Универзитет у Крагујевцу
Факултет инжењерских наука

SQL

Увод

SQL је стандардни релациони упитни језик. Настао је Сан Хосеу у Калифорнији 1974. године. Када се надовеже на релацију, за једну н-торку у релацији каже се да представља један ред табеле. Ако се релација представи као табела, јасно је шта се подразумева под појмом колоне - све вредности у н-торкама релације које одговарају једном атрибуту. Постоје три групе категорија у које су сврстане **SQL наредбе**, а то су:

- **Наредбе за дефинисање података**

Пре почетка рада са базом података је неопходно дефинисати њену структуру, односно које табеле постоје, који атрибути постоје у табелама и ког су типа, која ограничења постоје унутар табела и између њих , које помоћне структуре (индекси) за убрзање приступа подацима постоје и за које табеле.

- **Наредбе за манипулисање (руковање подацима)**

Поред упита над базом података, којим се добијају жељене информације, неопходно је обезбедити и ажурирање базе података, односно унос, измену и брисање података.

- **Наредбе за контролне функције**

У свакој бази података је неопходно остварити контролу коју корисници имају приступ којим подацима и шта могу да раде са тим подацима

Наредбе за дефинисање података омогућују дефинисање објекта базе. Примери наредби ове категорије јесу:
- ✧ CREATE TABLE (креирање табеле базе података)
- ✧ CREATE VIEW (креирање виртуелне табеле - "погледа")
- ✧ CREATE INDEX (креирање индекса над комбинацијом колона табеле)
- ✧ DROP TABLE (избацивање табеле из базе података)
- ✧ ALTER TABLE (измена дефиниције табеле)

Наредбе за манипулисање (руковање) подацима омогућују ажурирање и приказ података базе, а то су:
- ✧ SELECT (приказ садржаја релационе базе података)
- ✧ UPDATE (измена вредности колона табеле)
- ✧ DELETE (избацивање редова табеле)
- ✧ INSERT (додавање редова постојећој табели)

Наредбе за контролне (управљачке) функције омогућују опоравак, конкурентност, сигурност и интегритет релационе базе података:

- ✧ GRANT (додела права коришћења сопствене табеле другим корисницима)
- ✧ REVOKE (одузимање права коришћења сопствене табеле од других корисника)
- ✧ COMMIT (пренос дејстава трансакције на базу података)
- ✧ ROLLBACK (поништавање дејства трансакције)

1999 SQL стандард разврстава наредбе у SQL 7 категорија. Основни разлог за другачије разврставање наредби је увођење нових концепата у SQL у складу са

развојем информатичке технологије и потреба да се постојеће наредбе прецизније групишу. Дефинисане су следеће категорије SQL наредбе:

- Наредбе за шему базе података

Оне се користе за креирање, измену и избацивање шема и објеката шема (CREATE, ALTER, DROP)

- Наредбе за податке

Оне се користе за приказ и ажурирање података базе (SELECT, INSERT, UPDATE, DELETE)

- Наредбе за трансакције

Оне се користе за стартовање, завршавање и постављање параметара за транцакције (COMMIT, ROLLBACK)

- Наредбе за контролу

Користе се за контролу извршавања секвенце SQL наредби (CALL, ROLLBACK)

- Наредбе за конекцију

Користе се за успостављање и прекидање SQL конекције (CONNECT, DISCONNECT)

- Наредбе за сесије

Оне се користе за постављање default вредности и других параметара SQL сесије (SET)

- Наредбе за дијагностику

Користе дијагностичке податке и сигнализирају изузетке у SQL рутинама (GET DIAGNOSTICS)

SQL типови података

- **INTEGER -** цео број са или без предзнака чији број цифара зависи од конкретне имплементације

- **FLOAT -** децимални број са или без предзнака чија прецизност (број значајних цифара) зависи од конкретне имплементације

- **DECIMAL [m[, n]] -** децимални број са или без предзнака и са укупно m цифара од чега су n децимале; максимални број цифара зависи од конкретне имплементације

- **BOOLEAN -** логички податак са вредношћу TRUE или FALSE

- **CHARACTER [[n]] -** или CHAR; низ знакова фиксне дужине; максимални број знакова зависи од конкретне имплементације; ако дужина није

наведена подразумева се 1; константе овога типа се пишу између једноструких наводника

- ✧ **CHARACTER VARYING [n] -** низ знакова промењиве дужине од 0 до n, слично стрингу у програмском језику C; максимални број знакова зависи од конкретне имплементације; константе овога типа се пишу између једноструких наводника

- ✧ **TIMESTAMP -** универзални податак "година - месец - датум - сат - минут - секунда..." где најмања резолуција времена зависи од конкретне имплементације

- ✧ **DATE -** датумски податак облика yyymmdd; yyy - година, mm - месец, dd - дан

- ✧ **TIME -** временски податак облика hhmmss; hh - сат mm - минут ss - секунда

Наредба креирања табеле

Приликом креирања табеле, односно дефиниције њене структуре и особина, неопходно је навести следеће:

- ✧ Име табеле, које мора бити уникатно у бази података
- ✧ Име сваке колоне, које мора бити уникатно унутар табеле
- ✧ Тип сваке колоне
- ✧ Једно или више ограничења за колоне које их имају
- ✧ Једно или више ограничења за целу табелу, ако постоје

У усвојеној нотацији за општу дефиницију синтаксе наредбе креирања табеле:

```
NaredbaKreiranjaTabele ::=
        CREATE TABLE Tabela ( DefinicijaKolone,..
                              [OgranicenjeTabele,..]) ;
NaredbaKreiranjaTabele ::=
        Kolona Tip|Domen OgranicenjeKolone …
```

Табела и колона формирају се по правилу које важи за варијабле у већини програмских језика - први знак је слово, остали знаци су слова/цифре/знак. Тип је један од стандардних SQL типова, а домен је симбол наведен у склопу одговарајуће CREATE DOMAIN наредбе којом је над неким уграђеним типом дефинисан кориснички тип.

Ограничење колоне

Уз сваку колону се могу навести ни једно, једно или више ограничења за ту колону. Основне форме за OgranicenjeKolone и њихово значење су:

- ✧ NOT NULL - у колони није дозвољена NULL вредност
- ✧ UNIQUE - у колони није дозвољено понављање вредности
- ✧ PRIMARY KEY - колона је примарни кључ и у њој није дозвољена NULL вредност нити понављање вредности

- ✧ CHECK _(Predikat)_ - свака вредност у колони мора да задовољава услов задат логичким изразом који је означен предикатом
- ✧ DEFAULT = Const - ако се приликом уношења једног реда података у табелу за колону не зада вредност, преузима се подразумевана вредност константе

Следећа конструкција служи за назнаку да је једна колона страни кључ и за динамичку спецификацију референцијског интегритета:

```
REFERENCES Tabela [(Kolona)]
[  ON UPDATE  NO ACTION|RESTRICT|CASCADE|SET NULL|SET DEFAULT   ]
[  ON DELETE  NO ACTION|RESTRICT|CASCADE|SET NULL|SET DEFAULT   ]
```

Колона реферише циљну табелу Tabela, и то колону Kolona ако је наведена, а примарни кључ ако није. Ако нека од клаузула операција ON UPDATE или ON DELETE није задата, за ту операцију се подразумева NO ACTION.
Операција NO ACTION и новија операција RESTRICT имају исти крајњи ефекат - одбијање извршења промене у циљној табели ако се нарушава референцијални интегритет, али се разликују у имплементацији:

- ✧ NO ACTION је дијагностички, у смислу да не подразумева провере пре промене у циљној табели, него само проверу током саме операције промене. Уколико се детектује нарушавање референцијског интегритета, систем управљања базом података прекида операцију и поништава ефекте свих до тада извршених промена
- ✧ RESTRICT је предиктиван у смислу да подразумева провере пре било каквбе промене у циљној табели; то спроводи систем управљања базом података, и тек када ањ увњеи да нигде неће бити нарушен референцијални интегритет, спроводи операцију промене.

Треба нагласити да су сва ограничења наведена у CREATE TABLE дефиницији неке табеле активна у сваком тренутку постојања те табеле. Сваки покушај да се над некком табелом уради нешто што је у супротности са било којим од ограничења биће сигнализиран и одбијен од система за управљање базом података. То је огромна олакшица за пројектанте и програмере базе података, који би у супротном морали да све провере уграђују у своје апликативне програме. За SQL важи констатација да је као језик мешавина процедуралног и декларативног, али за CREATE TABLE наредбу та констатација сигурно не важи. Декларативна моћ те наредбе је огромна, нарочито у случају динамичке спецификације референцијалног интегритета.
Као илустрацију употребе CREATE TABLE наредбе наводи се комплетна дефиниција базе података KOMPANIJA (дата испод). Ради прегледности, називе табела и колона се наводе малим словима и дати су коментари неких ограничења

Шема релационе базе података KOMPANIJA је једноставан, али реалан пример. Користи се за праћење информација о пословима, радницима, пројектима и задацима у некој компанији. Такође, прати и стање на рачунима пројекта и информације о запосленима.

POSAO (POSAO, OPIS, CENA PO SATU, POSLEDNJA_IZMENA) - табела POSAO садржи информације о пословима које радници могу обављати

RADNIK (ID, IME, PREZIME, INICIJAL, DATUM_ZAPOSLENJA, RADNI STAZ, POSAO) - табела RADNIK садржи информације о раднику у компанији

PROJEKAT (PROJEKAT, NAZIV, VREDNOST, STANJE_NA_RACUNU, RADNIK) - табела PROJEKAT садржи информације о пројектима на којима фирма ради

ZADATAK (ZADATAK, DATUM, BROJ_RADNIH_SATI, RADNIK, PROJEKAT) - табела ZADATAK описује које задатке радници обављају у склопу пројекта

SQL

```
CREATE DATABASE KOMPANIJA1;
USE KOMPANIJA1;

-- Табела за ПОСАО
CREATE TABLE POSAO (
   POSAO INT PRIMARY KEY,
   OPIS VARCHAR(255),
   CENA_PO_SATU DECIMAL(10, 2),
   POSLEDNJA_IZMENA DATE
);

-- Табела за РАДНИК
CREATE TABLE RADNIK (
   ID INT PRIMARY KEY,
   IME VARCHAR(50),
   PREZIME VARCHAR(50),
   INICIJAL CHAR(1),
   DATUM_ZAPOSLENJA DATE,
   RADNI_STAZ INT,
   POSAO INT,
   FOREIGN KEY (POSAO) REFERENCES POSAO(POSAO)
);

-- Табела за ПРОЈЕКАТ
CREATE TABLE PROJEKAT (
   PROJEKAT INT PRIMARY KEY,
   NAZIV VARCHAR(255),
   VREDNOST DECIMAL(15, 2),
   STANJE_NA_RACUNU DECIMAL(15, 2),
   RADNIK INT,
   FOREIGN KEY (RADNIK) REFERENCES RADNIK(ID)
);

-- Табела за ЗАДАТАК
CREATE TABLE ZADATAK (
   ZADATAK INT PRIMARY KEY,
   DATUM DATE,
   BROJ_RADNIH_SATI INT,
   RADNIK INT,
   PROJEKAT INT,
   FOREIGN KEY (RADNIK) REFERENCES RADNIK(ID),
   FOREIGN KEY (PROJEKAT) REFERENCES PROJEKAT(PROJEKAT)
);

-- Убацивање података у табелу POSAO
INSERT INTO POSAO (POSAO, OPIS, CENA_PO_SATU, POSLEDNJA_IZMENA)
VALUES
```

```
(1, 'Programer', 1500.00, '2024-01-15'),
(2, 'Dizajner', 1200.00, '2024-02-10'),
(3, 'Menadžer projekta', 2000.00, '2024-03-05'),
(4, 'Analitičar', 1400.00, '2024-04-01'),
(5, 'Sistem inženjer', 1700.00, '2024-05-15'),
(6, 'Konsultant', 1800.00, '2024-06-20');

-- Убацивање података у табелу RADNIK
INSERT INTO RADNIK (ID, IME, PREZIME, INICIJAL, DATUM_ZAPOSLENJA, RADNI_STAZ, POSAO) VALUES
(1, 'Marko', 'Marković', 'M', '2020-06-01', 4, 1),
(2, 'Ana', 'Anić', 'A', '2021-04-15', 3, 2),
(3, 'Jovan', 'Jovanović', 'J', '2019-09-20', 5, 3),
(4, 'Ivana', 'Ivić', 'I', '2022-01-10', 2, 4),
(5, 'Petar', 'Petrović', 'P', '2018-11-05', 6, 5),
(6, 'Sanja', 'Savić', 'S', '2020-07-25', 4, 6);

-- Убацивање података у табелу PROJEKAT
INSERT INTO PROJEKAT (PROJEKAT, NAZIV, VREDNOST, STANJE_NA_RACUNU, RADNIK) VALUES
(1, 'Razvoj novog softvera', 100000.00, 50000.00, 1),
(2, 'Redizajn sajta', 50000.00, 30000.00, 2),
(3, 'Implementacija ERP sistema', 150000.00, 75000.00, 3),
(4, 'Migracija podataka', 70000.00, 20000.00, 4),
(5, 'Razvoj mobilne aplikacije', 80000.00, 35000.00, 5),
(6, 'Modernizacija sistema', 120000.00, 60000.00, 6);

-- Убацивање података у табелу ZADATAK
INSERT INTO ZADATAK (ZADATAK, DATUM, BROJ_RADNIH_SATI, RADNIK, PROJEKAT) VALUES
(1, '2024-01-20', 8, 1, 1),
(2, '2024-02-15', 5, 2, 2),
(3, '2024-03-10', 10, 3, 3),
(4, '2024-01-22', 4, 1, 3),
(5, '2024-02-20', 7, 4, 4),
(6, '2024-03-15', 6, 5, 5),
(7, '2024-04-10', 9, 6, 6),
(8, '2024-05-22', 5, 4, 5),
(9, '2024-06-30', 8, 6, 6);
```

Опис рада компаније

Компанија KOMPANIJA је организација која се бави пружањем различитих услуга у области информационих технологија и управљања пројектима. Компанија ангажује стручњаке у различитим областима као што су програмирање, дизајн, управљање пројектима, и системска интеграција. Њихови пројекти укључују развој софтвера, редизајн веб страница, имплементацију система и консалтинг. Ови основни подаци и опис рада могу послужити као полазна тачка за представљање компаније и њених активности.

Избор динамичких спецификација интегритета за случај уклањања представља деликатну одлуку. Ако претерано користимо клаузулу NO ACTION, надметнућемо врло крут режим рада са базом података који може довести и до тога да не можемо да у бази података региструјемо промене које се дешавају у реалном систему кога она представља, а у екстремним случајевима да чак не можемо да уклонимо погрешно унете податке из табела. Са друге стране, олако коришћење клаузуле CASCADE код операције DELETE може довести до непланираног ефекта брисања података који не треба да се бришу.

Наредба измене дефиниције табеле

Наредба измене дефиниције табеле је нешто сложенија , пошто треба да обезбеди следеће могућности измене табеле:

- ✧ Додавање нове дефиниције колоне
- ✧ Измена постојеће дефиниције колоне
- ✧ Уклањање постојеће дефинције колоне
- ✧ Додавање новог ограничења табеле
- ✧ Уклањање постојећег ограничења табеле

Треба нагласити да све то мора бити спроводиво над табелом која има неки садржај. У том смислу ALTER TABLE и пар DROP TABLE - CREATE TABLE нису еквивалентни пошто у другом случају се губи садржај табеле.

Синтаксна дефинција наредбе измене табеле је сложена и изложена је поступно:

```
NaredbaIzmeneTabele ::=
   ALTER TABLE  Tabela  SpecIzmeneTabele ;

SpecIzmeneTabele ::=
   SpecIzmeneKolona | SpecIzmeneOgranicenjaTabele

SpecIzmeneKolona ::=
   SpecIzmeneJedneKolone ,..

SpecIzmeneJedneKolone ::=
   DodavanjeKolone | IzmenaKolone | UklanjanjeKolone

DodavanjeKolone ::=
   ADD [ COLUMN ] DefinicijaKolone

IzmenaKolone ::=
   DodavanjePodrazumevanja | UklanjanjePodrazumevanja

DodavanjePodrazumevanja ::=
   ALTER [ COLUMN ] Kolona SET DEFAULT = Const

UklanjanjePodrazumevanja ::=
   ALTER [ COLUMN ] Kolona DROP DEFAULT

UklanjanjeKolone ::=
   DROP [ COLUMN ] Kolona  RESTRICT|CASCADE

SpecIzmeneOgranicenjaTabele :: =
   SpecIzmeneJednogOgranicenjaTabele ,..

SpecJednogOgranicenjaTabele ::=
   SpecDodavanjaOgranicenja | SpecUklanjanjaOgranicenja

SpecDodavanjaOgranicenja ::=
   ADD CONSTRAINT OgranicenjeTabele

SpecUklanjanjaOgranicenja ::=
   DROP CONSTRAINT OgranicenjeTabele RESTRICT|CASCADE
```

Напомене и објашњења:

- ✧ IzmenaKolone је ограничена само на могућност увођења нове или уклањања подразумеване вредности; у тим околностима, постојећа ограничења колоне се не

могу уклањати, а нова се могу додавати само преко додавања новог ограничења табеле са назначеном једном колоном

- ✧ UklanjanjeKolone не успева ако је наведена колона једина у табели, као и ако је наведена клаузула RESTRICT, а у бази података постоји бар један поглед који реферише колону која се уклања
- ✧ SpecUklanjanjaOgranicenja не успева ако је наведена клаузула RESTRICT, ограничење дефинише кандидат-кључ (преко UNIQUE клаузуле) и у бази података постоји бар један страни кључ који реферише тај кандидат-кључ

Наредба уклањања дефиниције табеле
Синтакса:

NaredbaUklanjanjaTabele : := DROP TABLE Tabela;

Код неких имплементација, табела која се уклања мора бити празна, иначе систем за управљање базом података неће извршити ту наредбу.

Наредбе креирања и уклањања индекса

Индекс је помоћна датотека која треба да убрза приступ подацима у некој основној датотеци. Поред тога, индекс има још једну намену: записи у основној датотеци налазе се у неком физичком редоследу и када приступимо непосредно тој датотеци записе очитавамо рим редоследном, али ако подацима приступамо посредством индекса, очитаваћемо их редоследом који одговара растућој или опадајућој вредности индексног израза.
Синтакса:

CREATE [UNIQUE] INDEX Indeks ON Tabela (Kolona, …) ;

UNIQUE - када се зада ова опција, индекс мора бити уникатан, односно у табели на коју се индекс односи не сме да се више пута понови нека вредност Kolona,...

Indeks - уникатни назив индекса у бази података, симбол се формира по правилу за називе варијабли

Kolona, … - једна или више колона по којима се формира индекс

Треба нагласити да већина имплементације дозвољава и индекс по изразима, односно, функцијама над колонама у саставу индекса, као и растуће и опадајуће индексе.

Над истом табелом по потреби може бити дефинисано више индекса. То се користи када су у разним ситуацијама потребни различити приступи подацима и различити редоследи података. Индекс може бити креиран одмах, док је табела на коју се односи празна или накнадно. Од тог тренутка, садржај индекса и табеле је синхронизован: свако додавање или уклањање података, као и измена вредности неке од колона која је у саставу индексног израза, одражава се на садржај индекса. Индекс може било када и без обзира на садржај своје табеле уклонити наредбом чија је синтакса дефиниција:

DROP INDEX Index;

Наредбе креирања и уклањања погледа

Поглед (на енглеском view) представља изведену табелу (претходне су биле основне), има редове и колоне и настаје као резултат упита над основним табелама и другим

погледима. Редови и колоне погледа нису нигде трајно записани. Уместо тога, сваки пут када се приступа погледу, извршава се упис којом је он дефинисан.
Синтакса:

```
CREATE VIEW Pogled [ (Kolona, …) ] AS Upit;
```

Pogled - уникатни назив погледа у бази података, симбол се нормира по правилу за називе варијабли

Kolona, … - ако се наведу колоне, поглед се понаша као табела са бројем, редоследном и именима како је наведено, а у супротном се преузимају имена колона из основних табела и погледа које су наведене у наредби упита. У оба случаја, поглед наслеђује типове колона из основних табела и погледа из упита

Upit - наредбу упита SELECT коју тек треба да обрадимо, а чији резултат извршавања даје "табелу" која представља поглед

Под одређеним околнстима, поглед може бити "ажурабилан", односно може се користити за измену садржаја основне табеле свог дефиниционог упита.
Поглед се уклања једноставном наредбом чија је синтаксна дефиниција:

```
DROP VIEW Pogled ;
```

Уклањање нема никаквог ефекта на основне табеле из упита.

SQL наредба упита SELECT

Ова наредба за упите представља најзначајнију и најчешће коришћену SQL наредбу за манипулисање подацима.

Редни упит над једном табелом
Генерално, код сваког упита се задаје:
- ✧ Који се подаци траже као резултат
- ✧ Из којих табела се то тражи
- ✧ Који услов треба да задовоље подаци да би били укључени у резултат
- ✧ По ком редоследу се жели приказ резултата

Под редним упитом над једном табелом се подразумева наредба упита SELECT над једном табелом која као резултат даје ни један ред, један ред или низ редова података, од којих сваки одговара подацима из једног реда табеле који задовољава евентуално задати услов.
Резултат упита не мора бити релација у смислу уникатности редова који улазе у резултат. То се испољава када за резултат упита бирамо само неке од колона, када може доћи до појаве истоветних редова у резултату. Стога, приликом формулације упита треба да постоји могућност спецификације да ли желимо елиминицију вишеструког појављивања истих редова у резултату или не.
Синтакса:

```
RedniUpitJednaTabela :: =
        SELECT      R-Lista
        FROM        Tabela
 [      WHERE       R-Predikat      ]
 [      ORDER BY    ( R-Izraz [ASC |DESC ] ) , .. ]  ;
```

R-Lista :: =
* | { [ALL|DISTINCT] R-Izray [[AS] Nadimak] } , …

* - специјални случај када желимо да укључимо све колоне табеле, и то оним редоследом којим су наведене у наредби креирања табеле;

ALL - тражимо да се у резултату прикажу сви редови укључујући и оне који су истоветни, подразумева се ако се ништа не наведе

DISTINCT - тражимо да се из резултата елиминишу сувишна појављивања (осим једног) истоветних редова

Izraz - израз израчунљив над сваким појединим редом табеле који поред назива колона може да садржи и операторе и константе; најчешће је у форми навођења једне колоне

Nadimak - израз иза кога се налази Nadimak понаша се као колона тог назива, како у својству назива колоне у интерактивном приказу резултата тако и у ситуацијама када се резултат даље понаша као табела током извршавања сложене SQL наредбе

Спецификација “одакле” налази се непосредно иза клаузуле FROM:

Tabela - назив основне табеле или погледа над којим се врши упит

Услов укључивања редова табеле у формирање резултата наводи се иза клаузуле WHERE:

R-Predikat - логички израз који је израчунљив над сваким појединим редом табеле; у формирање резултата упита улазе само они редови за које тај израз даје истинит резултат. У најједноставнијим случајевима, R-Predikat је у форми релационог израза у коме се са једне стране релационог оператора (>, <, = итд.) јавља име колоне, а са друге стране име колоне или константа

Жељени редослед приказа резултата наводимо иза ORDER BY клаузула, где наводимо једну или више категорија R-Izraz одвојених запетама по којима желимо уређеност. Најчешће су у питању колоне табеле. Подразумева се растући редослед ASC, а ако уз неки R-Izraz желимо супротно, наводимо клаузулу DESC уз њу.

Најједноставнији SQL-упит у форми

```
SELECT * FROM Tabela ;
```

Ова наредба за приказивање све редове табеле чије је име наведено иза FROM клаузуле. У сваком реду приказују се вредности свих колона, и то оним редоследом којим се налазе у записима у датотеци, односно редоследом навођења колона у наредби CREATE TABLE . Ако би извршили овакву наредбу над било ком табелом из наше базе података BIBLIOTEKA, добили би као приказ исту такву табелу.

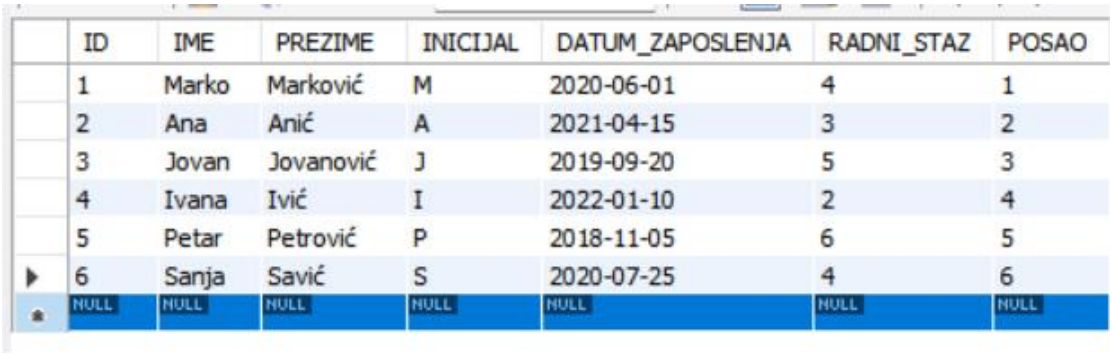
Код упита се најчешће тражи приказ само пдређених колона или приказ свих колона по редоследу који је другачији од стварног. То се постиже наредбом упита код које иза SELECT клаузуле наводимо жељене колоне. Оваква наредба одговара операцији пројекције у релационој алгебри, али постоји једна битна разлика: ако то не нагласимо, у табхели која настаје као приказ неће бити елиминисана вишеструка појављивања истих вредности.

Примери

Упити наведени лево дају за дати садржај базе података BIBLIOTEKA резултате који су наведени десно од њих.

Упит за приказ целе табеле

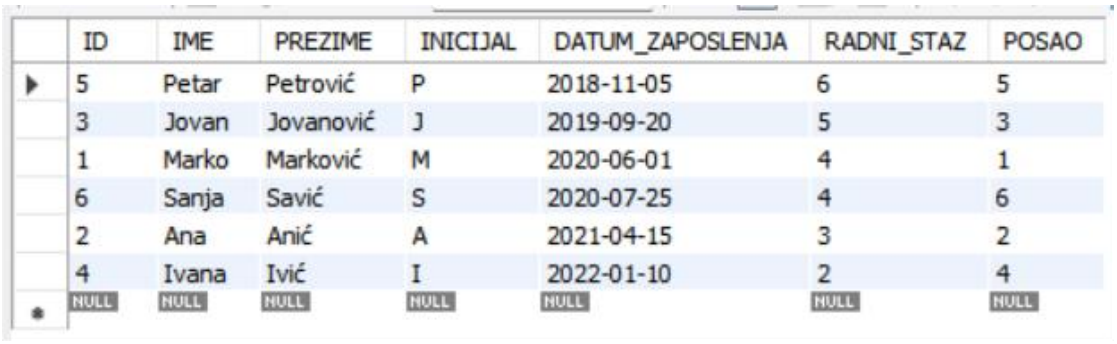
```
SELECT      *
    FROM  RADNIK  ;
```



ID	IME	PREZIME	INICIJAL	DATUM_ZAPOSLENJA	RADNI_STAZ	POSAO
1	Marko	Marković	M	2020-06-01	4	1
2	Ana	Anić	A	2021-04-15	3	2
3	Jovan	Jovanović	J	2019-09-20	5	3
4	Ivana	Ivić	I	2022-01-10	2	4
5	Petar	Petrović	P	2018-11-05	6	5
6	Sanja	Savić	S	2020-07-25	4	6
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Упит за приказ целе табеле у жељеном редоследу

```
SELECT          *
    FROM        RADNIK
    ORDER BY  DATUM_ZAPOSLENJA ASC  ;
```



ID	IME	PREZIME	INICIJAL	DATUM_ZAPOSLENJA	RADNI_STAZ	POSAO
5	Petar	Petrović	P	2018-11-05	6	5
3	Jovan	Jovanović	J	2019-09-20	5	3
1	Marko	Marković	M	2020-06-01	4	1
6	Sanja	Savić	S	2020-07-25	4	6
2	Ana	Anić	A	2021-04-15	3	2
4	Ivana	Ivić	I	2022-01-10	2	4
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Упит за приказ само једне колоне из табеле и без елиминације дупликата, при чему ће у приказу колона преузети свој назив као наслов

```
SELECT      IME   AS   'Име'
    FROM  RADNIK  ;
```

	Име
▶	Marko
	Ana
	Jovan
	Ivana
	Petar
	Sanja

Редни упит над једном табелом са сводним резултатом

Под редним упитом над једном табелом са сводним резултатом подразумева се наредба упита SELECT над једном табелом која као резултат даје један ред података који су изведени из свих редова табеле који задовољавају евентуално задати услов.
Спецификација резултата таквог упита састоји се из једног или више израза који су израчунљиви над више редова табеле, при чему више таквих израза одвајамо запетама.
С обзиром на околност да овакав упит даје само један ред резултата, задавање жељеног редоследа редова у резултату нема смисла.
Дефиниција синтаксе наредбе SELECT у варијанти редног упита над једном табелом са сводним резултатом гласи:

```
RedniUpitJednaTabelaSvodniRezultat ::=
   SELECT G-Lista
    FROM  Tabela
  [ WHERE R-Predikat ] ;

G-Lista ::= { G-Izraz [ AS ] Nadimak ] } ,..
```

Конструкцију G-Izraz најчешће чини једна од посебних SQL функција које се називају сводним или агрегатним функцијама, али у општем случају то могу бити изрази састављени из више таквих функција, оператора и константи.

Сводне функције:

```
SUM ( Kolona )

AVG ( Kolona )

MIN ( Kolona )

MAX ( Kolona )

COUNT ( * )
COUNT ( [ ALL ] Kolona ,.. )

COUNT ( DISTINCT Kolona ,.. )
```

Прва функција налази суму свих ненул вредности задате колоне.

Друга функција налази просечну вредност свих ненул вредности задате колоне.
Трећа функција налази минималну вредност свих ненул вредности задате колоне.
Четврта функција налази максималну вредност свих ненул вредности задате колоне.
Пета функција даје укупан број редова.
Шеста функција даје укупан број ненул вредности задате комбинације колона.
Седма функција даје укупан број различитих ненул вредности задате комбинације колона.

Треба нагласити да у срачунавање вредности агрегатних функција улазе само они редови који задовољавају услов у WHERE клаузули.
Примери

Упит за приказ укупног броја радника у табели

```
SELECT COUNT(*) AS BROJ_RADNIKA FROM RADNIK;
```

Резултат је 6.

Упит за укупно стање на рачуну свих пројеката у табели PROJEKAT

```
SELECT SUM(STANJE_NA_RACUNU) AS UKUPNO_STANJE_NA_RACUNU
FROM PROJEKAT;
```

Резултат је укупно стање на рачуну 270000.00

Приказ најстаријег радника у табели RADNIK - најстарији датум запослења

```
SELECT IME, PREZIME, DATUM_ZAPOSLENJA
FROM RADNIK
WHERE DATUM_ZAPOSLENJA = (SELECT MIN(DATUM_ZAPOSLENJA) FROM RADNIK);
```

Реултат је

Petar Petrović 2018-11-05

Сводни упит над једном табелом

```
SvodniUpitJednaTabela ::=
   SELECT    S-ListaKolona [ ,G-Lista ]
    FROM     Tabela
  [ WHERE    R-Predikat ]
    GROUP BY G-ListaKolona
  [ HAVING   G-Predikat ]
  [ ORDER BY { Element [ ASC|DESC ] } ,.. ] ;

S-ListaKolona ::= { Kolona [ [ AS ] Nadimak ] } ,..

G-Lista ::= { G-Izraz [ [ AS ] Nadimak ] } ,..

G-ListaKolona ::= Kolona ,..
```

S-ListaKolona колоне груписања које желимо да укључујемо у резултат, подскуп колона наведених у оквуру G-ListaKolona

G-Lista листа сводних израза G-Izraz које желимо да укључимо у резултат

R-Predikat услов који сваки ред у табели мора да задовољава да би био узет у поступак груписања

G-ListaKolona једна или више колона табеле по којима се врши груписање

G-Predikat услов који сваки ред формиран свођењем мора да задовољи да би био укључен у резултат
Element може бити само колона садржана у S-ListaKolona или израз садржан у H-Lista

Примери

Овај упит приказује укупну цену рада по сату за сваки тип посла. GROUP BY служи за групацију резултата по опису посла

```
SELECT POSAO.OPIS AS POSAO_OPIS, SUM(POSAO.CENA_PO_SATU) AS UKUPNA_CENA_PO_SATU
FROM POSAO
GROUP BY POSAO.OPIS;
```

POSAO_OPIS	UKUPNA_CENA_PO_SATU
Programer	1500.00
Dizajner	1200.00
Menadžer projekta	2000.00
Analitičar	1400.00
Sistem inženjer	1700.00
Konsultant	1800.00

Овај упит враћа суму илити укупну вредност пројеката са условом да се сабирају вредности пројеката већи од 50.000.

```
SELECT SUM(PROJEKAT.VREDNOST) AS UKUPNA_VREDNOST
FROM PROJEKAT
WHERE PROJEKAT.VREDNOST > 50000;
```

UKUPNA_VREDNOST
520000.00

Упити над више табела

Упити над више табела спајају табеле по неком услову и реализују се тако што се у FROM клаузули SELECT наредбе наведу уместо једног више назива табела одвојених запетама.
У случају да се услов не наведе, од редова наведених табела формира се Декартов производ.

Редни упит над више табела

У случају редног упита над више табела - што укључује и посебан случај једне табеле, синтакса наредбе SELECT је:

```
RedniUpitViseTabelaSvodniRezultat ::=
   SELECT     G-Lista
    FROM    { Tabela [ [ AS ] Nadimak ] } ,..
  [ WHERE     R-Predikat ]
```

- У R-Lista и R-Predikat могу се јавити изрази који су израчунљиви над сваким појединим редом кога чине све колоне свих наведених табела
- За колоне које постоје само у једној табели довољно је да наводимо само називе у R-Lista, R-Predikat и у ORDER BY клаузули
- За колоне које постоје у више табела морамо нагласити из које табеле је колона, и то у форми Tabela.Kolona или Nadimak.Kolona
- Nadimak је по правилу табела и треба да олакша навођење колона, а обавезно је код спајања табеле саме са собом
- У R-Predikat се задаје услов спајања, најчешће у форми услова једнакости вредности одређених колона у табелама
- Упит над више табела без услова спајања даје као резултат Декартов производ тих табела

Пример

Овај упит враћа име радника који имају посао са ИД-ем 1. Наредба DISTINCT уклања дупликате.

```
SELECT DISTINCT IME
FROM RADNIK
WHERE POSAO = 1;
```

Реултат је Марко.

Редни упит са сводним резултатом над више табела

Синтаксна дефиниција наредбе SELECT је:

```
RedniUpitViseTabelaSvodniRezultat ::=
   SELECT     G-Lista
    FROM    { Tabela [ [ AS ] Nadimak ] } ,..
  [ WHERE     R-Predikat ]
```

Сводни упит над више табела

```
SvodniUpitJednaTabela ::=
   SELECT     S-ListaKolona [ ,G-Lista ]
    FROM    { Tabela [ [ AS ] Nadimak ] } ,..
  [ WHERE     R-Predikat ]
    GROUP BY G-ListaKolona
  [ HAVING    G-Predikat ]
  [ ORDER BY { Element [ ASC|DESC ] } ,.. ] ;
```

Пример

Овај упит броји раднике по типу посла

```
SELECT
   r.POSAO,
   COUNT(*) AS BrojRadnika

FROM
   RADNIK r

GROUP BY
   r.POSAO;
```

POSAO	BrojRadnika
1	1
2	1
3	1
4	1
5	1
6	1

Форме предиката у клаузулама WHERE и HAVING

Првенствено треба објаснити појмове простог и сложеног предиката:

- Прост предикат је елементарни логички израз који је израчунљив над сваким појединим редом неке табеле и који се не може растављати на једноставније логичке изразе
- Сложен предикат је логички израз који је формиран из једног или више једноставнијих логичких израза применом логичких оператора AND, OR и NOT

За сложени предикат важи следећа синтаксна дефиниција рекурзивног карактера:

```
SlozenPredikat := [ NOT ] ProstPredikat [ AND|OR SlozenPredikat ]
```

По овој дефиницији могу се из простих формирати сви могући сложени предикати.

Форме простих предиката - SQL језик подржава укупно седам врста простих предиката, у наставку су дати пар од укупно седам:

Izraz <|<=|=|<>|>=|> Izraz
испитује да ли су вредности наведених скаларних израза у задатом односу

Izraz [NOT] BETWEEN Izraz AND Izraz
испитује да ли је (или није) вредност наведеног израза у задатим границама

Kolona IS [NOT] NULL
испитује да ли је (или није) вредност наведене колоне NULL

ZnakovniIzraz [NOT] LIKE ZnakovnaMaska
испитује да ли наведена знаковна вредност типа CHARACTER задовољава или не задати мотив, при чему се за задавање мотива у ZnakovnaMaska користе поред обичних знакова и два специјална знака _ има значење "било који знак", а % има значење "било који број знакова" што укључује и ни један знак

Izraz [NOT] IN (Konstanta, …)
испитује да ли је (или није) вредност наведеног израза једнака некој од наведених константи; Izraz и Konstanta морају бити истог типа

Примери

-- Prikaz svih šifri radnika koji su radili zadatke unutar određenog opsega radnih sati

```
SELECT RADNIK, SUM(BROJ_RADNIH_SATI) AS UKUPNO_SATI
FROM ZADATAK
GROUP BY RADNIK
HAVING SUM(BROJ_RADNIH_SATI) BETWEEN 10 AND 20;
```

Резултат је:

RADNIK	UKUPNO_SATI
1	12
3	10
4	12
6	17

-- Проналазак шифри радника који су радили задатке унутар датог временског периода

```
SELECT DISTINCT RADNIK
FROM ZADATAK
WHERE DATUM BETWEEN '2024-01-01' AND '2024-01-22';
```

Резултат је:

RADNIK
1

-- Проналазак имена радника са шифрама 1, 3 и 5

```
SELECT IME
FROM RADNIK
WHERE ID <> 2 AND ID <> 4 AND ID <> 6;
-- Искључује све остале шифре осим 1, 3 и 5
```

Резултат је:

IME
Marko
Jovan
Petar

Упити над подупитима

Под упитом називамо SELECT наредбу која се налази у склопу WHERE клаузуле и чије извршење претходи вредновању предиката у тим клаузулама. Можемо их класификовати по резултату којег дају и по начину извршавања у односу на "спољну" SELECT наредбу.
Класификација према резултату даје следеће врсте упита:

S-Upit скаларни упит, увек даје као резултат једну вредност
K-Upit колонски упит, као резултат или не даје ни једну вредност или даје скуп вредности, односно редове са једном колоном
R - Upit редни упит општег типа, као резутат или не даје ништа или даје скуп редова са више колона

Класификацију подупита по начину извршења спроводимо по томе да ли његово извршавање зависи од извршавања спољног упита (упита у коме се налази) или не. По томе разликујемо две врсте подупита - некорелациони подупит и корелациони подупит.

Некорелациони подупит јесте подупит чије извршавање ни на који начин не зависи од извршавања спољног упита. Овакав подупит се извршава само на једном, и то на почетку извршавања упита у коме се налази.
Корелациони подупит јесте подупит чије извршавање зависи од извршавања спољног упита. Овакав подупит се извршава за сваки ред табеле коју обрађује спољни упит.

Примери

Враћање имена радника из табеле RADNIK, али само за оне који имају радни стаж већи од одређене вредности која се израчунава у подупиту

```
SELECT IME
FROM RADNIK r
WHERE RADNI_STAZ > (
   SELECT AVG(RADNI_STAZ)
   FROM RADNIK
   WHERE DATUM_ZAPOSLENJA < '2022-01-01'
);
```

Резултат је

IME
Jovan
Petar

-- Пронађи све раднике који су радили на пројектима које води радник са ID 3

```
SELECT r.ID, r.IME
FROM RADNIK r
WHERE r.ID IN (
   SELECT p.RADNIK
   FROM PROJEKAT p
   WHERE p.RADNIK = 3
);
```
--резултат је 3 Јован

-- Пронађи све раднике који не раде на пројектима које води радник са ID 3

```
SELECT r.ID, r.IME
FROM RADNIK r
WHERE NOT EXISTS (
   SELECT 1
   FROM PROJEKAT p
   WHERE p.RADNIK = 3
     AND p.RADNIK = r.ID
);
```
Резултат је

ID	IME
1	Marko
2	Ana
4	Ivana
5	Petar
6	Sanja

Унија, разлика и пресек упита

Под унијом, разликом и пресеком упита подразумевамо примену одговарајућих скуповних оператора на скупове редова које дају поједини упити. Јасно је да при томе упити које на тај начин комбинујемо морају задовољавати услов унијске компатибилности, односно морају давати редове са истим бројем, редоследом и значењима и типовима вредности у ублаженој варијанти услова са истим бројем, редоследом и типовима вредности.

UNION {ALL}	унија два упита при чему се елиминишу истоветни редови ако не нагласи ALL
EXCEPT	разлика два упита; од редова упита испред EXCEPT клаузуле остају само они који се не налазе у резултату упита из те клаузуле
ITERESECT	пресек два упита; од редова оба упита остају само они који се налазе у резултатима оба упита

Унија, разлика у пресек се могу јављати и у подупитима.

Примери

-- Пронађи листу имена радника и назива пројеката

```
SELECT r.IME AS Ime_Radnika
FROM RADNIK r
UNION
SELECT p.NAZIV AS Naziv_Projekta
FROM PROJEKAT p;
```

Резултат је

Ime_Radnika
Marko
Ana
Jovan
Ivana
Petar
Sanja
Razvoj novog softvera
Redizajn sajta
Implementacija ERP sistema
Migracija podataka
Razvoj mobilne aplikacije
Modernizacija sistema

-- Пронађи називе пројеката који су присутни у табели PROJEKAT и као вредности у табели ZADATAK

```
SELECT p.NAZIV
FROM PROJEKAT p
INTERSECT
SELECT p1.NAZIV
FROM PROJEKAT p1
JOIN ZADATAK z ON p1.PROJEKAT = z.PROJEKAT;
```

Резултат је

NAZIV
Modernizacija sistema
Razvoj mobilne aplikacije
Redizajn sajta
Migracija podataka
Implementacija ERP sistema
Razvoj novog softvera

Додатне могућности у наредби SELECT

- **Контрукција CASE за условну селекцију вредности**
- **Конструкција S-Upit као елемент SELECT клаузуле**
- **Контрукција R-Upit као елемент FROM клаузуле**
- **Клаузула JOIN као елемент FROM клаузуле за све врсте спајања табела**

CASE селектор вредности

Пример

```
SELECT
   IME,
   RADNI_STAZ,
   CASE
      WHEN RADNI_STAZ < 3 THEN 'Краткотрајни'
      WHEN RADNI_STAZ BETWEEN 3 AND 5 THEN 'Средњи'
      ELSE 'Дуготрајни'
   END AS KATEGORIJA
FROM RADNIK;
```

Резултат је

IME	RADNI_STAZ	KATEGORIJA
Marko	4	Средњи
Ana	3	Средњи
Jovan	5	Средњи
Ivana	2	Краткотрајни
Petar	6	Дуготрајни
Sanja	4	Средњи

JOIN као елемент FROM клаузуле

✧ Спајање унутар FROM клаузуле - коришћењем JOIN спајања, могуће је формулисати ванъска спајања

Примери

Упит који приказује имена радника и опис послова које обављају

```
SELECT
   r.IME AS Radnik_IME,
   p.OPIS AS Posao_Opis
FROM
   RADNIK r
INNER JOIN
   POSAO p
ON
   r.POSAO = p.POSAO;
```

Резултат је

Radnik_IME	Posao_Opis
Marko	Programer
Ana	Dizajner
Jovan	Menadžer projekta
Ivana	Analitičar
Petar	Sistem inženjer
Sanja	Konsultant

```
SELECT
   r.IME AS Radnik_IME,
   p.OPIS AS Posao_Opis
FROM
   RADNIK r
RIGHT JOIN
   POSAO p
ON
   r.POSAO = p.POSAO;
```

Резултат исти као пример управо изнад.

SQL наредбе ажурирања

- ✧ INSERT наредба за убацивање нових редова у табелу
- ✧ UPDATE наредба за измене редова у табели
- ✧ DELETE наредба за уклањање редова из табеле

Примери

```
-- Убацивање података у табелу POSAO
INSERT INTO POSAO (POSAO, OPIS, CENA_PO_SATU, POSLEDNJA_IZMENA) VALUES
(1, 'Programer', 1500.00, '2024-01-15'),
(2, 'Dizajner', 1200.00, '2024-02-10'),
(3, 'Menadžer projekta', 2000.00, '2024-03-05'),
(4, 'Analitičar', 1400.00, '2024-04-01'),
(5, 'Sistem inženjer', 1700.00, '2024-05-15'),
(6, 'Konsultant', 1800.00, '2024-06-20');
```


```
-- Убацивање података у табелу POSAO
INSERT INTO POSAO (POSAO, OPIS, CENA_PO_SATU, POSLEDNJA_IZMENA) VALUES
(1, 'Programer', 1500.00, '2024-01-15'),
(2, 'Dizajner', 1200.00, '2024-02-10'),
(3, 'Menadžer projekta', 2000.00, '2024-03-05'),
(4, 'Analitičar', 1400.00, '2024-04-01'),
(5, 'Sistem inženjer', 1700.00, '2024-05-15'),
(6, 'Konsultant', 1800.00, '2024-06-20');
```

Погледи

Илити изведене табеле; једном креиран поглед се понаша као нова табела у бази података, али уз једну битну разлику у односу на основне табеле креиране CREATE наредбом: табела која би одговарала погледу трајно не постоји - она се добија извршавањем упита којим је дефинисан поглед.

Особине и предности погледа

- ✧ Поглед не сме да одговара имену табеле и погледа који већ постоје
- ✧ Упит може бити било који
- ✧ Када је дата спецификација колона, између ње и дефиниционог упита мора да постоји сагласност по броју, редоследу и типу
- ✧ Једном креиран поглед може да се увек користи као и свка друга табела у режиму упита, а под одређеним условима и у режиму ажурирања
- ✧ Предност погледа као једне врсте потпрограма
- ✧ Предност погледа као разрешавања проблема појављивања вишка података у сводним упитима
- ✧ Предност погледа као знатно олакшање које прижа у успостави контроле приступа бази података

Примери

```
-- Креирање погледа који приказује све раднике

CREATE VIEW Svi_Radnicii AS
SELECT
   ID AS Radnik_ID,
   IME AS Radnik_IME,
   PREZIME AS Radnik_PREZIME
FROM
   RADNIK;


-- Креирање погледа који приказује раднике и њихове послове
CREATE VIEW Radnici_Posao_Info AS
SELECT
   r.ID AS Radnik_ID,
   r.IME AS Radnik_IME,
   r.PREZIME AS Radnik_PREZIME,
   p.OPIS AS Posao_OPIS
FROM
   RADNIK r
LEFT JOIN
   POSAO p
ON
   r.POSAO = p.POSAO;

-- Извлачење података из погледа
SELECT * FROM Radnici_Posao_Info;
```

Резултат је испод

Radnik_ID	Radnik_IME	Radnik_PREZIME	Posao_OPIS
1	Marko	Marković	Programer
2	Ana	Anić	Dizajner
3	Jovan	Jovanović	Menadžer projekta
4	Ivana	Ivić	Analitičar
5	Petar	Petrović	Sistem inženjer
6	Sanja	Savić	Konsultant

Додатни примери за вежбу и демонстрацију команди SQL-а

Вратити имена свих запослених чија имена почињу словом 'M'.

```
SELECT IME

FROM RADNIK

WHERE IME LIKE 'M%';
```

Симбол '%' делује као џокер, дозвољавајући било ком низу знакова да следи након слова 'M'. '%' представља стринг од 0 или више карактера (док би "_" представља позицију тачно једног карсктера) Овде је присутно коришћење клаузуле LIKE.

Резултат је Марко.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE 'M_';
```

Овај упит ће вратити сва имена из табеле RADNIK где име почиње словом 'M' и има тачно један додатни знак након 'M'. На пример, вратиће имена попут "Ме" или "Ма", али неће вратити имена као што су "Марко" или "Милош", јер имају више од једног знака након 'M'. Стога, резултат овог упита је празна табела.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE '%R_G%';
```

Овај упит враћа имена ге су слова у имену распоређена тако да слово 'G' иде одмах после 'R'. Резултат је празна табела јер немамо таквих имена.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE '__R%';
```

Овај упит ће вратити сва имена где је трећи карактер 'R'. Прва два знака су означена са __, а после њих може следити било шта (%). Резултат је Марко.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE '%N';
```

Овај упит ће вратити сва имена која се завршавају словом 'N'. Резултат је Јован.

```
SELECT IME
FROM RADNIK
WHERE IME LIKE '_____';
```

Овај упит ће вратити сва имена која имају тачно 5 карактера, јер свака доња црта представља један појединачни знак. Резултат упита над овом базом јесте:

```
# IME
Marko
Jovan
Ivana
Petar
Sanja
```

```
SELECT IME
FROM RADNIK
WHERE IME NOT LIKE '_____';
```

Овај упит ће вратити сва имена која **немају** тачно 5 карактера, јер пет доњих црта (_____) представља имена од тачно 5 знакова, а NOT LIKE искључује таква имена. Резултат је Ана.

```
SELECT
   PROJEKAT.PROJEKAT,
   MIN(POSAO.CENA_PO_SATU) AS Minimalna_Plata,
   AVG(POSAO.CENA_PO_SATU) AS Prosecna_Plata,
   MAX(POSAO.CENA_PO_SATU) AS Maksimalna_Plata,
   COUNT(RADNIK.ID) AS Ukupan_Broj_Radnika
FROM
   PROJEKAT
JOIN
   RADNIK ON PROJEKAT.RADNIK = RADNIK.ID
JOIN
   POSAO ON RADNIK.POSAO = POSAO.POSAO
GROUP BY
   PROJEKAT.PROJEKAT;
```

Да би се пронашле минимална, просечна и максимална цена по сату (плата), као и укупан број радника за сваки пројекат, могу се користити SQL агрегатне функције за израчунавање тих вредности груписаних по пројекту.
Резултат:

# PROJEKAT	Minimalna_Plata	Prosecna_Plata	Maksimalna_Plata	Ukupan_Broj_Radnika
1	1500.00	1500.000000	1500.001	

2	1200.00	1200.000000	1200.001
3	2000.00	2000.000000	2000.001
4	1400.00	1400.000000	1400.001
5	1700.00	1700.000000	1700.001
6	1800.00	1800.000000	1800.001

SELECT IME, EXTRACT(YEAR FROM DATUM_ZAPOSLENJA) AS Godina_Zaposlenja
FROM RADNIK;

# IME	Godina_Zaposlenja
Ana	2021
Ivana	2022
Jovan	2019
Marko	2020
Petar	2018
Sanja	2020

SELECT NAZIV, CHARACTER_LENGTH(NAZIV) AS Duzina_Naziva
FROM PROJEKAT;

# NAZIV	Duzina_Naziva
Razvoj novog softvera	21
Redizajn sajta	14
Implementacija ERP sistema	26
Migracija podataka	18
Razvoj mobilne aplikacije	25
Modernizacija sistema	21

Резиме:

Функција EXTRACT у SQL-у користи се за извлачење специфичних делова датума или интервала. Њена синтакса је:

- EXTRACT (polje_datum FROM datum): Ова варијанта функције извлачи одређени део датума (као што је година, месец, дан, сат, итд.) из датума или временске ознаке.
- EXTRACT (polje_datum FROM interval): Ова варијанта функције извлачи одређени део из интервала (као што је година, месец, дан, сат, итд.) из интервалске вредности.
- EXTRACT (vremenska_zona FROM datum): Ова варијанта функције извлачи информације о временској зони из датума или временске ознаке.

Функције CHARACTER_LENGTH и CHAR_LENGTH у SQL-у су синоними и користе се за мерење дужине стринга у карактерима. Њихова синтакса је:

- CHARACTER_LENGTH(string): Ова функција враћа број карактера у датом стрингу.

- CHAR_LENGTH(string): Ова функција такође враћа број карактера у датом стрингу.

```
SELECT IME, OCTET_LENGTH(IME) AS Duzina_U_Oktetima
FROM RADNIK;
```

--пример враћања дужине у октетима

```
# IME   Duzina_U_Oktetima
Marko   5
Ana     3
Jovan   5
Ivana   5
Petar   5
Sanja   5
```

```
SELECT IME, BIT_LENGTH(IME) AS Duzina_U_Bitovima
FROM RADNIK;
```

--пример враћања дужине у битовима

```
# IME   Duzina_U_Bitovima
Marko   40
Ana     24
Jovan   40
Ivana   40
Petar   40
Sanja   40
```

```
SELECT IME, POWER(POSAO.CENA_PO_SATU, 2) AS Plata_na_Kvadrat
FROM RADNIK
JOIN POSAO ON RADNIK.POSAO = POSAO.POSAO;
```

--Израчунавање плата које су подигнуте на квадрат

```
# IME   Plata_na_Kvadrat
Marko   2250000
Ana     1440000
Jovan   4000000
Ivana   1960000
Petar   2890000
Sanja   3240000
```

```
SELECT IME, ROUND(POSAO.CENA_PO_SATU, 2) AS Zaokruzena_Plata
FROM RADNIK
JOIN POSAO ON RADNIK.POSAO = POSAO.POSAO;
```

--Заокруживање плата по сату на две децимале

```
# IME   Zaokruzena_Plata
Marko   1500.00
```

Ana 1200.00
Jovan 2000.00
Ivana 1400.00
Petar 1700.00
Sanja 1800.00

Пример:

SELECT ZADATAK, BROJ_RADNIH_SATI, SIGN(BROJ_RADNIH_SATI) AS ZNAK FROM ZADATAK;

Резултат је

# ZADATAK	BROJ_RADNIH_SATI	ZNAK
1	8	1
2	5	1
3	10	1
4	4	1
5	7	1
6	6	1
7	9	1
8	5	1
9	8	1

Овај упит приказује листу задатака, број радних сати, и знак (1 за позитиван број радних сати, 0 ако су сати 0, и -1 за негативне вредности). У овом случају, пошто су радни сати увек позитивни, вредност SIGN ће увек бити 1.

Пример:
Овај упит враћа тренутно време са тачношћу на стоте делове секунде, укључујући временску зону (нпр. 17:12:30.32 +01:00). Временско одступање представља разлику у времену од УТЦ-а (Универзално време координисано).

SELECT CURRENT_TIME(2) AS TEKUCE_VREME_SA_VREMENSKIM_ODSTUPANJEM;

Резултат:
TEKUCE_VREME_SA_VREMENSKIM_ODSTUPANJEM
23:22:25.58

Пример:
Овај упит враћа тренутно локално време са тачношћу на стоте делове секунде, али **без** временске зоне (нпр. 17:12:30.32). Ово је корисно када није потребно приказати временску зону.

SELECT LOCALTIME(2) AS TEKUCE_VREME_BEZ_VREMENSKOG_ODSTUPANJA;

Резултат:

TEKUCE_VREME_BEZ_VREMENSKOG_ODSTUPANJA

2024-09-15 23:24:25.28

Пример:
Овај упит враћа тренутни датум и време, укључујући временску зону, са тачношћу на два децимална места (стоте делове секунде). Резултат ће бити у формату као што је:
2024-09-15 17:12:30.32 +01:00.

```
SELECT CURRENT_TIMESTAMP(2) AS TEKUCI_DATUM_I_VREME_SA_VREMENSKIM_ODSTUPANJEM;
```

Резултат:

```
# TEKUCI_DATUM_I_VREME_SA_VREMENSKIM_ODSTUPANJEM
2024-09-15 23:27:49.13
```

Пример:
Овај упит враћа тренутни датум и време **без временске зоне**, са тачношћу на два децимална места. Резултат ће бити у формату:
2024-09-15 17:12:30.32, без приказа временске зоне.

```
SELECT LOCALTIMESTAMP(2) AS TEKUCI_DATUM_I_VREME_BEZ_VREMENSKOG_ODSTUPANJA;
```

Резултат:

```
# TEKUCI_DATUM_I_VREME_BEZ_VREMENSKOG_ODSTUPANJA
2024-09-15 23:30:05.02
```

Наведене датумске функције, комбиноване са вредностима интервалног типа, могу формирати изразе са резултујућом вредношћу датумског типа. На пример, израз:

```
CURRENT DATE + INTERVAL '1' DAY
```

вратиће исту вредност коју ће функција CURRENT_DATE имати сутра. Међутим, ако од једног датума одузмемо други датум, добићемо резултујућу вредност интервалног типа, као у следећем случају:

```
(CURRENT_DATE - DATZAP) YEAR TO MONTH
```

Интервалне функције за обраду појединачних редова. Постоји само једна интервална функција, а уведена је у SQL:1999 стандарду. То је функција ABS и потпуно је аналогна функцији ABS која се примењује на нумеричке вредности. Функција ABS има један операнд, који мора бити интервалног типа, и враћа резултујућу вредност која је истог типа и исте прецизности као и операнд функције. На пример, функција:

```
ABS (TIME '12:00:00' - TIME '13:00:00')
```

вратиће следећу резултујућу вредност:

```
INTERVAL +'1:00:00' HOUR TO SECOND
```

Изрази CASE, NULLIF, COALESCE и CAST. Поред претходно наведених основних функција и израза, SQL:1999 стандард подржава и комплексне условне изразе (CASE), изразе којима се конвертују подаци једног типа у други (CAST) и специјалне случајеве условних CASE израза (NULLIF и COALESCE). CASE израз има следећу синтаксу:

CASE

WHEN USL0V1 THEN rezultat1

WHEN USLOV2 THEN rezultat2

WHEN USLOVN THEN rezultatn

ELSE rezultat"

END

Вредност `CASE` израза је резултат, ако је истинит `USLOV1`, резултат, ако је истинит `USLOV2`, ..., резултат, ако је истинит услов. Ако ниједан од услова није истинит, вредност `CASE` израза је резултат. `CASE` израз је сличан `CASE` наредби неког класичног програмског језика. Суштинска разлика је у томе да `CASE` израз није извршна наредба, већ условни израз, који се у SQL наредбама може користити на било ком месту где је дозвољено коришћење вредносног израза.

Једна од важних примена `CASE` израза је за трансформацију недефинисане (`NULL`) вредности у одређену конкретну вредност. `NULL` вредност се не користи при израчунавању израза и функција. Да би се израчунавање ипак омогућило, користи се `CASE` израз који привремено, унутар упита, мења `NULL` вредност са вредношћу за коју се сами одлучимо, најчешће вредношћу која је неутрална у односу на жељену операцију.

Пример

Направити SQL упит који ће из базе података враћати информације о радницима који су запослени на послу са идентификатором 1. Упит треба да комбинује податке из табела RADNIK, POSAO, и ZADATAK, и да врати следеће информације:

- Име и презиме радника
- Опис посла
- Цена по сату
- Број радних сати
- Укупна примања

Укупна примања треба да буду израчуната као:

- Плата (цена по сату * број радних сати)
- Премија која зависи од броја радних сати:
    - Ако је број радних сати мањи од 5, премија је 0.
    - Ако је број радних сати 5 или више, премија је 100.

```
SELECT R.IME, R.PREZIME, P.OPIS AS POSAO, P.CENA_PO_SATU, Z.BROJ_RADNIH_SATI,
(P.CENA_PO_SATU * Z.BROJ_RADNIH_SATI) +
(CASE
   WHEN Z.BROJ_RADNIH_SATI < 5 THEN 0 -- Ако је број радних сати мањи од 5, нема премије
   ELSE 100 -- Додаје 100 као премију ако су сати 5 или више
END) AS PRIMANJA
FROM RADNIK R
JOIN POSAO P ON R.POSAO = P.POSAO
JOIN ZADATAK Z ON R.ID = Z.RADNIK
WHERE R.POSAO = 1
LIMIT 0, 1000;
```

Резултат:

# IME	PREZIME	POSAO	CENA_PO_SATU	BROJ_RADNIH_SATI PRIMANJA
Marko	Marković	Programer	1500.008	12100.00
Marko	Marković	Programer	1500.004	6000.00

NULLIF израз

NULLIF се користи за замену једне вредности са NULL у случају да су две вредности једнаке. Синтакса је:

NULLIF(израз1, израз2)

Ако је израз1 једнак израз2, резултат је NULL.
Ако нису једнаки, резултат је израз1

COALESCE израз

COALESCE се користи за враћање прве не-NULL вредности из списка израза. Синтакса је:

COALESCE(израз1, израз2, ..., изразN)

Враћа прву не-NULL вредност из списка.
Ако су сви изрази NULL, резултат је NULL.

Конверзија CASE израза у COALESCE

COALESCE је специјалан облик CASE израза који омогућава једноставнију синтаксу. Пример CASE израза који је еквивалентан COALESCE:

```
CASE
   WHEN израз1 IS NOT NULL THEN израз1
   WHEN израз2 IS NOT NULL THEN израз2
   ELSE изразN
END
```

Или написан као

```
COALESCE(израз1, израз2, ..., изразN)
```

Примери

Овим упитом ће сви BROJ_RADNIH_SATI са вредношћу 0 бити приказани као NULL.

```
SELECT R.IME, R.PREZIME,
   P.OPIS AS POSAO,
   P.CENA_PO_SATU,
   NULLIF(Z.BROJ_RADNIH_SATI, 0) AS BROJ_RADNIH_SATI
FROM RADNIK R
JOIN POSAO P ON R.POSAO = P.POSAO
JOIN ZADATAK Z ON R.ID = Z.RADNIK
WHERE R.POSAO = 1
LIMIT 0, 1000;
```

Резултат

# IME	PREZIME	POSAO	CENA_PO_SATU	BROJ_RADNIH_SATI
Marko	Marković	Programer	1500.00	8
Marko	Marković	Programer	1500.00	4

Написати SQL упит за базу података KOMPANIJA1 који враћа податке о запосленима на основу следећих услова:

Циљ: Извршити упит који ће вратити информације о запосленима који имају посао са идентификатором 1.

Резултати

1. Приказати име (IME) запосленог.
2. Приказати опис посла (POSAO).
3. Приказати цену по сату (CENA_PO_SATU) за сваког запосленог.
4. Израчунити PRIMANJA као производ цене по сату и броја радних сати. У случају да је резултат израчунавања NULL (што се може догодити ако су CENA_PO_SATU или BROJ_RADNIH_SATI NULL), резултат PRIMANJA треба да буде 0.

Услови:

Користити функцију COALESCE да би се осигурало да ако је резултат израчунавања NULL, буде замењен са 0.

Ограничење:

Ограничити резултате на првих 1000 редова.

```
SELECT R.IME, P.OPIS AS POSAO, P.CENA_PO_SATU,
   COALESCE(P.CENA_PO_SATU * Z.BROJ_RADNIH_SATI, 0) AS PRIMANJA
FROM RADNIK R
JOIN POSAO P ON R.POSAO = P.POSAO
JOIN ZADATAK Z ON R.ID = Z.RADNIK
WHERE R.POSAO = 1
LIMIT 0, 1000;
```

Резултат

# IME	POSAO	CENA_PO_SATU	PRIMANJA
Marko	Programer	1500.00	12000.00
Marko	Programer	1500.00	6000.00

CAST израз има следећу синтаксу:

CAST (израз AS тип_podataka)

Ова функција омогућава трансформацију вредности једног типа података у други.

Пример

```
SELECT
   CAST(R.ID AS CHAR(4)) AS RADNIK_ID,
   R.IME AS IME,
   P.OPIS AS POSAO
FROM RADNIK R
JOIN POSAO P ON R.POSAO = P.POSAO;
```

# RADNIK_ID	IME	POSAO
1	Marko	Programer
2	Ana	Dizajner
3	Jovan	Menadžer projekta
4	Ivana	Analitičar
5	Petar	Sistem inženjer
6	Sanja	Konsultant

Оператори ANY и SOME су синоними у SQL-у и користе се за упоређивање са бар једном вредношћу у подупиту. Такође, оператор ALL се користи за упоређивање са свим вредностима у подупиту.

Примери:

```
--Izvodi sve informacije o zadacima za određenog radnika
SELECT Z.ZADATAK, Z.DATUM, Z.BROJ_RADNIH_SATI, P.NAZIV AS PROJEKAT_NAZIV
FROM ZADATAK Z
JOIN PROJEKAT P ON Z.PROJEKAT = P.PROJEKAT
WHERE Z.RADNIK = 1;
```

Резултат:

# ZADATAK	DATUM	BROJ_RADNIH_SATI	PROJEKAT_NAZIV
1	2024-01-20	8	Razvoj novog softvera
4	2024-01-22	4	Implementacija ERP sistema

--Pokaži projekte za koje svi radnici koji rade na tom projektu imaju cenu po satu veću od 10000

```
SELECT P.*
FROM PROJEKAT P
WHERE 10000 < ALL (
   SELECT PS.CENA_PO_SATU
   FROM RADNIK R
   JOIN POSAO PS ON R.POSAO = PS.POSAO
   WHERE R.ID IN (
      SELECT Z.RADNIK
      FROM ZADATAK Z
      WHERE Z.PROJEKAT = P.PROJEKAT
   )
);
```

Резултат:
Нал вредности

Овај упит ће вратити све раднике из табеле RADNIK који немају ниједан задатак у табели ZADATAK. NOT EXISTS овде проверава да не постоји запис у табели ZADATAK за дати ID радника.

```
SELECT IME, PREZIME
FROM RADNIK R
WHERE NOT EXISTS (
   SELECT 1
   FROM ZADATAK Z
   WHERE Z.RADNIK = R.ID
);
```

Резултат:
Празна табела

Еквиспајање увек даје резултујућу табелу са две идентичне колоне, односно колоне са идентичним садржајем. Када се из резултата еквиспајања избаци једна од две идентичне колоне добија се природно спајање. Експлицитним навођењем операције природног спајања добија се следећа SQL WHERE.
Услов спајања није експлицитно наведен. Спајање се врши по свим колонама са идентичним називима у обе табеле.

```
SELECT *
FROM RADNIK NATURAL JOIN POSAO;
```

Уколико се у упиту изостави услов спајања добија се декартов производ. Ако се примењује на две табеле тада се свака n-торка прве спаја са сваком n-торком друге табеле. Ако се примењује на више од две табеле тада се резултујућа табела састоји од свих комбинација n-торки свих табела.

```
SELECT R.IME, P.NAZIV
FROM RADNIK R
CROSS JOIN PROJEKAT P
JOIN POSAO S ON R.POSAO = S.POSAO
WHERE S.OPIS = 'Analitičar'
LIMIT 0, 1000;
```

Резултат:

# IME	NAZIV
Ivana	Razvoj novog softvera
Ivana	Redizajn sajta
Ivana	Implementacija ERP sistema
Ivana	Migracija podataka
Ivana	Razvoj mobilne aplikacije
Ivana	Modernizacija sistema

Спајање са самим собом - SELF JOIN

```
SELECT R1.IME AS Radnik1, R2.IME AS Radnik2, R1.POSAO
FROM RADNIK R1
JOIN RADNIK R2 ON R1.POSAO = R2.POSAO
WHERE R1.ID <> R2.ID;
```

Демонтрација LEFT OUTER JOIN-a

Рецимо да желимо да пронађемо све раднике и њихове пројекте, али чак и ако неки радник није додељен ниједном пројекту, и даље желимо да се тај радник прикаже у резултатима.

```
SELECT R.IME, R.PREZIME, P.NAZIV AS NAZIV_PROJEKTA
FROM RADNIK R
LEFT OUTER JOIN PROJEKAT P ON R.ID = P.RADNIK;
```

Резултат:

# IME	PREZIME	NAZIV_PROJEKTA
Marko	Marković	Razvoj novog softvera
Ana	Anić	Redizajn sajta
Jovan	Jovanović	Implementacija ERP sistema
Ivana	Ivić	Migracija podataka
Petar	Petrović	Razvoj mobilne aplikacije
Sanja	Savić	Modernizacija sistema

--демонстрација DELETE наредбе

```
CREATE TEMPORARY TABLE TempIDs AS
SELECT ID
FROM RADNIK
WHERE IME = 'Bojan';

DELETE FROM RADNIK
WHERE ID IN (SELECT ID FROM TempIDs);
```

--демонстрација UNIQUE ограничења

```
CREATE TABLE RADNIK (
   ID INT PRIMARY KEY,
```

```
    IME VARCHAR(50),
    PREZIME VARCHAR(50),
    MLB CHAR(13) NOT NULL UNIQUE,  -- Колона MLB мора имати јединствене вредности
    DATUM_ZAPOSLENJA DATE,
    POSAO INT,
    FOREIGN KEY (POSAO) REFERENCES POSAO(POSAO)
);
```

PRIMARY KEY ограничење дефинише примарни кључ табеле. Оно је задовољено ако и само ако не постоје два реда у табели са истим вредностима у колонама на којима је ограничење специфицирано и ако не постоји ни једна недефинисана вредност у тим колонама.

```
-- Табела за ПОСАО
CREATE TABLE POSAO (
    POSAO INT PRIMARY KEY,
    OPIS VARCHAR(255),
    CENA_PO_SATU DECIMAL(10, 2),
    POSLEDNJA_IZMENA DATE
);
```

(UPDATE) редова референциране табеле. Референцијалне акције су NO ACTION, RESTRICT, CASCADE, SET NULL и SET DEFAULT. При спецификацији референцијалног ограничења мора се изабрати једна референцијална акција за операцију DELETE, а друга за операцију UPDATE. NO ACTION референцијална акција поништава ефекте SQL наредбе ако је на крају њеног извршавања нарушено референцијално ограничење. У следећем примеру је специфицирано референцијално ограничење са NO ACTION референцијалном акцијом и за брисање и за измену редова референциране табеле.

```
--демонстрација

CREATE TABLE RADNIK (
    ID INT PRIMARY KEY,
    IME VARCHAR(50),
    PREZIME VARCHAR(50),
    MLB CHAR(13) NOT NULL UNIQUE,
    DATUM_ZAPOSLENJA DATE,
    POSAO INT,
    FOREIGN KEY (POSAO) REFERENCES POSAO(POSAO)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
CREATE TABLE RADNIK ( ID INT PRIMARY KEY, IME VARCHAR(50), PREZIME VARCHAR(50), MLB CHAR(13) NOT NULL UNIQUE, DATUM_ZAPOSLENJA DATE, POSAO INT, FOREIGN KEY (POSAO) REFERENCES POSAO(POSAO) ON DELETE SET NULL ON UPDATE SET NULL );
```

NO ACTION је подразумевана референцијална акција, што значи да се примењује ако ниједна акција није експлицитно наведена.

RESTRICT и NO ACTION су веома сличне референцијалне акције. Разлика је у томе што NO ACTION дозвољава да референцијално ограничење буде нарушено током извршавања SQL наредбе, док RESTRICT не дозвољава такво нарушавање.

CASCADE референцијална акција задовољава референцијално ограничење тако што наставља започету операцију (DELETE или UPDATE) на одговарајућим редовима референцирајуће табеле.

SET NULL референцијална акција задовољава референцијално ограничење тако што поставља вредности референцирајућих колона на NULL, док SET DEFAULT поставља вредности на подразумеване вредности. Наравно, SET NULL акција је могућа само ако референцијске колоне нису дефинисане као NOT NULL.

```
CREATE TABLE RADNIK ( ID INT PRIMARY KEY, IME VARCHAR(50), PREZIME
VARCHAR(50), MLB CHAR(13) NOT NULL UNIQUE, DATUM_ZAPOSLENJA DATE,
POSAO INT DEFAULT 0, -- Подразумевана вредност
FOREIGN KEY (POSAO)
REFERENCES POSAO(POSAO)
 ON DELETE SET DEFAULT
 ON UPDATE SET DEFAULT );
```

Пример са коришћењем CHECK ограничења

```
CREATE TABLE RADNIK (
   ID INT PRIMARY KEY,
   IME VARCHAR(50),
   PREZIME VARCHAR(50),
   MLB CHAR(13) NOT NULL UNIQUE,
   DATUM_ZAPOSLENJA DATE,
   POSAO INT,
   RADNI_STAZ INT,
   FOREIGN KEY (POSAO) REFERENCES POSAO(POSAO),
   CONSTRAINT CK_RADNI_STAZ CHECK (RADNI_STAZ >= 0 AND RADNI_STAZ
<= 50)
);
```