FER2.net

Tutorial za 1. međuispit iz Baza podataka

web verzija tutoriala:
http://www.fer2.net/showthread.php?t=10526

autor: **christair**
http://www.fer2.net/private.php?do=newpm&u=755
christair@nescume.com

Sadržaj:

1.1 - Općenite informacije o SQL-u te tipovima podataka
2.1 - Relacijski model i dizajn baze podataka
3.1 - SELECT naredba: osnovni oblik i uvjet
3.2 - SELECT naredba: agregatne funkcije i grupiranja
3.3 - SELECT naredba: izjednačavanje atributa i muka po JOIN-ovima
4.1 - Objašnjeni primjeri zadataka

1.1 - Općenite o SQL-u te tipovima podataka

Opće znanje SQL-a i DBMS-ova

SQL (**S**tructured **Q**uery **L**anguage) je skriptni jezik te služi za upravljanje bazom podataka. Svi veliki **DBMS**-ovi (**D**ata**B**ase **M**anagement **S**ystem, ili SUBP na hrvatskom) poput Oracle, MS SQL Server, MySQL, PostgreSQL, DB2, Informix, Sybase, Access i drugi oslanjaju se na SQL za izvršavanje upita/naredbi prema bazi.

S obzirom da je SQL standardiziran prema ANSI-ju, postoje razlike u inačicama SQL-a u različitim DBMS-ovima. Osnovni set naredbi (SELECT, UPDATE, DELETE, INSERT i ostali) gotovo se jednako ponaša u svim DBMS-ovima, dok su razlikuju većinom u količini te nazivima dodatnih ugrađenih SQL funkcija.

Na kolegiju Baze podataka radimo u IBM **Informix** DBMS-u, koji za SQL koristi **Transact-SQL** (T-SQL) varijantu SQL jezika, koja se može naći još u MS SQL Server i sličnim DBMS okruženjima.

Značajke SQL-a

SQL je **case insenstive** što znači da upiti nisu osjetljivi na mALa I vELikA sLovA i to se odnosi na kompletan SQL kod koji napišemo osim na **nizove znakova** koje označavamo s jednostrukim navodnicima - 'niz znakova'. Nizovi znakova jesu osjetljivi na mala i velika slova i tu moramo paziti.

SQL je također **jezik slobodnog formata naredbi** što znači da će iduća dva koda biti sintaktički ispravna i davati isti rezultat:

```
inSErT INtO oBavIJesTi (oznAkA, tEkSt) VALuES (1,'Paziti na mala i velika
slova samo u znakovnim nizovima kao što je ovaj');
```

i

```
INSERT INTO
obavijesti ( oznaka, tekst )
VALUES
( 1, 'Paziti na mala i velika slova samo u znakovnim nizovima kao što je
ovaj' );
```

Termini korišteni u literaturi i zadatcima

Kako bi u potpunosti razumjeli što se od nas u zadatku traži, moramo prvo proći kroz najčešće korištene termine da bi prema istima znali postupiti sa SQL kodom.

relacija
Relacija je naziv za jednu tablicu koju dobijemo kao rezultat SQL upita. Da bi tablica bila relacija, moraju svi stupci biti imenovani i ne smije biti duplih naziva. Relacija je skup n-torki.

atribut
Atribut je naziv za svaki stupac u jednoj tablici. Nazivi atributa mogu biti jednaki ključnim riječima u SQL-u ako naziv atributa stavimo u uglate zagrade - [atribut].

n-torka
Svaki redak u tablici smatra se n-torkom.

projekcija
Projekcija je tablica nastala rezultatom SELECT DISTINCT upita kojoj su svi retci različiti, dakle nema duplikata u retcima. U literaturi projekcija se označava znakom pi.

selekcija
Selekcija je tablica nastala rezultatom SELECT upita koja može imati duplikate u retcima. U literaturi selekcija se označava malim znakom sigma - **σ**.

stupanj
Stupanj relacije je broj njezinih atributa. Označava se sa **deg(relacija)**.

kardinalnost
Kardinalnost relacije je broj njezinih redaka (n-torki). Označava se sa **card(relacija)**.

Kartezijev produkt
Operacija koja će "pomnožiti" dvije tablice, tj. spojiti će svaki redak prve tablice sa svakim retkom druge tablice. Broj redaka će biti jednak **card(tablica1)*card(tablica2)**, a broj stupaca (atributa) **deg(tablica1)+deg(tablica2)**.

Algebarske oznake iz literature

∪	unija (*union*)
∩	presjek (*intersection*)
\	razlika (*set difference*)
÷	dijeljenje (*division*)
π	projekcija (*projection*)
σ	selekcija (*selection*)
×	Kartezijev produkt (*Cartesian product*)
ρ	preimenovanje (*renaming*)
⊳⊲	spajanje (*join*)
	agregacija, grupiranje

Naredbe SQL-a

Naredbe SQL-a koje smo obradili u 1. bloku predavanja su:

SELECT
Vrši upit nad tablicama ili viewovima u bazi. Da bi naredba bila ispravna mora minimalno sadržavati SELECT, nazive atributa, FROM te naziv tablice. Najosnovniji oblik je:

```
SELECT * FROM naziv_tablice
```

U idućem poglavlju ćemo detaljno razložiti SQL naredbu te prikazati moguće varijante upita.

CREATE TABLE
Stvara novu tablicu u bazi. U 1. blicu je bilo pitanja koja su se odnosila na ovo naredbu pa bi bilo važno spomenuti kako naredba radi.

```
CREATE TABLE naziv_tablice
(
naziv_stupca tip_podatka(veličina_podatka)
)
```

Osnovni tipovi podatka u SQL-u su:

brojčani
int
smallint
tinyint
integer (isto što i int)
decimal
float
money

ostali
char(veličina)
varchar(veličina)
date
bit

Usporedbu i raspone brojčanih tipova podatka možete naći ovdje.

Primjer CREATE TABLE naredbe:

```
CREATE TABLE student
(

jmbag bigint,
ime varchar(30),
prezime varchar(50),
prosjek decimal,
ukupnoects int,
datumrod date,

)
```

2.1 - Relacijski model i dizajn baze podataka

O ovoj temi te o organiziranju i usklađivanju podataka u bazi bi se mogao raspisati do besvijesti, no to bi prelazilo opseg 1. međuispita, na kojem nas ne uče još kako organizirati vlastite podatke na optimalan način nego kako već dobivenima podacima manipulirati.

No, ono što je bitno za znati je **osnovna ideja usklađivanja informacija u bazi podataka**.

U jednoj "ćeliji", tj. zapisu koji pripada nekom retku i nekom stupcu mora se nalaziti samo jedan relevantni podatak. Tehnički ne postoji varijanta da jedan zapis (ćelija) u sebi sadrži niz (array) elemenata.

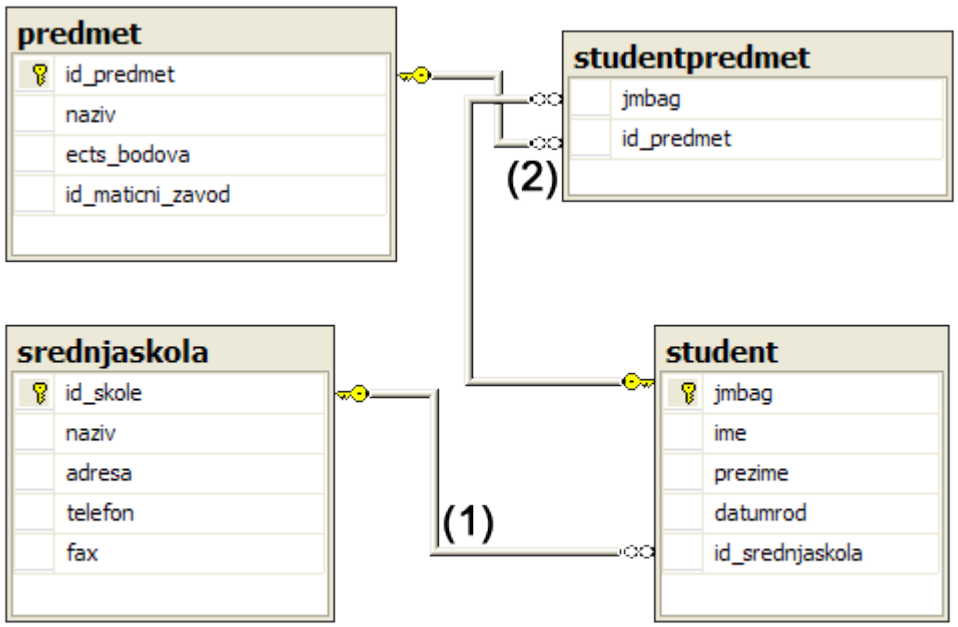
Cijela ideja relacijskog sustava (RDBMS - Relational DBMS) razvija se oko potonje činjenice. Zamislimo tablicu *student* u kojoj želimo osim osnovnih podataka koji određuju studenta, navesti i koje je sve predmete pojedini student upisao te u koju je srednju školu išao.

Naravno, mi možemo napraviti stupac *upisani_predmeti* tipa podatka znakovni niz i unutra navesti npr. "Vjerojatnost i statistika, Signali i sustavi, Uvod u teoriju računarstva" i sl. Također možemo dodati stupac *srednja_skola* i u nju upisati "I. tehnička škola TESLA".

No, što je tu loše? Loše je u slučaju upisanih predmeta što s tim podacima nije lako manipulirati niti u smislu upita baze, niti izmjene te dodavanja novih vrijednosti. U oba slučaja loše je to što dolazi do redudantnosti podataka, s obzirom da će u navednoj tablici *student* više studenata slušati isti predmet pa će jedan predmet biti "definiran" na više mjesta. Također bi više puta morali upisivati isto ime srednje škole s čime imamo neželjeno redudanciju i nemamo uniformnost sustava kojoj težimo. Ukratko: izazvalo bi velike probleme.

Kako se to onda radi?
Relacijama - referenciranjem vrijednosti preko ključeva. Napravit ćemo dvije nove tablice - *predmet* i *studentpredmet* i onda ćemo ih logičkim putem povezati.

Rezultat je:



Slika 1. Dizajn baze podataka

One-to-many i many-to-many veze (1tM i MtM)
Gornji dijagram prikazuje osnovna pravila povezivanja podataka u bazama. Najprije pogledajmo tablice *predmet, srednjaskola* i *student*. Svaka od navedenih ima jedan stupac označen s ključem. Ključ predstavlja Primary Key (PK) tablice te označava da sadržaj (retci) pod tim stupcem moraju biti unikatni. Drugim riječima, ako definiramo neki

stupac kao primarni ključ u tablici, to je onda identifikator tablice i ne može se dogoditi da dva retka imaju istu vrijednost unutar tog stupca.

Povezat ćemo *id_srednjaskola* (student) s *id_skole* (srednjaskola) (oznaka **(1)** na slici) te smo time napravili **one-to-many (1tM)** vezu. Takva vrsta veze je osnovna veza između 2 tablice te nam omogućuje uniformnost sustava - da su podaci koji se ponavljaju kroz zapise jedne tablice definirani na jednom mjestu u drugoj tablici (u našem slučaju u tablici *student* dogodit će se da je više studenata došlo iz iste škole).

U slučaju **(2)** sa slike imamo **many-to-many (MtM)** vezu za koju nam je uvijek potrebna dodatna tablica kako bi povezali vrijednosti. U ovoj varijanti nam nikakav zapis o predmetima pa čak niti neposredna referenca na zapis nije potrebna u samoj *student* tablici.
Student je jednoznačno definiran JMBAG-om koji predstavlja strani ključ (foreign key) u tablici *studentpredmet,* a predmet je isto tako jednoznačno definiran svojom šifrom te također u navedenoj tablici predstavlja strani ključ. Takva vrsta "spoja" nam omogućuje da jednom studentu preko njegovog JMBAG-a pridružimo više predmeta preko njihove šifre.

Navedeni način je 100% uniforman i nije redudantan iz razloga što su svi ključni podaci **definirani na jednom mjestu**. Pomoću 1tM i MtM veza logički povezujemo podatke, a u nastavku ćemo vidjeti kako to učiniti i tehnički kroz SQL kod.

Život uživo
"Baze podataka" je jedan od predmeta na kojem su stečena znanja ekstremno praktična te onaj tko položi baze podataka stvarno odmah može raditi na njima, što recimo nije slučaj s predmetom poput Elektronike :)

Pogledajte ovdje kako izgleda dizajn "ozbiljne" baze podataka i odmah ćete vidjeti paralelu s primjerom sa *Slike 1*. Ako mislite da je to puno i previše, pogledajte ovdje. Za utjehu ću samo reći da naš dragi forum ima doslovno duplo više tablica u bazi od ove baze s posljednjeg linka.

3.1 - SELECT naredba: osnovni oblik i uvjeti

SELECT naredbom prihvaćamo sadržaj jedne ili više tablica uz mogućnost dodavanja uvjeta, metoda spajanja, sortiranja rezultata i sl.

Osnovni oblik i operator *

```
SELECT * FROM nazivtablice
```

Oznaka * u kodu označava da želimo prihvatiti sve stupce iz tražene tablice. Ako napravimo upit:

```
SELECT * FROM tablica1, tablica2
```

dobit ćemo Kartezijev produkt te dvije tablice te će s obzirom na * biti izlistani svi stupci. Ako želimo ograničiti broj stupaca (atributa) koje želimo prikazati možemo napraviti ovakav upit:

```
SELECT tablica1.*, tablica2.atribut1, tablica2.atribut5 FROM tablica1,
tablica2
```

Kao rezultat ćemo dobiti sve atribute iz *tablica1* te *atribut1* i *atribut5* iz *tablica2*.
Služit ćemo se isključivo strogim imenovanjem atributa u selekciji... tj. uvijek ćemo navoditi *imetablice.imeatributa* nakon SELECT naredbe ukoliko će query biti kroz više tablica. Taj način pristupa nas osigurava od grešaka koje se vrlo lako mogu dogoditi zbog potencijalnih duplikatnih naziva atributa u različitim tablicama.

WHERE klauzula

Najčešće ćemo htjeti ograničiti prikaz rezultata na samo određeni sadržaj naših tablica... tu dolazi WHERE klauzula.

```
SELECT * FROM tablica WHERE neki_uvjet
```

Primjerice:

```
SELECT * FROM student WHERE prosjek >= 3.5
```

Tablice primjera neću prikazivati jer je ovo sve još prejednostavno. Uvjete uvijek možemo logički povezivati.
Primjerice

```
SELECT * FROM student WHERE prosjek >= 3.5 AND ostvarenoECTS > 40
```

```
SELECT * FROM student WHERE prosjek >= 3.5 OR ostvarenoECTS < 40
```

Kompliciranije uvjete možemo grupirati u zagrade kako bi smo bili potpuno sigurni da će se izvršiti redoslijedom koje želimo (da ne bismo zapeli u redoslijedu izvršenja logičih, matematičkih i relacijskih operatora).

```
SELECT * FROM tablica WHERE uvjet1 AND uvjet2 AND (uvjet3 OR
(uvjet4 AND uvjet5))
```

Građenje i sintaksa uvjeta

Uvjete ćemo najbolje naučiti na primjerima da uštedimo besmisleno teoretiziranje...

```
... WHERE prosjek >= 3.5
... WHERE prosjek > 3.5
```

```
... WHERE prosjek <= 4
... WHERE prosjek < 4
```

Za potonje primjere sve je jasno... no stvari se počinju razlikovati:

```
... WHERE prosjek = 4
... WHERE komentar = 'Neki znakovni niz'
... WHERE prosjek <> 3.5
... WHERE komentar <> 'Neki znakovni niz'
```

Važno: jednakost se označava sa znakom = a ne == kao u jezicima nalik C-u. Različitost označavamo sa operatorom <> umjesto != kako smo navikli. Oba = i <> mogu uspoređivati sve tipove podataka.

```
... WHERE prosjek BETWEEN 3 AND 4
```

Važno: u različitim DBMS-ovima BETWEEN-AND ima različito ponašanje s obzirom na rubne uvjete. U Transact SQL-u, a prema tome i u Informixu BETWEEN-AND predstavlja **zatvoreni interval**, tj. **uključuje rubne vrijednosti**.

Važno: Znakovne nizove također možemo uspoređivati koristeći < i > operatore. Veći od (>) znači **poslije** prema ASCII rasporedu. Mala slova su nakon velikih slova, oba setova slova su nakon brojeva. Dakle redoslijed je: brojevi, velika slova, mala slova.

Operator LIKE

Navedeni ćemo operator koristiti ako tražimo dio rezultata isključivo u nekom znakovnom nizu (varchar, nvarchar, text i ostali tipovi podataka). Još jednom, u znakovnim nizovima SQL-a imamo case sensitivity, odnosno moramo paziti prilikom uspoređivanja na mala i velika slova.

Uvjetom

```
... WHERE mjesto LIKE 'Zagreb'
```

postići ćemo isti rezultat kao da smo napisali

```
... WHERE mjesto = 'Zagreb'.
```

No osnovna ideja LIKE naredbe je traženje podniza u znakovnom nizu. Primjerice:

```
... WHERE mjesto LIKE 'Velika%'
```

Vratiti će sve retke u kojima atribut *grad* počinje na riječ "Velika". Znak **%** u LIKE naredbi mijenja **bilo koji znakovni niz**, bez obzira na njegovu duljinu. S obzirom da smo upit s lijeva ograničili na "Velika", a s desne strane dodali %, dobit ćemo rezultat koji mora počinjati na "Velika", a umjesto % može se nalaziti bilo koji, pa i prazan niz.

Ako imamo sadržaj tablice pod atributom **mjesto** jednak:

Velika Gorica
Velika Kopanica
Velika Mlaka
Velika
Zagreb
(...)

... navedenim uvjetom `WHERE mjesto LIKE 'Velika%'` zahvatit će se idući rezultati:

Velika Gorica
Velika Kopanica
Velika Mlaka
Velika
(...)

U rezultat će ući i mjesto **Velika** zato što % mijenja i prazan niz. Slično, možemo ograničiti niz s lijeve strane:

```
... WHERE mjesto LIKE '%Pušća'
```

Uz sadržaj atributa **mjesto** jednak:

Donja Pušća
Gornja Pušća
Zaprešić
(...)

Zahvatit će se samo rezultati **Donja Pušća** i **Gornja Pušća** jer smo upit proširili s lijeve strane. Kombiniranjem možemo znak % staviti i unutar niza. Primjerice:

```
... WHERE mjesto LIKE 'Vin%ci'
```

U navedenom će slučaju biti vraćena sva mjesta koja počinju na **Vin** i završavaju na **ci**, primjerice, **Vinkovci**. Također je moguće koristiti više znakova % unutar niza, primjerice:

```
... WHERE mjesto LIKE '%am%'
```

Navedena uvjet ostavit će samo one rezultate - ona mjesta koja u sebi, bilo gdje unutar niza (jer smo proširili niz s lijeve i desne strane), sadrže podniz **am**.

Operator LIKE podržava osim % još jedan wildcard. Znak _ (downslash, downstroke, podvlaka, kako god želite). Za razliku od % koji mijenja **0 <= N < besk.** znakova, znakom _ mijenjamo samo jedan znak u podnizu, točnije **N=1** (znakom _ ne možemo mijenjati prazan niz, mora striktno biti jedan znak).

Operator IN

Operatorom IN unutar uvjeta specificiramo točno određeni ili definiramo novi niz podataka (novi upit prema bazi). Na primjeru će se najbolje vidjeti:

```
SELECT * FROM dvorana WHERE kapacitet IN (30,35,40,100,120)
```

Rezulat ovog upita će biti svi zapisi o dvoranama koji imaju kapacitet točno 30, 35, 40, 100 ili 120 mjesta, dakle najmanje jedan od navedenih kapaciteta u zagradi mora biti ispunjen. Ostale vrijednosti će biti zanemarene. Ovaj ćemo operator koristiti samo u onim zadacima gdje u tekstu zadatka imamo neposredno zadano više vrijednosti, kao u potonjem primjeru.

U nizu ne moraju nužno biti samo brojčane vrijednosti. Sljedeći izraz također daje ispravan rezultat:

```
SELECT * FROM student WHERE nadimak IN ('rozi PsyBurn','ljubicasti
g09o','s.e.x.i l.e.x.i')
```

Napredniji oblik korištenja IN operatora je sljedeći:

```
SELECT * FROM nekatablica WHERE nekiatribut IN (SELECT atribut FROM
tablica WHERE nekiuvjet)
```

Navedenu funkcionalnost IN-a nećemo koristiti nego ćemo iste rezultate dobijati drugim načinima (JOIN-ovi i izjednačavanje atributa u WHERE-u).

Problematika NULL tipa podataka i operator IS

Naveli smo kako možemo uspoređivati znakovne nizove i brojčane tipove podataka koristeći klasične operatore algebre. No postoji problematika NULL tipa podatka. NULL u bazama služi za vrijednosti u kojima nema podatka.

Ne možemo uspoređivati vrijednosti na način

```
... WHERE atribut = NULL
... WHERE atribut <> NULL
```

jer NULL je specijalni tip podatka. Čak **ne vrijedi** relacija NULL=NULL.
Da bi ipak mogli upravljati podacima NULL tipa postoji operator IS. Ispravna uspoređivanja s NULL tipom glase ovako:

```
... WHERE atribut IS NULL
... WHERE atribut IS NOT NULL
```

Aliasi - alternativni nazivi tablica i atributa

U slučaju da unutar upita spajamo više tablica koje imaju barem jedan isti naziv stupca (atributa), a te stupce (s istim imenom) želimo prikazati u izlaznoj relaciji, morat ćemo dodati alias atributa.

```
SELECT atribut1 AS prviatribut, atribut2 AS drugiatribut FROM tablica
```

Na navedeni način u rezultirajućoj relaciji preimenovali smo atribute iz njihovih izvornih naziva u naše proizvoljne nazive. Ovo nam je obavezno raditi kad imamo više atributa istog imena kako bismo ih mogli razlikovati te da imenujemo atribute nastale rezultatom agregatnih funkcija ili CASE naredbi, no o tome više u idućem poglavlju.

Pri izradi refleksivnih upita (više o tome u idućim poglavljima) i još nekim situacijama, morat ćemo unutar jedne SELECT naredbe više puta pozivati istu tablicu s različitim "parametrima", odnosno različitim setom uvjeta kako bi smo dobili traženi rezultat.
Kako bi razlikovali tablice u navedenom slučaju, koristit ćemo aliase tablice.

```
SELECT tab1.atribut, tab1.drugiAtribut, tab2.nekiAtribut FROM tablica1 AS
tab1, tablica2 AS tab2 WHERE nekiuvjeti
```

Sortiranje zapisa

Jedna od osnovnih potreba prilikom prihvaćanja podataka je sortiranje istih. Na samom kraju svakog upita možemo dodati ORDER BY operator te navesti redoslijed atributa po kojima se sortira.
Postoje dvije metode sortiranja: uzlazno - ASC (od ASCending: A-Z, 0-besk.) i silazno - DESC (od DESCending: Z-A, besk.-0).

```
SELECT * FROM tablica ORDER BY atribut1
```

Ako ne navedemo metodu sortiranja, kao u potonjem primjeru, pretpostavlja se uzlazno (ASC) sortiranje. Sortiranje po više kriterija ćemo promotriti u sljedećem primjeru:

```
SELECT * FROM student ORDER BY
prezime DESC,
ime ASC,
mjestorodjenja
```

Konačna će se relacija prvo silazno sortirati po prezimenu studenta (Z-A). Zatim će se po imenu uzlazno sortirati samo oni zapisi studenata koji imaju isto prezime, a na kraju će se uzlazno (jer je to default ako ništa ne navodimo) po mjestu rođenja sortirati svi oni koji imaju jednako i ime i prezime.

Selektiranje prvih N i zadnjih N redova

Naredbom

```
SELECT FIRST 10 * FROM tablica
```

selektirati ćemo prvih 10 zapisa u našoj tablici. Naravno, FIRST se primjenjuje tek na rezultirajućoj projekciji, nakon što se izvrše svi uvjeti i spajanja.
Ukoliko uz FIRST trebamo koristiti još DISTINCT ili ALL ključne riječi, tada ih dodajemo na kraj, kao u primjeru:

```
SELECT FIRST 10 DISTINCT * FROM tablica
```

Ako želimo prihvatiti zadnjih 10 zapisa, ne postoji ključna riječ LAST nego ćemo koristiti FIRST, ali dodati ORDER BY i sortirati rezultate po silaznom (DESC) kriteriju atributa.

Unija

Kako bi spojili više SELECT naredbi u jedan rezultat možemo koristiti UNION i UNION ALL operatore. Primjerice:

```
SELECT * FROM zaposleniciZagreb
UNION
SELECT * FROM zaposleniciOsijek
```

Da bi se ova naredba uspješno izvršila navedene tablice moraju biti identične po svom **dizajnu** - moraju imati jednak broj atributa, iste tipove podataka nad atributima. **UNION naredbu nećemo koristiti** gotovo nigdje te sve upite koje želimo izvršiti lakše ćemo riješiti na druge načine.

Rad s datumima

Datume možemo uspoređivati na jednak način kao i brojeve i znakovne nizove operatorima > < >= <= = <>, itd. No ponekad želimo iz kompletnog datuma izvući samo dan, mjesec, godinu, dan u tjednu i sl.
To ćemo učiniti pomoću ugrađenih funkcija koje vraćaju cijele brojeve:

DAY(datum)	vraća dan u mjesecu
MONTH(datum)	vraća mjesec u godini
YEAR(datum)	vraća godinu
WEEKDAY(datum)	vraća indeks dana u tjednu.

Najmanji indeks koji WEEKDAY vraća je 0 i označava nedjelju (ne ponedjeljak!). 6 je najveći i označava subotu.
Oduzimanje datuma *datum1-datum2* vraća cijeli broj koji označava razliku u **danima** između ta dva datuma. SQL podržava ključnu riječ **TODAY** koja je objekt tipa podatka datum te označava datum na serveru (u Informixu u našem slučaju) u trenutku izvršavanja SQL skripte.

3.2 - SELECT naredba: agregatne funkcije i grupiranja, ugrađene fje

Agregatne funkcije

Dosad su naši upiti s obzirom na zadan uvjet filtrirali neodgovarajuće n-torke, a vraćali isključivo rezultate koje odgovaraju uvjetu i točan su sadržaj nekog od n-torki (redaka) tablice...
No što ako mi želimo postaviti drukčiji upit, npr. pronaći maksimalnu, minimalnu, prosječnu vrijednost nekog atributa u retcima ili jednostavno prebrojati retke, bilo sve ili uz određeni uvjet?
Tu nam kao rješenje dolaze takozvane agregatne funkcije SQL-a. U najosnovnijem obliku agregatne funkcije vraćati će skalarnu vrijednost, a sintaksa je sljedeća:

```
SELECT AgregatnaFunkcija(atribut) FROM tablica
```

Takvim pristupom, kao što je i rečeno dobivamo skalar: relaciju koja ima 1 redak i 1 stupac sadržaja kojeg izračuna agregatna funkcija. Stupcu će biti dodijeljeno ime naziva agregatne funkcije, osim ako aliasom stupca ne definiramo drugačije. Dakle, agregatne funkcije se obavljaju nad skupinom vrijednosti (n-torkama), ali vraćaju jednu, skalarnu vrijednost.

Pregled osnovnih agregatnih funkcija SQL-a:

AVG(atribut**)** - vraća prosječnu vrijednost atributa u relaciji
COUNT(atribut**)** - vraća ukupan broj redaka (bez NULL vrijednosti) navedenog stupca
COUNT(DISTINCT atribut**)** - vraća ukupan broj razlčitih redaka u relaciji
COUNT(*) - vraća ukupan broj redaka u relaciji
FIRST(atribut**)** - vraća prvi zapis navedenog atributa u relaciji
LAST(atribut**)** - vraća zadnji zapis navedenog atributa u relaciji
MIN(atribut**)** - vraća najmanji zapis navedenog atributa u relaciji
MAX(atribut**)** - vraća najveći zapis navedenog atributa u relaciji
STDEV(atribut**)** - vraća standardnu devijaciju atributa u relaciji
SUM(atribut**)** - vraća sumu svih zapisa (n-torki) po navedenom atributu u relaciji

Grupiranja

Ako pokušamo napisati izraz u stilu:

```
SELECT AgregatnaFunkcija(atribut1), atribut2 FROM tablica
```

Napravit ćemo jako veliku pogrešku i takav nam kod nikako neće raditi. Primjerice da imamo tablicu **studentocjena** koja sadrži zapise o studentu, predmetu i ocjeni studenta za navedeni predmet.
Izračunajmo općenito prosječnu ocjenu na svim predmetima:

```
SELECT AVG(ocjena) FROM studentocjena
```

No što ako želimo izračunati koja je prosječna ocjena na svakom predmetu? Trebali bi dodati u SELECT dio i atribut *predmet*, no već smo naveli u primjeru da to neće tako raditi. Rješenje leži u **grupiranju atributa**. Naredbom:

```
SELECT predmet, AVG(ocjena) FROM studentocjena GROUP BY predmet
```

... dobit ćemo upravo to što trebamo. U svakom retku ćemo imati predmet te prosječnu ocjenu za taj predmet.

Važno: Ako navodimo nakon SELECT izraza više atributa te agregatnu funkciju **neizostavno je** u GROUP BY izrazu dodati **sve atribute koji se nalaze nakon SELECT naredbe**.

Generalni izraz za navedeno upozorenje bi bio:

```
SELECT atribut1, atribut2, atribut3, AGR_FUNKCIJA(atribut4) FROM tablica
GROUP BY atribut1, atribut2, atribut3
```

Još je bitno za spomenuti da redoslijedom u GROUP BY izrazu ne mijenjamo odabir redaka nego samo njihov poredak u rezultirajućoj relaciji.

Zaključak: GROUP BY je dodan u SQL zato što agregatne funkcije vraćaju agregiranu vrijednost svih zapisa te je bez GROUP BY-a bilo nemoguće naći primjerice, prosječnu vrijednost svake grupe atributa zasebno.

Uvjetovanja agregatnih funkcija (HAVING)

Zamislimo tablicu prema sljedećem predlošku...

	tvrtka	prodaja
1	IBM	25000
2	Microsoft	12000
3	Barco	9000
4	IBM	3200

Želimo ispisati sve tvrtke koje kojima je ukupna prodaja veća 26000. Inicijalna ideja će nam biti da napišemo:

```
SELECT tvrtka, SUM(prodaja) FROM tablica GROUP BY tvrtka WHERE
prodaja>26000
```

Takav kod će dati **neispravan rezultat**, tj. tražiti će one rezultate koje inicijalno u tablici sadrže prodaju veću od 26000, a na navedenoj tablici sa slike nema niti jednog takvog.

HAVING uvjet je dodan u SQL iz razloga što WHERE nije mogao biti korišten nad agregatnim funkcijama te bi **bez HAVING-a bilo nemoguće vršiti uvjete nad agregatnim funkcijama**.

Ispravan kod bi glasio:

```
SELECT tvrtka, SUM(prodaja) FROM tablica GROUP BY tvrtka HAVING
SUM(prodaja) > 26000
```

Rezultantna relacija bi glasila:

	tvrtka	prodaja
1	IBM	28200

Važno! U HAVING bloku možemo izvršavati uvjete isključivo nad atributima navedenim u GROUP BY bloku i nad agregatnim funkcijama (kao primjeru, da ih ponovno napišemo).

Konačni oblik - shema SELECT naredbe

Iako je navedena slika daleko nepreciznija od onog što imamo u službenom šalabahteru, jednostavnije prikazuje mogućnosti SELECT naredbe.

Jedine stvarno potrebne ključne riječi na slici su SELECT i FROM, naravno uz nazive atributa i ime tablice. Sve ostale su opcionalne.

SELECT DISTINCT
    nazivi atributa AS *aliasatributa*
FROM
    nazivi tablica i/ili aliasa
JOIN
    naziv tablice AS *alias* ON *uvjet*
WHERE
    zadani uvjeti
GROUP BY
    nazivi atributa po kojima se grupira
HAVING
    naziv uvjeta po kojem se grupiranje izvodi
ORDER BY
    nazivi atributa i metode po kojoj se sortira

JOIN-ove ćemo raditi u zadnjem poglavlju teoretskog dijela tutoriala. Bitno je za znati da ih možemo nanizati koliko god želimo. Dakle, između FROM i WHERE može se nalaziti koliko god je potrebno JOIN-ova sa pripadajućim uvjetima. Ostale se ključne riječi mogu pojaviti samo jednom po SELECT naredbi.

Za **rekapitulaciju**, izlistati ćemo "srodne" izraze jednog upita:

SELECT (DISTINCT), FROM - osnovni dio svakog upita, označavamo želimo li samo različite n-torke uzeti u obzir

JOIN, ON - pridruživanje druge tablice na određeni uvjet

agregatna funkcija, GROUP BY, HAVING - agregatna fja vraća skalar, a ako dodamo atribut uz u ag. fju u SELECT dijelu moramo i odrediti GROUP BY. Operaciju HAVING koristimo kad želimo agregatnu funkciju ograničiti nekim uvjetom.

Uvod u ugrađene funkcije - izmjena vrijednosti atributa u projekciji

Dosad smo uvijek projekcijom vraćali sam sadržaj atributa bez manipuliranja tim podacima, no često je potrebno sadržaj nekog atributa prilagoditi potrebama projekcije.

Zamislimo da u tablici *proizvod* imamo atribut *cijena* čiji je sadržaj izražen u dolarima, a mi želimo cijenu prikazati u kunama, pri čemu nam je poznat tečaj za pretvorbu, primjerice 5.6 kuna za 1 dolar.
Jednostavnim upitom:

```
SELECT naziv, cijena * 5.6 FROM proizvod
```

kao rezultat dobit ćemo projekciju gdje je svaka vrijednost atributa cijena pomnožena sa konstantom 5.6. Sukladno tome, ispravno pretpostavljamo da će nad brojčanim tipovima podataka raditi i ostale matematičke operacije **+** , **-** , ***** , **/**.

Skalarne funkcije SQL-a: matematičke funkcije

Matematičke funkcije koristimo kao i ostale na način:

```
SELECT funkcija(atribut) FROM tablica
```

gdje je *funkcija* jedna od sljedećih matematičkih:

SIN(atribut) - sinus
COS(atribut) - kosinus
ASIN(atribut) - arcsin
ACOS(atribut) - arccos
TAN(atribut) - tangens
ATAN(atribut) - arctan
LOG10(atribut) - logaritam po bazi 10
POW(atribut, eksponent) - potenciranje na eksponent
MOD(atribut, broj) - ostatak pri cjelobrojnom dijeljenju s *broj*em
ROUND(atribut, broj) - zaokružava realni broj *atribut* na *broj* znamenaka

Skalarne funkcije SQL-a: brisanje praznina - TRIM

Ako koristimo znakovne nizove tipa podatka *varchar(duljina)* prilikom prihvaćanja atributa s takvim znakovnim nizom u projekciji, dogodit će se da će se ispisivati određen broj praznina dok se ne ispuni potpuna *duljina*.
Drugim riječima, ako nam je atribut tipa podatka *varchar(20)*, a jedan zapis glasi 'FER' prilikom prihvaćanja atributa ispisati će se 'FER_____' gdje oznake _ predstavljaju praznine (njih 17) dok se ne napuni 20 znakova.

Funkcijom **TRIM(ulazni niz)** riješit ćemo se svih nepotrebnih praznina prije početka prvog znaka u nizu i nakon završetka zadnjeg znaka. Dakle, TRIM('___FER2.net_____') gdje su _ praznine glasit će 'FER2.net'.

Funkciju TRIM koristimo u selekciji na način pokazan idućim primjerom:

```
SELECT TRIM(nazivProizvoda) FROM proizvod
```

Ugrađene funkcije SQL-a: spajanje stringova (concatenation)

Pretpostavimo da želimo u projekciji, uz cijenu proizvoda navesti i valutu, u našem slučaju "kn". Svakoj vrijednosti iz atributa *cijena* treba dodati znakovni niz "kn". To ćemo učiniti operatorom **||**.

```
SELECT naziv, (cijena * 5.6) || " kn" FROM proizvod
```

Potonji kod učinit će točno to što trebamo. Nema ograničenja koliko puta možemo iskoristiti konkatenaciju (spajanje) u izrazu za vrijednost jednog atributa.

Skalarne funkcije SQL-a: podnizovi - SUBSTRING

Malo kompliciraniji slučaj manipulacije znakovnim nizovima je vađenje podniza nekog niza. Idući primjer uključuje brisanje praznina (trim) i spajanje stringova (konkatenacija).

U tablici *student* imamo atribute *ime* i *prezime*. Projekciju želimo organizirati na takav način da imamo atribut koji će

sadržavati oblik prezimena, zareza, prvog slova imena i točke na kraju. Točnije, za studenta **Ivan Horvat** vrijednost atributa treba glasiti "**Horvat, I.**".

```
SELECT
TRIM(prezime)
|| ', '
|| SUBSTRING(TRIM(ime) FROM 1 FOR 1)
|| '.'
AS skracenoIme
FROM student
```

Ključni dio je SUBSTRING funkcija koju možemo poopćiti **SUBSTRING(znakovniNiz FROM x FOR y)**, gdje je **x** indeks slova od kojeg počinje rezultirajući podniz (prvi znak u nizu ima indeks **1**, ne 0 kao u C-olikim jezicima), a **y** je duljina niza od indeksa **x**.

Napomena: **y nije indeks niza s kojim želimo završiti podniz** nego duljina od znaka s indeksom **x**.

Skalarne funkcije SQL-a: duljina niza - CHARACTER_LENGTH

Funkcijom **CHARACTER_LENGTH(*znakovniNiz)*** dobit ćemo broj znakova koje taj znakovni niz sadržava. Nikakvih problema neće biti ako direktno unesemo konstanti znakovni niz u funkciju, npr.: CHARACTER_LENGTH('FER2.net') - rezultat je nedvojbeno 7, no postoji isti problem kao što smo naveli u prijašnjim retcima, s *varchar(duljina)* tipom podataka, tj. za bilo koji zapis čiji atribut ima potonji tip podataka, rezultat će biti konstanta vrijednosti *duljina*. Dakle, CHARACTER_LENGTH ćemo najćešće kombinirati s TRIM naredbom.

Primjerice, idemo ispisati imena svih studenata i duljine njihovih imena:

```
SELECT ime, CHARACTER_LENGTH(TRIM(ime))
FROM student GROUP BY ime
```

Skalarne funkcije SQL-a: uvjetovanja - CASE

Zadnja, no ne i najmanje bitna ugrađena funkcija koju ćemo opisati je CASE, pomoću koje možemo uvjetovati što će se ispisati u ovisnosti o ulaznom podatku.
Za primjer objašnjenja sintakse i funkcionionalnosti CASE naredbe zamislimo da imamo atribut *oznaka* brojčane vrijednosti. U tablici nam se pod tim atributom nalaze cijeli brojevi i NULL vrijednosti. Za sve parne brojeve želimo ispisati "paran", za sve neparne "neparan", a za sve NULL-ove "nema podatka".

To ćemo učiniti koristeći sljedeći kod:

```
SELECT

     CASE

          WHEN MOD(oznaka,2)=0 THEN 'paran'
          WHEN MOD(oznaka,2)=1 THEN 'neparan'
          WHEN oznaka IS NULL THEN 'nema vrijednosti'

     END

FROM student
```

S obzirom da za ispitivanje parnosti moramo koristiti ugrađenu funkciju, nemoguće je na jednom mjestu nju definirati i onda samo ispitivati brojeve, kao što je to slučaj u sljedećem (nevezanom) primjeru, gdje koristimo jednostavni oblik CASE-a:

```
SELECT

     CASE broj

          WHEN 0 THEN 'nula'
          WHEN 1 THEN 'jedan'
          ELSE 'neki drugi broj'

     END

FROM tablica
```

na potonjem primjeru upoznali smo se i sa ispisivanjem svih nenavedenih mogućnosti - ELSE. Trebamo pripaziti na to da nam atribut pod kojim će biti izlistani rezultati neće imati naziv ako mu ne dodijelimo alias koristeći AS. Tada CASE ne završavamo samo s END već s **END AS *alias***.

Važno: Kod nekih ugrađenih funkcija nije opisana njihova potpuna funkcionalnost!

3.3 - SELECT naredba: izjednačavanje atributa i muka po JOIN-ovima

Stop right here...

Ako niste pročitali poglavlje **2.1 - Relacijski model i dizajn baze podataka**, lijepo odskrolajte gore i pročitajte. Tek ćemo onda pričati o izjednačavanju atributa i JOIN-ovima te podvrstama JOIN-ova.

... and here!

Od sada pa do kraja tutoriala služit ćemo se primjerima iz FER-ove **studAdmin** baze podataka. Referencu na strukturu te baze te popis atributa i pripadajuće tipove podataka u tablicama možete downloadati ovdje (obratiti pažnju na .doc datoteku iz arhive):

Opis studAdmin baze podataka

- **180.75 KiB** - *popis tablica i atributa u studAdmin bazi podataka*

To također znači da sve navedene kodove možete testirati lokalno u ADS-u / Informixu.

Izjednačavanje preko WHERE-a

Primjer 1: Ispisati ime i prezime studenta te dodatno naziv mjesta u kojem taj student živi.

```
SELECT

      student.ime,
      student.prezime,
      mjesto.nazmjesto

FROM

      student,
      mjesto

WHERE student.pbrstan=mjesto.pbr
```

Da smo izostavili WHERE dobili bi Kartezijev produkt navedenih tablica. No mi želimo povezati dvije tablice kroz navedeni uvjet `student.pbrstan=mjesto.pbr`.

Time smo obavili izjednačavanje "srodnih" vrijednosti preko poštanskog broja (primarni ključ tablice *mjesto* (*pbr*) strani je ključ tablici *student* (*pbrstan*) te je time to one-to-many veza) kroz dvije tablice i kao rezultat imamo skup koji sadrži samo one n-torke gdje je poštanski broj iz tablice student jednak poštanskom broju u tablici mjesto.

Drugim riječima, neće se dogoditi da se u tom izjednačenju prikažu n-torke studenata kojima nije definiran poštanski broj stanovanja (*pbrstan* je NULL) ili n-torke mjesta u kojima ne stanuje niti jedan student. Dakle, rezultat je točno onaj skup koji zadovoljava "lijevi" i "desni" uvjet, respektivno, u odnosu na prethodnu rečenicu.

Iz fragmenta dobivene relacije vidimo da nema NULL vrijednosti i nepotpunih podataka:

15	Zeljka	...	Ban	...	Zagreb	...
16	Vedrana	...	Radinović	...	Zagreb	...
17	Ivana	...	Braim	...	Zagreb	...
18	Tatjana	...	Vukić	...	Zagreb	...
19	Ana	...	Marčetić	...	Zagreb	...

Izjednačavanje primarnog ključa jedne tablice sa sekundarnim ključem druge je najbitnija akcija u povezivanju dvije tablice (Kartezijev produkt tablica u praksi se (gotovo) nikad ne koristi).

Pridruživanje JOIN-om

Drugi način kako možemo pridružiti jednu tablicu drugoj je koristeći naredbu **JOIN**, no za razliku od prethodnog primjera, gdje smo WHERE i pripadajući uvjet mogli izostaviti (i dobiti Kartezijev produkt), to kod JOIN-a nije slučaj, jer pridružiti jednu tablicu drugoj možemo samo **na određeni uvjet** (ili više uvjeta) - tj. nema mogućnosti ostvarenja Kartezijevog produkta.

```
SELECT

      student.ime,
      student.prezime,
      mjesto.nazmjesto

FROM student
JOIN mjesto ON student.pbrstan=mjesto.pbr
```

Najnormalnije možemo raditi aliase i više uvjeta, npr.:

```
      ... JOIN tablica AS aliasTablice ON uvjet1 AND uvjet2 AND (uvjet3 OR
uvjet4)
```

Važno: Naredba **JOIN** obavlja isti posao kao i **INNER JOIN**. Kod pridruživanja JOIN-om ili INNER JOIN-om dobit ćemo **identične rezultate kao kod izjednačavanja u WHERE-u**.

Pridruživanje LEFT JOIN-om

Kod izjednačavanja uz WHERE spomenuli smo da su i "lijevi" i "desni" uvjet zadovoljeni, tj. da se ne mogu prikazati n-torke studenata kojima nije definiran p.br. stanovanja (*pbrstan* je NULL) za **lijevi** uvjet te da se ne mogu prikazati n-torke mjesta u kojima ne stanuje niti jedan student (za neki poštanski broj, odnosno mjesto, nema reference među stanovanjima studenata) - **desni** uvjet.

LEFT JOIN će nam upravo omogućiti da prikažemo rezultate koji zanemaruju samo lijevi, ali ne i desni uvjet.

```
SELECT

      student.ime,
      student.prezime,
      mjesto.nazmjesto

FROM student
LEFT JOIN mjesto ON student.pbrstan=mjesto.pbr
```

Izvršavanjem navedenog upita nad *studAdmin* bazom, dobit ćemo relaciju čiji fragment možemo vidjeti ovdje:

37	Hrvoje	...	Vego	...	(null)		
38	Tomislav	...	Makar	...	(null)		
39	Igor	...	Cigić	...	(null)		
40	Marko	...	Čop	...	(null)		
41	Neven	...	Kolarić	...	Vrbovec	...	
42	Domagoj	...	Topić	...	Zagreb	...	
43	Marko	...	Ravnik	...	(null)		
44	Ante	...	Miličić	...	Zagreb	...	
45	Branimir	...	Dragičević	...	Zagreb	...	

Jasno je vidljivo da smo dobili n-torke studenata kojima je pridruženo mjesto stanovanja, kao i u prethodnim primjerima, ali i n-torke studenata kojima nije pridruženo mjesto stanovanja, odnosno zanemaren je samo lijevi uvjet.

Opaska: Izlistani su svi studenti, ali ne i sva mjesta. Dakle, "vodeća" tablica je *student*. NULL vrijednosti imamo samo s "desne strane".

Važno: Naredba **LEFT JOIN** obavlja isti posao kao i **LEFT OUTER JOIN**.

Pridruživanje RIGHT JOIN-om

Sad kad smo utvrdili LEFT JOIN, zadaća RIGHT JOIN-a je potpuno jasna. RIGHT JOIN-om ćemo ispuniti sve što i običnim JOIN-om (ili WHERE-om uz izjednačavanje), ali ćemo zanemariti samo desni uvjet.

Promjenom samo jedne ključne riječi u odnosu na prethodni kod (LEFT u RIGHT) dobivamo:

```
SELECT

student.ime,
student.prezime,
mjesto.nazmjesto

FROM student
RIGHT JOIN mjesto ON student.pbrstan=mjesto.pbr
```

Izvršenjem tog koda rezultat je sasvim jasan:

329	Jan	...	Čemeljić	...	Ivanić-Grad	...	
330	(null)		(null)		Novoselec	...	
331	Neven	...	Kolarić	...	Vrbovec	...	
332	(null)		(null)		Gradec	...	
333	Danijel	...	Milin	...	Velika Gorica	...	
334	Bruno	...	Travalja	...	Velika Gorica	...	
335	(null)		(null)		Novo Čiče	...	
336	(null)		(null)		Sveti Martin pod Okićem	...	

Dobili smo n-torke mjesta i među njima nema NULL-ova, ali smo dobili i n-torke studenata kojima je pridruženo neko od navedenih mjesta. Nazivi mjesta se ponavljaju (primjer redak 333 i 334) ako i samo ako više studenata živi u istom mjestu (jer ih moramo izlistati sva mjesta).

Opaska: Izlistani su sva mjesta, ali ne i svi studenti. Dakle, "vodeća" tablica je *mjesto*. NULL vrijednosti imamo samo s "lijeve strane".

Važno: Naredba **RIGHT JOIN** obavlja isti posao kao i **RIGHT OUTER JOIN**.

Potpuno pridruživanje - FULL JOIN

Logički se nameće da će FULL JOIN biti rezultat kombinacije LEFT i RIGHT JOIN-a te da će sadržavati sve što oni sadržavaju.

```
SELECT

      student.ime,
      student.prezime,
      mjesto.nazmjesto

FROM student
FULL JOIN mjesto ON student.pbrstan=mjesto.pbr
```

Izvršimo SQL kod i dobijemo...

519	Sanja	...	Lalić	...	(null)	
520	Ana	...	Žakula	...	(null)	
521	Marija	...	Zanketić	...	Zagreb	...
522	Nikolina	...	Smilović	...	Zagreb	...
523	Gordana	...	Martinković	...	(null)	
524	Mia	...	Zekanović	...	Zagreb	...
525	(null)		(null)		Daruvar	...
526	(null)		(null)		Veliki Bastaji	...

Primjećujemo odlike LEFT joina, RIGHT joina te "početne" situacije kad su oba - lijevi i desni uvjet zadovoljeni.

Opaska: Izlistani su svi studenti koji imaju definirano mjesto, svi studenti koji nemaju definirano mjesto (LEFT) te sva mjesta za kojih nema "para" sa studentom (RIGHT). NULL vrijednosti imamo s "lijeve i desne strane".

Važno: Naredba **FULL JOIN** obavlja isti posao kao i **FULL OUTER JOIN**.

Refleksivno pridruživanje

Refleksivno pridruživanje je jednostavno posljedica načina razmišljanja kako povezati podatke istog tipa u **hijerarhijsku ovisnost**, a koristeći dosad navedene načine pridruživanja (ne radi se o novoj vrsti JOIN-a).

Primjer u dosadašnjim materijalima je bila hijerarhijska ovisnost u visokom školstvu: unutar Sveučilišta djeluju Fakulteti, unutar njih Zavodi i/ili Katedre, a potonji/e djeluju u interesnim Grupama.

Drugi bi primjer bila ovisnost foruma i podforuma i podpodforuma na nekom bulletin board sistemu poput FER2.net-a.

Treći primjer, gotovo identičan drugom, bi bila hijerarhijska organizacija repozitorija na *materijali.fer2.net* sustavu. Glavni se repozitorij dijeli na semestre, semestri na kolegije, kolegiji na kategorije poput "domaće zadaće", "međuispiti", "predavanja"...

S obzirom da najbolje služi svrsi refleksivnog pridruživanja, objasnit ću refleksivno pridruživanje na primjeru repozitorija *materijali.fer2.net* sustava.

Ovako izgleda fragment tablice *repozitorij*:

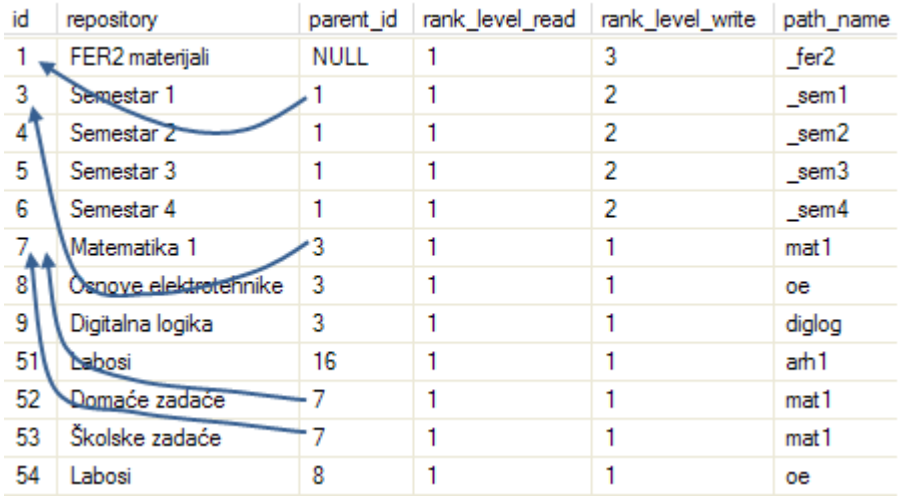
id	repository	parent_id	rank_level_read	rank_level_write	path_name
1	FER2 materijali	NULL	1	3	_fer2
3	Semestar 1	1	1	2	_sem1
4	Semestar 2	1	1	2	_sem2
5	Semestar 3	1	1	2	_sem3
6	Semestar 4	1	1	2	_sem4
7	Matematika 1	3	1	1	mat1
8	Osnove elektrotehnike	3	1	1	oe
9	Digitalna logika	3	1	1	diglog
51	Labosi	16	1	1	arh1
52	Domaće zadaće	7	1	1	mat1
53	Školske zadaće	7	1	1	mat1
54	Labosi	8	1	1	oe

Atributi:

- *id* je primarni ključ, identifikator repozitorija
- *parent_id* je glavni atribut nakon primarnog ključa. Dosad smo referencirali samo vrijednost iz druge tablice (primarni ključ jedne sa stranim ključem druge tablice). U ovom slučaju pozivamo se na primarni ključ iz iste te tablice
- *rank_level_read*, *rank_level_write* i *path_name* nisu bitni za analizu refleksivnog pridruživanja, imaju upotrebu unutar sustava Materijala, a prikazani su iz razloga da se primjeti kako svaki repozitorij ima istu "strukturu podataka", tj. definiran je istim atributima

Različite nijanse boja redaka predstavljaju različite dubine repozitorija, gdje je najsvjetlija nijansa plave najniža dubina.

Na idućoj slici vidjeti ćemo hijerarhijsku ovisnost *Matematike 1* i njezinih podrepozitorija unutar Sustava:



Domaće zadaće (**52**) i *Školske zadaće* (**53**) imaju *parent_id* (**7**). Pogledamo li (**7**) vidjet ćemo repozitorij *Matematike 1* koji svoju referencu, svog roditelja ima u (**3**).
Na (**3**) nalazi se *Semestar 1* koji svojeg roditelja ima u ID-u (**1**). Potonji je početni identifikator te nema roditelja - vrijednost za njegov *parent_id* mora biti NULL.

Sada ćemo na našoj *studAdmin* bazi izvršiti sljedeći komad koda za primjer:

```
SELECT
org1.nazorgjed AS jedinica,
org2.nazorgjed AS nadredjenaJedinica
FROM
orgjed AS org1
LEFT JOIN
orgjed AS org2 ON org1.sifnadorgjed=org2.siforgjed
```

Fragment relacije kad se izvrši navedeni kod:

	jedinica	nadredjenajedinica
1	Sveučilište u Zagrebu ...	(null)
2	Sveučilište u Osijeku ...	(null)
3	Sveučilište u Rijeci ...	(null)
4	Sveučilište u Splitu ...	(null)
5	Farmaceutsko-biokemijski fakultet ...	Sveučilište u Zagrebu ...
6	Geodetski fakultet ...	Sveučilište u Zagrebu ...
7	Filozofski fakultet u Rijeci ...	Sveučilište u Rijeci ...
8	Ekonomski fakultet ...	Sveučilište u Osijeku ...
9	Kemijsko-tehnološki fakultet ...	Sveučilište u Splitu ...
10	Edukacijsko-rehabilitacijski fakul...	Sveučilište u Zagrebu ...
11	Fakultet političkih znanosti ...	Sveučilište u Zagrebu ...
12	Fakultet organizacije i informatik...	Sveučilište u Zagrebu ...
13	Pravni fakultet ...	Sveučilište u Splitu ...
14	Fakultet elektrotehnike, strojarst...	Sveučilište u Splitu ...
15	Fakultet za turizam i vanjsku trgo...	Sveučilište u Splitu ...

Vidimo da Sveučilišta imaju NULL vrijednosti za naređenu organizacijsku jedinicu, dok su isti nadređena jedinica Fakultetima. Pri dnu gore navedene relacije nalaze se i Zavodi kojima su nadređene jedinice Fakulteti, te Grupe kojima su nadređene jedinice Zavodi.

Opaska: da smo koristili JOIN umjesto LEFT JOIN-a nebi vidjeli prva 4 zapisa sa slike, tj. one n-torke koje nemaju zadanu nadređenu jedinicu (*sifnadorgjed* je NULL).

4.1 - Objašnjeni primjeri zadataka

Za kraj tutoriala, objasnit ću rješenja nekoliko različitih tipova zadataka i prokomentirati ih:

Primjer 1

Izvor: zadatak 2. domaće zadaće

Znanja: manipuliranje datumima, promjene sadržaja atributa, aliasi, zaokruživanje realnih brojeva

Tekst zadatka:

Potrebno je napraviti analizu kako bi se promijenile plaće nastavnika sa stažom od barem 5 godina ako bi im se koeficijent zaokružio. Naime, plaća nastavnika se određuje kao koeficijent * 1000 kn.

Tako, na primjer, nastavnik s koeficijentom:

5.3 ima plaću 5,300 kn

5.6 ima plaću 5,600 kn

Ukoliko bi se zaokružili koeficijenti:

prvi nastavnik bi imao plaću 5,000 kn (razlika = -300)

drugi nastavnik bi imao plaću 6,000 kn (razlika = 400)

Napisati upit koji će vratiti **šifru**, **ime**, **prezime** i **razliku** za svakog nastavnika koji radi u nekoj od organizacijskih jedinica **neposredno podređenih** Sveučilištu u Zagrebu, a čijeg datuma zaposlenja do danas je prošlo **najmanje pet godina**.

Atribute nazvati *sifnastavnik*, *ime*, *prezime* i *razlika* (za razliku ispisivati samo broj bez oznake kuna, kao što je gore u primjeru).

U prvoj polovici posla odrediti ćemo atribute i izračunati razliku:

```
SELECT

nastavnik.sifnastavnik,
nastavnik.ime,
nastavnik.prezime,
ROUND(ROUND(koef)*1000-koef*1000) AS razlika
```

Prva tri atributa već imaju nazive kakvi trebaju biti u izlaznoj relaciji, dok posljednji atribut treba prilagoditi da bi se izračunala razlika.

```
ROUND(koef)*1000 - koef*1000
```

Nakon ovog fragmenta koda dobit ćemo razliku koju treba prikazati, ali to treba još zaokružiti još jednim ROUND-om, te napraviti alias koji se zove *razlika*:

```
ROUND(ROUND(koef)*1000-koef*1000) AS razlika
```

Sada na drugi dio koda...

```
FROM nastavnik
JOIN orgjed ON nastavnik.siforgjed=orgjed.siforgjed
WHERE

TODAY-datumzaposlenod >= 5*365
AND orgjed.sifnadorgjed=9996
```

Glavni dio informacija hvatamo iz tablice *nastavnik*, zato ćemo nju staviti u FROM, a njoj ćemo pridružiti tablicu *orgjed* preko šifre organizacijske jedinice. Imati ćemo 2 uvjeta:

```
TODAY-datumzaposlenod >= 5*365
```

Ovo je jedini mogući način kako da ostvarimo dio zadatka "... a čijeg datuma zaposlenja do danas je prošlo **najmanje pet godina**". Iako ni ovo nije strogo gledano točno, s obzirom da prijestupne godine imaju 366 dana, smatrat ćemo ovu izvedbu zadovoljavajućom.

Alternativni način bi bio napisati:

```
YEAR(TODAY) - YEAR(datumzaposlenod) >= 5
```

... no tu bi problem bio u rubnim vrijednostima, odnosno zaokruživanju, s obzirom da to da se netko može zaposliti na samom kraju npr. 2002. te bi danas (2007) - 2002 bilo jednako 5 (što bi zadovoljavalo uvjet >=5), ali je prošlo efektivno 4 godine, jer će zaposlenik punih 5 godina navršiti tek na samom kraju 2007.

U drugom dijelu uvjeta (`AND orgjed.sifnadorgjed=9996`) limitiramo projekciju samo na one rezultate gdje je nadređena organizacijska jedinica Sveučilište u Zagrebu.

Kompletan kod glasi:

```
SELECT

     nastavnik.sifnastavnik,
     nastavnik.ime,
     nastavnik.prezime,
     ROUND(ROUND(koef)*1000-koef*1000) as razlika

FROM nastavnik
JOIN orgjed ON nastavnik.siforgjed=orgjed.siforgjed
WHERE

     TODAY-datumzaposlenod >= 5*365
     AND orgjed.sifnadorgjed=9996
```

Primjer 2

Izvor: zadatak 2. domaće zadaće
Znanja: manipuliranje datumima, promjene sadržaja atributa, aliasi, rad sa stringovima (spajanje i brisanje praznina - trim)

Tekst zadatka:

Za parove studenata muškog spola koji su rođeni istog mjeseca i istog dana u mjesecu bilo koje godine u istom mjestu, ispisati prezime i inicijal imena te datum rođenja u sljedećem obliku:

student1	datRod1	student2	datRod2
Florijan, S.	04.10.1983	Stevančević, D.	04.10.1984
Florijan, S.	04.10.1983	Karić, I.	04.10.1986
Frančišković, V.	11.03.1982	Barešić, M.	11.03.1985
	...	...	...

Nazive stupaca u listi izlaznih rezultata imenovati u skladu s gornjim predloškom. Iz rezultata izbaciti n-torke u kojima se isti student pojavljuje pod stupcima sa sufiksom 1 i sufiksom 2. Obratiti pažnju na formatiranje znakovnih nizova koji se ispisuju pod stupcima *student1* i *student2* - prezime bez pratećih praznina odvojeno je zarezom od inicijala imena iza kojeg je navedena točka.
Poredak zapisa u rezultatu nije bitan.

Pregled onog što moramo učiniti:

- imena studenata moramo TRIM-ati kako bi dobili ime bez završnih praznina, kako bi mogli formirati traženi oblik za atribut
- s obzirom da prema slici relacije u zadatku vidimo kako trebamo ispisivati n-torke čiji sadržaj iz iste tablice 2 puta u jednom retku, morat ćemo JOIN-om pridružiti istu tu tablicu (student), ali pod drugim aliasom kako bi ih mogli razlikovati
- ugrađenim funkcijama datuma razlikovati ćemo dane i mjesece iz datuma rođenja studenata

Kompletan kod glasi:

```
SELECT

      TRIM(student.prezime) || ', ' || SUBSTRING(student.ime FROM 1 FOR 1) ||
      '.' AS student1,
      student.datrod AS datRod1,
      TRIM(Kstudent.prezime) || ', '|| SUBSTRING(Kstudent.ime FROM 1 FOR 1) ||
      '.' AS student2,
      Kstudent.datrod AS datRod2

FROM student
JOIN student AS Kstudent ON

student.spol='M' AND Kstudent.spol='M'
AND MONTH(student.datrod) = MONTH(Kstudent.datrod) AND DAY(student.datrod) =
DAY(Kstudent.datrod) AND student.pbrrod=Kstudent.pbrrod AND
student.jmbag<>Kstudent.jmbag
```

Primjer 3

Izvor: zadatak 2. domaće zadaće
Znanja: uvjetovanje operatorom LIKE, pridruživanje više tablica JOIN-om, izbacivanje duplikata

Tekst zadatka:

Ispisati oznaku i kapacitet svih dvorana kojima oznaka počinje slovom "A". Za one dvorane u kojima se izvodila nastava za grupe kojiam oznaka počinje sa slovom "C", ispisati uz podatke o dvorani i akademsku godinu kada se ta nastava odvijala, u suprotnom ispisati NULL vrijednost. Dodatno, ispisati i naziv predmeta iz kojeg se odvijala takva nastava, ukoliko je predmet nosio 5 ECTS bodova, inače ispisati NULL vrijednost. Iz ispisa izbaciti duplikate.

Primjer rezultata:

ozndvorana	kapacitet	akgodina	naziv
A102	60	2003	NULL
A109	40	2003	Telekomunikacijske mreže
A110	40	NULL	NULL
	...	...	...

Pregled onog što moramo učiniti:

potrebno je izbaciti duplikate iz ispisa, radimo SELECT DISTINCT na samom početku

osnovne informacije hvatamo iz tablice *dvorana*. Ostale tablice (*predmetgrupa*, *predmet*) JOIN-om ćemo pridružiti tablici *dvorana*.

potrebno je koristiti LIKE kako bi odredili s kojim slovom počinje neki znakovni niz

pridruživati ćemo tablicu *predmetgrupa* kako bi dobili akademsku godinu, i to LEFT JOIN kriterijem zbog kojeg će se ispisivati NULL vrijednosti ako nema pripadajuće dvorane

pridruživati ćemo tablicu *predmet* kako bi dobili naziv predmeta, LEFT JOIN kriterijem iz istog razloga kao za *predmetgrupa*

sve zajedno uzimamo u obzir samo ako oznaka dvorane počinje s "A", što ćemo navesti u WHERE dijelu osnovnog upita

Kompletan kod glasi:

```
SELECT DISTINCT

     dvorana.ozndvorana,
     dvorana.kapacitet,
     predmetgrupa.akgodina,
     predmet.naziv

FROM dvorana
LEFT JOIN predmetgrupa ON

     predmetgrupa.ozndvorana=dvorana.ozndvorana
     AND ozngrupa LIKE 'C%'
```

```
LEFT JOIN predmet ON

     predmet.sifpredmet=predmetgrupa.sifpredmet
     AND predmet.ectsbod=5

WHERE dvorana.ozndvorana LIKE 'A%'
```

Primjer 4

Izvor: zadatak 2. domaće zadaće

Znanja: agregatne funkcije (s grupiranjem i uvjetima za agregaciju)

Tekst zadatka:

Za svaki predmet i akademsku godinu ispišite naziv predmeta, šifru predmeta, akademsku godinu te ukupan kapacitet dvorana u kojima se taj predmet te akademske godine održavao (stupac nazovite *uk_kapacitet*). Ispisati zapise samo za predmete koje nose barem 6 ECTS bodova i za koje je ukupni kapacitet veći od 200.

Pregled onog što moramo učiniti:

zadatak je prilično jednostavan, no jedna je catchy stvar... radi se o uvjetovanju za ukupan kapacitet koji mora biti veći od 200 - taj uvjet ne možemo ispitavati u WHERE dijelu, nego isključivo kao posljedicu agregatne funkcije SUM, nakon HAVING ključne riječi

pridruživanje tablica izvršit ćemo JOIN-ovima

trebamo 2 JOIN-a, agregatnu funkciju SUM, GROUP BY i HAVING...

Kompletan kod glasi:

```
SELECT

     predmet.naziv,
     predmet.sifpredmet,
     predmetgrupa.akgodina,
     SUM(dvorana.kapacitet) uk_kapacitet

FROM predmetgrupa
JOIN predmet ON

     predmet.sifpredmet=predmetgrupa.sifpredmet
     AND predmet.ectsbod>=6

JOIN dvorana ON predmetgrupa.ozndvorana=dvorana.ozndvorana
GROUP BY

predmet.naziv, predmet.sifpredmet, predmetgrupa.akgodina

HAVING SUM(dvorana.kapacitet) > 200
```