON Student.PredID = Predmet.PredID
```

Student	Smer	Predmet
Goran Arsić	Inž.informatika	
Neda Bojić	Inž.informatika	
Milica Tasić	Informatika	Fizika
Zoran Marić	Elektronika	Fizika
Mirko Spasić	Menadžment	Fizika
Milan Ilić	Informatika	Informacioni sistemi
Sanja Lukić	Elektronika	Programski jezici
Marko Marić	Menadžment	Operativni sistemi
Željko Spasić	Informatika	Operativni sistemi
Nevena Simić	Menadžment	Marketing
Tamara Dasić	Menadžment	Marketing
Dragan Perić	Elektronika	Digitalna elektronika

Levim spoljnim spajanjem omogućuje se uključivanje u rezultujuću tabelu svih redova tabele sa leve strane *spajanja* tako što se praznim redom proširuje tabela sa desne strane. Odnosno, prikazuje sve zapise iz tabele Student bez obzira da li su uneti predmeti koje slušaju. Predmeti kojima nije dodeljen student neće biti prikazani u tabeli.

⊙ Levo spajanje tabela Student i Predmet može se realizovati i sledećim SQL naredbom po standardima SQL-92 i SQL:1999 standard:

```
SELECT *
   FROM Student  LEFT [OUTER] JOIN  Predmet
   USING (PredID);
```

- **Spoljno desno spajanje**

Desno spoljno spajanje (RIGHT OUTER) omogućuje uključivanje u rezultujuću tabelu svih redova tabele sa desne strane JOIN odredbe tako što se praznim redom proširuje tabela sa leve strane.

SQL naredba sa eksplicitnim navođenjem spoljnog (OUTER) spajanja bila bi sledećeg izgleda.

Primer:

```
SELECT  Predmet, Student, Smer
FROM Student RIGHT  OUTER  JOIN Predmet
ON Student.PredID  = Predmet.PredID
```

Student	Smer	Predmet
		Matematika
		Engleski jezik
		Informatika
Milica Tasić	Informatika	Fizika
Zoran Marić	Elektronika	Fizika
Mirko Spasić	Menadžment	Fizika
Milan Ilić	Informatika	Informacioni sistemi
Sanja Lukić	Elektronika	Programski jezici
Marko Marić	Menadžment	Operativni sistemi
Željko Spasić	Informatika	Operativni sistemi
Nevena Simić	Menadžment	Marketing
Tamara Dasić	Menadžment	Marketing
Dragan Perić	Elektronika	Digitalna elektronika

Desnim spoljnim spajanjem omogućuje se uključivanje u rezultujuću tabelu svih redova tabele sa desne strane spajanja tako što se praznim redom proširuje tabela sa leve strane. Odnosno, prikazuje sve predmete bez obzira da li su uneti studenti koji slušaju taj predmet. Studentim kojima nije dodeljen predmet neće biti prikazani u tabeli.

⊙ Desno spajanje tabela Student i Predmet može se realizovati i sledećim SQL naredbom po standardima SQL-92 i SQL:1999 standard:

```
SELECT *
FROM Student  RIGHT [OUTER] JOIN  Predmet
USING (PredID);
```

1.1.2 Podupit ili unutrašnji upit

Podupiti su upiti gde se nad jednom relacijom postavlja kao argument u upitu nad drugom relacijom. Kod takvih upita rezultat upita koji ima ulogu argumenta može biti dinamički zamenjen u WHERE odredbi drugog upita.

```
SELECT  kolona [,kolona...]
FROM tabela
WHERE uslov_selekcije
```

```
[SELECT kolona [,kolona...]
FROM tabela]
```
podupit

Primeri:

- Prikazati ime i prezime i smer svakog studenta koji ide na isti smer kao Marko Marić:

```
SELECT  Student,   Smer
```

```
    FROM Student
WHERE  Smer = (SELECT Smer FROM Student
                WHERE Student  =  'Marko_Marić');
```

Podupit najpre pronalazi smer na koji ide Marko Marić, a zatim spoljašnji upit po pronađenom smeru iz podupita traži i ostale studente na tom smeru.

Student	Smer
Marko Marić	Menadžment
Nevena Simić	Menadžment
Mirko Spasić	Menadžment
Tamara Dasić	Menadžment

- Koji studenti ima najviše poena na svom smeru?

```
SELECT  Student,  Smer,  SmerID, Ispitni_poeni
FROM Student
WHERE (SmerID, Ispitni_poeni) IN
        (SELECT SmerID, MAX (Ispitni_poeni)
        FROM Student
        GROUP BY SmerID);
```

Student	Smer	SmerID	Ispitni poeni
Nevena Simić	Menadžment	1	85
Sanja Lukić	Elektronika	2	69
Željko Spasić	Informatika	3	79

⊙ Upit koji se izvršava jedanput za svaki selektovani red spoljnog upita:

Primer:

- Prikaži šifru smera, ime, prezime i ispitne poene svakog studenta čiji je broj poena veći od prosečnog broja poena svog smera.

```
SELECT SmerID, Student,  Ispitni_poeni
        FROM Student
```

SmerID	Student	Ispitni poeni
1	Nevena Simić	85
3	Milica Tasić	77,5
2	Zoran Marić	68,5
2	Sanja Lukić	69

1.2 Naredbe za ažuriranje baze podataka

Ažuriranje u širem smislu značenja te reči obuhvata dodavanje, izmenu sadžaja i brisanje reda ili redova tabele. Te osnovne operacije realizuju se SQL naredbama INSERT, UPDATE i DELETE sa sledećim značenjem:

- INSERT- dodavanje reda ili redova u tabelu
- UPDATE- izmena sadržaja postojećeg reda ili redova tabele
- DELETE- brisanje postojećeg reda ili redova tabele

1.2.1 Dodavanje novih redova - INSERT

Razmatraće se sledeći tipovi INSERT naredbe:

1. Ubacivanje vrednosti svih atributa n-torke;
2. Ubacivanje vrednosti nekih atributa n-torke;
3. Ubacivanje podataka iz jedne tabele u drugu.

⊙ U prvom slučaju nije potrebno specificirati nazive atributa, pa INSERT naredba ima oblik:

```
INSERT INTO naziv_tabele
VALUES   (vrednost_atr1,  vrednost_atr2,  . . . ).
```

Za svaki atribut mora postojati vrednost, pri čemu je NULL dozvoljena opcija za svaki atribut koji nije NOT NULL.

Primeri:

- Ubaciti podatke o studentu Bojanu Simić, studentu koji se upisao na smer Informatika dana 7.04.2013. sa ispitnim poenima 81,5 i dodatnim poenima 23,5. Neposredni profesor još nije poznat.

```
INSERT  INTO Student  VALUES
        (143/13,   'Bojan Simić',   'Informatika', 3,
         '07-04-2013', 81.5, 23.5, NULL);
```

Prikažimo rezultat insert naredbe:

```
SELECT  *
FROM Student
WHERE Student =  'Bojan Simić';
```

⊙ Ako želimo da unesemo vrednost za samo neke atribute, nazivi tih atributa moraju se eksplicitno navesti. U tom slučaju naredba INSERT ima oblik:

```
INSERT INTO naziv_tabele   (atr1,  atr2,  . . . )
VALUES   (vrednost_atr1,  vrednost_atr2,  . . . ).
```

Vrednosti za NOT NULL atribute moraju biti unete.

Primeri:

- Uneti podatke o Milu Simić, Studentu, koji se upisao na smer Elektronika i šifra profesora kod koga sluša predmet je 2.

```
INSERT INTO Student
```

```
(Student, Broj_indeksa, PredID, Smer)
VALUES   ('Mile Simić',  543/13,  2,   'Elektronika')
```

Prikažimo rezultat INSERT naredbe

```
SELECT *
FROM Student
WHERE Broj_indeksa =  '543/13';
```

Broj Indeksa	Student	Smer	SmerID	DatUpisa	Ispitni poeni	Dodatni poeni	PredID
143/13	Bojan Simić	Informatika	3	07.04.2013	81,5	23,5	
543/13	Mile Simic	Elektronika					2

⊙ Ukoliko obe tabele imaju isti broj atributa i ukoliko su atributi identično definisani, naredba INSERT je oblika:

```
INSERT INTO tabela1
SELECT *  FROM tabela2;
```

inače:

```
INSERT INTO tabela1 (atribut1, atribut2, . . . )
SELECT atribut,   izraz
FROM tabela2
WHERE kriterijum selekcije
```

Primeri:

- Dodati svim studentima sa elektronike i automatike broj poena u iznosu od 10% ispitnih poena . Te informacije uneti u tabelu PREMIJA zajedno sa datumom upisa.

```
INSERT  INTO  PREMIJA  (Broj_indeksa, Ispitni_poeni, Smer, Datum_upisa)
 SELECT Broj_indeksa,Ispitni_poeni+Ispitni_poeni *.10,  Smer,Datum_upisa
FROM Student
WHERE Smer  IN   ('Informatika',   'Elektronika');
```

- Prikazati izgled tabele premija.

```
SELECT  *  FROM PREMIJA;
```

Broj Indeksa	Smer	Ispitni poeni	Datum upisa
114/10	Elektronika	73,7	31.3.2010
315/11	Elektronika	74,58	10.10.2011
221/10	Elektronika	75,9	9.10.2010

```
WHERE Predmet =   'Fizika');
```

1.3 Naredbe za definisanje podataka

- **Kreiranje tabele –CREATE TABLE osnovna sintaksa**

Nova (tabela) kreira se pomoću programskog paketa SQL naredbom CREATE TABLE. Opšti oblik ove naredbe glasi:

```
CREATE TABLE <naziv tabele>
        (<naziv kolone1> <tip podataka> [not null],
         <naziv kolone2> <tip podataka> [not null],
            . . .
        )
```

Ime relacije i kolona mora počinjati slovom engleskog alfabeta, može sadržati samo _ od specijalnih znakova i ne sme biti nijedna od ključnih reći programskog paketa SQL.. U jednoj bazi podataka ne mogu biti dva ista imena tabela u relaciji i atributa u istoj tabeli.

- **Indeksi**

U tabelama, podaci se smeštaju po redosledu njihovog unošenja. Postupak pretraživanja zahtevanog podatka u tabeli može dugo da potraje, pa se za rešavanje problema pretraživanja koriste specijalni tipovi tabela pod imenom indeksi, u kojima se nalaze adrese redova. Može se uporediti sa knjigom bez indeksa pojmova bi trebalo svaki put pretražiti od prve do zadnje stranice. Dobra strana je jer ubrzava pristup zapisima, a loša strana potreban je dodatan prostor za čuvanje indeksa.

Indeks je uređena datoteka čiji zapisi sadrže dva polja:

- polje indeksa (sadrži sortirane vrednosti indeksnog polja)
- polje pokazivača (sadrži adrese blokova na disku koje imaju vrednost indeksa).

Pretraživanje indeksne datoteke je binarno. Pronalazak traženog indeksa vodi nas do zapisa koji sadrži pokazivač na zapis s ostalim poljima o traženom entitetu.

⊙ Tipična sintaksa za kreiranje indeksa je*:*

```
CREATE [UNIQUE] INDEX <naziv indeksa>
ON <naziv tabele>   (<naziv kolone1>
                        [,  naziv kolone2,   ...]);
```

⊙ Indeks se izbacuje naredbom:

```
DROP INDEX <naziv indeksa>;
```

- **Šema baze podataka**

Šema je kolekcija svih objekata koji dele isti prostor imenovanja. Svaki objekat (tabela, pogled, itd.) pripada tačno jednoj šemi. Pod pripadnošću se ne podrazumeva fizička pripadnost, već hijerarhijska veza u kojoj, na primer, šema sadrži nula ili više tabela, a svaka tabela logički pripada tačno jednoj šemi.

Šema ima naziv, koji se može koristiti za kvalifikovanje naziva objekata koji pripadaju šemi.

⊙ Šema se kreira CREATE SCHEMA naredbom, iza koje slede naredbe za kreiranje objekata šeme, prvenstveno tabela, domena i pogleda*:*

CREATE SCHEMA <naziv seme> . . .
CREATE TABLE <naziv tabele$_1$> . . .
CREATE TABLE <naziv tabele$_2$> . . .
. . .

⊙ Šema se izbacuje DROP SCHEMA naredbom. Standard zahteva da se obavezno navede CASCADE ili RESTRICT način zadovoljavanja integriteta pri izbacivanju šeme.

Naredba DROP SCHEMA <naziv seme> CASCADE izbaciće šemu i sve objekte koji joj pripadaju.
Naredba:

DROP SCHEMA <naziv seme> RESTRICT

izbaciće šemu samo ako je prazna, odnosno ako se u njoj ne nalazi ni jedan objekat.
RESTRICT odredba sprečava izbacivanje šeme, ako je u njoj bar jedan objekat.

- **Pogled - VIEW**

Pogled je "prozor" kroz koji se vide podaci baze podataka onako kako su potrebni da ih vidimo. Pogled se umnogome ponaša kao tabela -sadrži zapise i polja koji se mogu prikazati u redovima i kolonama. Međutim, za razliku od tabela, pogledi ne čuvaju nikakve podatke. Pogledi sadrže instrukcije koje DBMS-u omogućavaju da pronađe potrebne podatke. Kada otvorite pogled, DBMS izvršava te instrukcije i na osnovu pogleda formira virtuelnu tabelu. Ova tabela postoji samo dok radite s njom - nikad se ne snima na čvrsti disk.

Pogled je virtuelna tabela i sa njim se radi gotovo kao sa baznom tabelom, mada nema svoje podatke i ne zauzima nikakav memorijski prostor. Preciznije rečeno, pogled se koristi kao bilo koja druga tabela pri izveštavanju.

Ažuriranje baze podataka preko pogleda ima brojna ograničenja.

- Ne može se vršiti ažuriranje preko pogleda ukoliko je pogled definisan nad više tabela. Takvo ažuriranje bi značilo da se jednom naredbom vrši ažuriranje više tabela baze podataka, što je suprotno definiciji INSERT, DELETE i UPDATE naredbi.
- Zatim se ne može vršiti ažuriranje preko pogleda ukoliko se u definiciji pogleda iza SELECT odredbe nalaze funkcije i aritmetički izrazi.
- Isto tako, ažuriranje se ne može vršiti ukoliko u definiciju pogleda nisu uključene sve NOT NULL kolone tabele nad kojom je pogled definisan.

Dakle, da bi mogli vršiti ažuriranje preko pogleda:

- pogled mora biti definisan nad jednom tabelom
- u definiciju pogleda moraju biti uključene sve NOT NULL kolone te tabele i
- kolone pogleda moraju sadržati samo prost, neizmenjen sadržaj kolona tabele nad kojom je pogled definisan.

Prednosti korišćenja pogleda:

- Jednostavnost korišćenja - uprošćava upite,
- Tajnost - moćan mehanizam kontrole pristupa podacima,
- Performanse - čuva se u kompajliranom obliku,

– Nezavisnost podataka - menjaju se definicije pogleda, a ne aplikacioni programi koji koriste podatke baze podataka preko pogleda.

⊙ Opšti oblik naredbe za kreiranje pogleda je*:*

```
CREATE VIEW  naziv-pogleda [(nazivi atributa pogleda)] AS
SELECT   . . .
FROM     . . . } bilo koja ispravna select naredba
[WITH [LOCAL|CASCADED]  CHECK OPTION]
```

Primeri:

- Kreirati pogled koji se sastoji od imena, šifara i smerova studenata koji slušaju predmet matematiku.

```
CREATE VIEW Matrmatika  AS
        SELECT  Broj_indeksa,   Student,   Smer
        FROM Student
        WHERE PredID = 1;
```

- Kreirati pogled koji će davati informacije o predispitnim obavezama i dodatnim poenima studenata bez navođenja imena studenata:

```
CREATE VIEW  POENI  AS
SELECT Broj_indeksa, PredID, Ispitni_poeni, Dodatni_poeni
FROM  Student;
```

Pogled omogućava sledeći pregled podataka.

```
SELECT  *
FROM POENI;
```

Broj Indeksa	PredID	Ispitni poeni	Dodatni poeni
113/12	7	75,5	20,5
114/10	9	67	23,5
110/11	8	85	16,5
211/10	4	77,5	20,5
220/09	5	72,5	23
315/11	4	68,5	20
221/10	6	69	19,5
510/08	4	71,5	24
110/11	8	67,5	22,5
103/10	7	79	18,5

⊙ Pored originalnih vrednosti kolona baznih tabela pogled može sadržati i izvedene vrednosti svojih virtuelnih kolona.

Primeri:

- Kreirati pogled sa atributima PredID, Smer, ukupan i srednji broj poeni studenata određenog smera koji slušaju određene predmete:

```
CREATE VIEW POENI
(PredID, Smer,   Ukupnopoena,   Sredljepoena)   AS
        SELECT PredID, Smer, SUM (Ispitni_poeni), AVG (Ispitni_poeni)
                FROM RADNIK
                GROUP BY  PredID,  Smer;
```

U ovom pogledu se može videti:

```
SELECT  *
FROM POENI;
```

PredID	Smer	Ukupnopoena	Srednjepoena
7	Menadžment	75,5	75,5
9	Elektronika	67	67
8	Menadžment	85	85
4	Informatika	77,5	77,5
5	Informatika	72,5	72,5
4	Elektronika	68,5	68,5
6	Elektronika	69	69
4	Menadžment	71,5	71,5
8	Menadžment	67,5	67,5
7	Informatika	79	79

1.4 Naredbe za kontrolne (upravljačke) funkcije

Naredbe za kontrolne (upravljačke) funkcije omogućuju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka:

- COMMIT (prenos dejstava transakcije na bazu podataka)
- ROLLBACK (poništavanje dejstava transakcije)
- GRANT (dodela prava korijenja sopstvene tabele drugim korisnicima)
- REVOKE (oduzimanje prava korijenja sopstvene tabele od drugih korisnika)

Moguće je kontrolisati vremenski trenutak prenosa dejstava transakcije na bazu podataka pomoću SQL naredbi COMMIT i ROLLBACK.

Pod transakcijom se podrazumevaju sve operacije (INSERT, UPDATE, DELETE) od poslednjeg prenosa dejstava na bazu, tj. od poslednjeg izdavanja naredbe COMMIT. Naredbom COMMIT se, dakle, prenose dejstva transakcije na bazu podataka.

Naredbom ROLLBACK poništavaju se sva dejstva od poslednjeg izdavanja naredbe COMMIT. Operacije koje potencijalno narušavaju referencijalni integritet su:

- brisanje (DELETE) ili
- izmena (UPDATE) redova referencirane tabele.