Standardni upitni jezik SQL

Standardni upitni jezik SQL

- CHAMBERLIN, IBM RESEARCH LABORATORY, SAN JOSE CALIFORNIA (1974)

- ISTORIJAT
  - SQL-86
  - SQL-89
  - SQL-92 (SQL2)
  - SQL:1999
  - SQL:2003

Standardni upitni jezik SQL

- Osnovna literatura
  - Marjanovi}, Z., *SQL:1999 standard,* Breza, Beograd, 2002.
  - Melton, J., Simon, A., *SQL:1999, Understanding Relational Language Components*, Academic Press, 2002.
  - American National Standars for Information Technology - Database Languages - SQL, ANSI/ISO/IEC 9075-x-1999, 1999.

- KARAKTERISTIKE
  - SQL je u stalnom razvoju. Na po~etku je bio prili~no jednostavan, blizak korisniku i u velikoj meri deklarativan (neproceduralan)
  - Danas se za SQL mo`e re}i da je kompleksan, proceduralno/deklarativan jezik.

- Zaklju~no sa SQL-92 standardom SQL naredbe su svrstavane u jednu od slede}e tri kategorije:

    - naredbe za definisanje podataka (data definition statements)
    - naredbe za manipulisanje (rukovanje) podacima (data manipulation statements) i
    - naredbe za kontrolne (upravlja~ke) funkcije (data control statements).

- Naredbe za definisanje podataka omogu}uju definisanje objekata baze. Primeri naredbi ove kategorije su:

    - **CREATE TABLE** (kreiranje tebele baze podataka)
    - **CREATE VIEW** (kreiranje virtuelne tabele - "pogleda")
    - **CREATE INDEX** (kreiranje indeksa nad kombinacijom kolona tabele)
    - **ALTER TABLE** (izmena definicije tabele)
    - **DROP TABLE** (izbacivanje tabele iz baze podataka)

- Naredbe za manipulisanje (rukovanje) podacima omogu}uju a`uriranje i prikaz podataka baze:

  - **SELECT** (prikaz sadr`aja relacione baze podataka)
  - **UPDATE** (izmena vrednosti kolona tabele)
  - **DELETE** (izbacivanje redova tabele)
  - **INSERT** (dodavanje redova postoje}oj tabeli)

- Naredbe za kontrolne (upravlja~ke) funkcije omogu}uju oporavak, konkurentnost, sigurnost i integritet relacione baze podataka:

    - **GRANT** (dodela prava kori{}enja sopstvene tabele drugim korisnicima)
    - **REVOKE** (oduzimanje prava kori{}enja sopstvene tabele od drugih korisnika)
    - **COMMIT** (prenos dejstava transakcije na bazu podataka)
    - **ROLLBACK** (poni{tavanje dejstava transakcije)

- SQL:1999 standard razvrstava SQL naredbe u sedam kategorija:
    1. **Naredbe za {emu baze podataka** (SQL-schema statements), koje se koriste za kreiranje, izmenu i izbacivanje {ema i objekata {ema (CREATE, ALTER, DROP)
    2. **Naredbe za podatke** (SQL-data statements), koje se koriste za prikaz i a`uriranje podataka baze (SELECT, INSERT, UPDATE, DELETE)
    3. **Naredbe za transakcije** (SQL-transaction statements), koje se koriste za startovanje, zavr{avanje i postavljanje parametara za transakcije (COMMIT, ROLLBACK)
    4. **Naredbe za kontrolu** (SQL-control statements), koje se koriste za kontrolu izvr{avanja sekvence SQL naredbi (CALL, RETURN)
    5. **Naredbe za konekcije** (SQL-connection statements), koje se koriste za uspostavljanje i prekidanje SQL konekcije (CONNECT, DISCONNECT)
    6. **Naredbe za sesije** (SQL-session statements), koje se koriste za postavljanje default vrednosti i drugih parametara SQL sesije (SET)
    7. **Naredbe za dijagnostiku** (SQL-diagnostic statements), koje koriste dijagnosti~ke podatke i signaliziraju izuzetke u SQL rutinama (GET DIAGNOSTICS)

- Dva osnovna na~ina kori{}enja SQL-a su:
  - direktno (interaktivno) kori{}enje SQL-a i
  - povezivanje SQL-a sa klasi~nim programskim jezicima ("ugradjeni" SQL).

  Kursor i naredbe za rad sa kursorom:

  a) **OPEN** <kursor> - konceptualno prouzrokuje generisanje skupa n-torki kao rezultat izvr{avanja upitnog izraza tj. SELECT upitnog bloka definisanog za <kursor>,

  b) **FETCH** <kursor> - izvr{ava se jedanput za svaku n-torku kada je kursor otvoren naredbom OPEN <kursor> i prouzrokuje pomeranje kursora na slede}u n-torku generisanog skupa i prikaz te n-torke,

  c) **CLOSE** <kursor> - zatvara kursor nakon obrade n-torki generisanog skupa.

  d) **DECLARE** <kursor> - unutar te naredbe navodi se i upitni izraz, odnosno SELECT upitni blok koji, pri otvaranju kursora, generi{e skup n-torki.

Relaciona {ema i model

```
ODELJENJE (ODELJENJE#, NAZIV, GRAD)
RADNIK (SRADNIK, IME, POSAO, SRUKOV, DATZAP, PLATA,
             PREMIJA, ODELJENJE#)
```

relaciona shema

ODELJENJE#	NAZIV	GRAD
10	RAD_ZAJ	BEOGRAD
20	PRIPREMA	NIS
30	PROJEKTOVANJE	BOR
40	ISTRAZIVANJE	SARAJEVO

Relaciona {ema i model

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJE#
3069	Steva	ANALITICAR	3602	1980-12-17	80000		20
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3266	Marko	RUKOVODILAC	3539	1981-04-02	297500		20
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3398	Dragan	RUKOVODILAC	3539	1981-05-01	285000		30
3482	Ivan	RUKOVODILAC	3539	1981-06-09	245000		10
3488	Pavle	SAVETNIK	3266	1981-11-09	300000		20
3539	Jovan	PREDSEDNIK		1981-11-17	500000		10
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30
3576	Adam	ANALITICAR	3488	1981-09-23	110000		20
3600	Janko	ANALITICAR	3398	1981-12-03	95000		30
3602	Filip	SAVETNIK	3266	1981-12-03	300000		20
3634	Dejan	ANALITICAR	3482	1982-01-23	130000		10

Definisanje koncepata strukture

- SQL tipovi podataka
- domeni
- tabele i kolone - osnovna svojstva
- {ema baze podataka
- katalog

SQL tipovi podataka

- SQL:1999 standard podr`ava numeri~ke, tekstualne, binarne, datumske, intervalne i logi~ke tipove podataka
- Numeri~ki tipovi (ta~ni i pribli`ni)
    - INTEGER (ili INT)
    - SMALLINT
    - NUMERIC (p, d)
    - DECIMAL (p, d) (ili DEC)

    - REAL
    - DOUBLE PRECISION
    - FLOAT (p)

SQL tipovi podataka

- Tekstualni tipovi
  - CHARACTER (ili CHAR)
  - CHARACTER VARYING (ili CHAR VARYING ili VARCHAR)
  - CHARACTER LARGE OBJECT (ili CLOB ili CHAR LARGE OBJECT)

SQL tipovi podataka

- Binarni tipovi
  - BIT
  - BIT VARYING
  - BINARY LARGE OBJECT (ili BLOB)

SQL tipovi podataka

- Datumski tipovi

  Definisani su u odnosu na univerzalno koordinisano vreme (UTC)

  - DATE
  - TIME
  - TIMESTAMP
  - TIME WITH TIME ZONE
  - TIMESTAMP WITH TIME ZONE

SQL tipovi podataka

- Intervalni tipovi
  - Godina-mesec interval

    INTERVAL YEAR, INTERVAL YEAR (p), INTERVAL MONTH, INTERVAL MONTH (p), INTERVAL YEAR TO MONTH, INTERVAL YEAR (p) TO MONTH
  - Dan-vreme interval

    INTERVAL DAY TO HOUR, INTERVAL DAY (6) TO MINUTE, INTERVAL SECOND (7), INTERVAL DAY (5) TO SECOND (10), INTERVAL MINUTE (3) TO SECUNDE (4)

SQL tipovi podataka

- Logi~ki tipovi
  - BOOLEAN

    Tri logi~ke vrednosti TRUE, FALSE, UNKNOWN

Domeni

```
CREATE DOMAIN <naziv domena> [AS] <predefinisani tip>
   [DEFAULT <vrednost>]
   [[ CONSTRAINT <naziv ogranicenja>] CHECK (<ogranicenje>)]
   ...
```

Primer:

```
CREATE DOMAIN novac AS DECIMAL (7,2)

CREATE DOMAIN vrsta_racuna AS CHAR (1)
   DEFAULT 'L'
   CONSTRAINT vrsta_racuna_provera
   CHECK (value IN ('L', 'R', 'P'))
```

Domeni

Definicija domena menja se naredbom ALTER DOMAIN:

```
ALTER DOMAIN <naziv domena>
SET DEFAULT <vrednost> |
DROP DEFAULT |
ADD [CONSTRAINT <naziv ogranicenja>] CHECK (<ogranicenje>) |
DROP CONSTRAINT <naziv ogranicenja>
```

Domen se uni{tava slede}om naredbom:

```
DROP DOMEN <naziv domena>
```

Tabele i kolone - osnovna svojstva

- Bazne tabele
  - Perzistentne bazne tabele
  - Globalne privremene tabele
  - Kreirane lokalne privremene tabele
  - Deklarisane lokalne privremene tabele
- Tabele pogleda
- Izvedene tabele

Kreiranje tabele - osnovna sintaksa

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

Kreiranje tabele - osnovna sintaksa

Tabele RADNIK i ODELJENJE kreirane su slede}im naredbama:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER NOT NULL,
   IME VARCHAR (20) NOT NULL,
   POSAO VARCHAR (20),
   SRUKOV INTEGER,
   DATZAP DATE,
   PLATA NUMERIC (10),
   PREMIJA NUMERIC (10),
   ODELJENJE# INTEGER NOT NULL);

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER NOT NULL,
   NAZIV VARCHAR (20) NOT NULL,
   GRAD VARCHAR (20) );
```

Kreiranje tabele - osnovna sintaksa

- Pri definisanju kolone, umesto tipa podataka, mogu}e je navesti domen
- Pored navedenih osnovnih svojstava, definicija kolone mo`e da obuhvati i specifikaciju default vrednosti i ograni~enja na vrednosti kolone.
- Za default vrednost neke kolone mogu}e je specificirati:
  - literal odgovaraju}eg tipa,
  - neku od datumskih funkcija (CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, LOCALTIME ili LOCALTIMESTAMP)
  - identifikator korisnika (CURRENT_USER, SESSION_USER ili SYSTEM_USER) ili NULL
- Kolona DATZAP mogla je biti specificirana na slede}i na~in:

```
DATZAP DATE DEFAULT CURRENT_DATE
```

Kreiranje tabele - osnovna sintaksa

- Globalna privremena tabela definiše se na sledeći način:

```
CREATE GLOBAL TEMPORARY TABLE <naziv tabele>
   (<naziv kolone1> <tip podataka> [NOT NULL],
    <naziv kolone2> <tip podataka> [NOT NULL],
        ...                )
```

- Sadržaj jedinstven u SQL sesiji

Kreiranje tabele - osnovna sintaksa

- Kreirana lokalna privremena tabela defini{e se slede}om naredbom:

```
CREATE LOCAL TEMPORARY TABLE <naziv tabele>
   (<naziv kolone1> <tip podataka> [NOT NULL],
    <naziv kolone2> <tip podataka> [NOT NULL],
     . . .      )
```

- Sadr`aj jedinstven unutar modula ili “embedded” sql programa jedne SQL sesije

Kreiranje tabele - osnovna sintaksa

- Deklarisana lokalna privremena tabela defini{e se na slede}i na~in:

```
DECLARE LOCAL TEMPORARY TABLE MODULE.<naziv
  tabele>
  (<naziv kolone1> <tip podataka> [not null],
   <naziv kolone2> <tip podataka> [not null],
    . . .      )
```

- Sadr`aj jedinstven unutar procedure modula

Izmena definicije tabele

- Definicija postoje}e tabele se mo`e izmeniti:
  - dodavanjem nove kolone
  - izmenom definicije postoje}e kolone
  - izbacivanjem kolone i
  - dodavanjem ili izbacivanjem ograni~enja na vrednosti podataka tabele

Izmena definicije tabele

- Ukoliko `elimo da dodamo novu kolonu postoje}oj tabeli op{ti oblik naredbe je:

```
ALTER TABLE <naziv tabele>
ADD [COLUMN] <definicija kolone>
```

- Strukturu tabele ODELJENJE mo`emo izmeniti dodavanjem nove kolone BROJ_RADNIKA, {to se realizuje slede}om naredbom:

```
ALTER TABLE ODELJENJE
ADD COLUMN BROJ_RADNIKA NUMERIC (4)
```

Izmena definicije tabele

- Izmena definicije postoje}e kolone realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
ALTER [COLUMN] <naziv kolone>
  SET DEFAULT <vrednost> |
  DROP DEFAULT.
```

- Ovim oblikom ALTER TABLE naredbe odredjuje se, menja se ili izbacuje default vrednost kolone, kao u slede}im primerima:

```
ALTER TABLE ODELJENJE
ALTER COLUMN GRAD
  SET DEFAULT 'BEOGRAD'

ALTER TABLE RADNIK
ALTER COLUMN DATZAP
  DROP DEFAULT
```

Izmena definicije tabele

- Izbacivanje kolone iz tabele realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
DROP [COLUMN] <naziv kolone>
```

- Kolonu BROJ_RADNIKA mo`emo izbaciti iz tabele ODELJENJE na slede}i na~in:

```
ALTER TABLE ODELJENJE
DROP COLUMN BROJ_RADNIKA
```

Izmena definicije tabele

- Dodavanje ili izbacivanje ograni~enja na vrednosti podataka tabele realizuje se slede}om naredbom:

```
ALTER TABLE <naziv tabele>
ADD [CONSTRAINT <naziv ogranicenja>]
                <ogranicenje tabele> |
DROP CONSTRAINT <naziv ogranicenja>
```

Izbacivanje tabele

- Ukoliko ho}emo da izbacimo iz baze podataka definiciju tabele, zajedno sa podacima koje sadr`i, koristi se DROP TABLE naredba.

  Op{ti oblik naredbe je:

  ```
  DROP TABLE <naziv tabele>
  ```

- Tabelu PREMIJA mo`emo izbaciti iz baze podataka naredbom:

  ```
  DROP TABLE PREMIJA;
  ```

Indeksi

- Tipi~na sintaksa za kreiranje indeksa je:

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

- Indeks se izbacuje naredbom:

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

[ema baze podataka

- [ema je kolekcija svih objekata koji dele isti prostor imenovanja. Svaki objekat (tabela, pogled, itd.) pripada ta~no jednoj {emi. Pod pripadno{}u se ne podrazumeva fizi~ka pripadnost, ve} hijerarhijska veza u kojoj, na primer, {ema sadr`i nula ili vi{e tabela, a svaka tabela logi~ki pripada ta~no jednoj {emi.
- [ema ima naziv, koji se mo`e koristiti za kvalifikovanje naziva objekata koji pripadaju {emi.
- [ema se kreira CREATE SCHEMA naredbom, iza koje slede naredbe za kreiranje objekata {eme, prvenstveno tabela, domena i pogleda:

```
CREATE SCHEMA <naziv seme> ...

CREATE TABLE <naziv tabele1> ...
CREATE TABLE <naziv tabele2> ...
...
```

[ema baze podataka

- [ema se izbacuje DROP SCHEMA naredbom. Standard zahteva da se obavezno navede CASCADE ili RESTRICT na~in zadovoljavanja integriteta pri izbacivanju {eme. Naredba:

  ```
  DROP SCHEMA <naziv seme> CASCADE
  ```

  izbaci}e {emu i sve objekte koji joj pripadaju.
- Naredba:

  ```
  DROP SCHEMA <naziv seme> RESTRICT
  ```

  izbaci}e {emu samo ako je prazna, odnosno ako se u njoj ne nalazi ni jedan objekat. RESTRICT klauzula spre~ava izbacivanje {eme, ako je u njoj bar jedan objekat.

Katalog

- Katalog je imenovana kolekcija {ema baze podataka u SQL okru`enju. SQL okru`enje sadr`i nula ili vi{e kataloga, a katalog sadr`i jednu ili vi{e {ema. Svaki katalog sadr`i {emu sa nazivom INFORMATION_SCHEMA, koja predstavlja re~nik podataka. Nju ~ini skup pogleda, odnosno sistemskih tabela, koje sadr`e sve bitne informacije o SQL okru`enju. Sadr`aj sistemskih tabela se automatski odr`ava.
- U SQL standardu ne postoje naredbe za kreiranje i uni{tavanje kataloga. Na~in njihovog kreiranja i uni{tavanja je implementaciono-definisan (implementation-defined), odnosno prepu{ten je vlasnicima softverskih proizvoda koji implementiraju SQL okru`enje.

Katalog

- Pun naziv objekata {eme ima tri komponente, razdvojene ta~kama: naziv kataloga, naziv {eme i naziv objekta. Ako je posmatrani objekat tabela, tada se njen pun naziv specificira na slede}i na~in:

```
<naziv kataloga>.<naziv seme>.<naziv tabele>.
```

- Objekti {eme se mogu referencirati sa eksplicitnim ili implicitnim nazivom kataloga i {eme:

```
FROM <naziv tabele>  -- nekvalifikovan naziv
FROM <naziv seme>.<naziv tabele>
                              -- delimi~no kvalifikovan naziv
FROM <naziv kataloga>.<naziv seme>.<naziv tabele>
                              -- potpuno kvalifikovan naziv
```

Operacije - upiti

- Osnova SQL-a je upitni blok oblika:

```
SELECT < lista atributa >
FROM < lista relacija >
WHERE < kvalifikacioni izraz >.
```

- Listom atributa zadaje se operacija PROJEKCIJE. Kvalifikacionim izrazom zadaju se uslovi SELEKCIJE i SPAJANJA, odnosno iskazi sli~ni iskazima u relacionom ra~unu.
- Klauzule SELECT i FROM su obavezne, dok klauzula WHERE nije.

Upiti nad jednom tabelom kojima se prikazuje prost, neizmenjen sadr`aj te tabele

- *Prikazati {ifre, nazive i lokacije svih odeljenja iz relacije o odeljenjima.*

```
SELECT ODELJENJE#, NAZIV, GRAD
FROM ODELJENJE;

ODELJENJE# NAZIV          GRAD
---------- -------------- -------------
        10 RAD_ZAJ        BEOGRAD
        20 PRIPREMA       NIS
        30 PROJEKTOVANJE  BOR
        40 ISTRAZIVANJE   SARAJEVO
```

Upiti nad jednom tabelom kojima se prikazuje prost, neizmenjen sadr`aj te tabele

- *Kada se tra`e svi atributi neke relacije, umesto navo|enja svakog atributa pojedina~no, mogu}e je koristiti znak "***** "sa istim dejstvom.*

```
SELECT *
FROM ODELJENJE;
```

```
ODELJENJE# NAZIV          GRAD
---------- -------------- -------------
 10 RAD_ZAJ        BEOGRAD
 20 PRIPREMA       NIS
 30 PROJEKTOVANJE  BOR
 40 ISTRAZIVANJE   SARAJEVO
```

- Pored prikazivanja svih atributa neke relacije (tj. kolona neke tabele) koriste}i SELECT klauzulu mogu}e je:
    - izdvojiti specifi~ne atribute relacije
    - kontrolisati redosled pojavljivanja atributa
    - spre~iti selekciju duplih n-torki
- *Prikaziti nazive i {ifre svih odeljenja*

```
SELECT NAZIV, ODELJENJE#
FROM ODELJENJE;

NAZIV          ODELJENJE#
-------------- ----------
RAD_ZAJ                10
PRIPREMA               20
PROJEKTOVANJE          30
ISTRAZIVANJE           40
```

- *Prikazati sve poslove radnika*

```
SELECT POSAO
FROM RADNIK;

POSAO
------------
ANALITICAR
TRG_PUTNIK
TRG_PUTNIK
RUKOVODILAC
TRG_PUTNIK
RUKOVODILAC
RUKOVODILAC
SAVETNIK
PREDSEDNIK
TRG_PUTNIK
ANALITICAR
ANALITICAR
SAVETNIK
ANALITICAR
```

- Koriste}i klauzulu DISTINCT obezbedjujemo prikazivanje samo razli~itih poslova.

```
SELECT DISTINCT POSAO
FROM RADNIK;

POSAO
------------
ANALITICAR
PREDSEDNIK
RUKOVODILAC
SAVETNIK
TRG_PUTNIK
```

WHERE klauzula

- Where klauzula nije obavezna, a mo`e se koristiti sa SELECT, UPDATE i DELETE naredbama. Kori{}ena u SELECT bloku ona omogu}uje:
  1. Selekciju specifi~nih n-torki relacije (redova tabele)
  2. Selekciju n-torki koje zadovoljavaju vi{estruke uslove
  3. Selekciju n-torki koje zadovoljavaju bar jedan od vi{e uslova
  4. Selekciju n-torki koje ne zadovoljavaju odredjene uslove
  5. Selekciju n-torki istovremenim kori{}enjem AND i OR logi~kih operatora
  6. Selekciju n-torki unutar izvesnog raspona
  7. Selekciju n-torki koje zadovoljavaju vrednost u listi vrednosti
  8. Selekciju n-torki koje sadr`e odredjenu kombinaciju karaktera

- *Prikazati sve podatke o radnicima koji rade u odeljenju 30*

```
SELECT *
FROM RADNIK
WHERE ODELJENJE# = 30;
```

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJ
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3398	Dragan	RUKOVODILAC	3539	1981-05-01	285000		30
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30
3600	Janko	ANALITICAR	3398	1981-12-03	95000		30

- Pored operatora "**=**" mogu}e je koristiti i ostale operatore poredjenja:

  - RAZLI^ITO ("**<>**") WHERE ODELJENJE# <> 30
  - VE]E ("**>**") WHERE PREMIJA > PLATA,
  - MANJE ("**<**") WHERE PREMIJA < PLATA,
  - VE]E ILI JEDNAKO ("**>=**")
    WHERE DATZAP >= ‘1981-09-08’
  - MANJE ILI JEDNAKO ("**<=**").
    WHERE DATZAP <= ‘1981-09-08’

- *Prika`i ime, posao i platu svakog radnika iz odeljenja 20 koji zaradjuje vi{e od 200000 din. (kori{}enje logi~kog operatora AND)*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 20
  AND PLATA > 200000;
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Marko      RUKOVODILAC      297500
Pavle      SAVETNIK         300000
Filip      SAVETNIK         300000
```

- *Prikazati sve podatke o rukovodiocima odeljenja i predsedniku (kori{}enje logi~kog operatora OR)*

```
SELECT *
FROM RADNIK
WHERE POSAO = 'RUKOVODILAC'
   OR POSAO = 'PREDSEDNIK';
```

```
SRADNIK IME    POSAO       SRUKOV DATZAP     PLATA PREMIJA ODELJENJENJE#
------- ------ ----------- ------ --------- ------ ------- -------------
   3266 Marko  RUKOVODILAC   3539 02-APR-81 297500                    20
   3398 Dragan RUKOVODILAC   3539 01-MAY-81 285000                    30
   3482 Ivan   RUKOVODILAC   3539 09-JUN-81 245000                    10
   3539 Jovan  PREDSEDNIK         17-NOV-81 500000                    10
```

- *Prika`i ime, platu, posao i {ifru odeljenja za rukovodioce koji ne rade u odeljenju 30 (kori{}enje logi~kog operatora NOT)*

```
SELECT IME, PLATA, POSAO, ODELJENJE#
FROM RADNIK
WHERE POSAO = 'RUKOVODILAC'
  AND NOT (ODELJENJE# = 30);

IME              PLATA POSAO        ODELJENJE#
---------- ---------- ------------ ----------
Marko            297500 RUKOVODILAC          20
Ivan             245000 RUKOVODILAC          10
```

- *Prika`i ime, platu, posao i {ifru odeljenja za rukovodioca i analiti~are u odeljenju 10 (kori{}enje zagrada da bi se definisao redosled ispitivanja uslova kod istovremene primene AND i OR logi~kih operatora)*

```
SELECT IME, PLATA, POSAO, ODELJENJE#
FROM RADNIK
WHERE (POSAO = 'RUKOVODILAC' OR
       POSAO = 'ANALITICAR' )
  AND ODELJENJE# = 10;

IME              PLATA POSAO        ODELJENJE#
---------- ---------- ------------ ----------
Ivan             245000 RUKOVODILAC          10
Dejan            130000 ANALITICAR           10
```

- *Prika`i ime, posao i platu radnika koji zaradjuju izmedju 120000 i 150000 (kori{}enje klauzule BETWEEN, koja proverava da li je tra`ena vrednost atributa u definisanom rasponu)*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE PLATA BETWEEN 120000 AND 150000;
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Petar      TRG_PUTNIK       125000
Josip      TRG_PUTNIK       125000
Goran      TRG_PUTNIK       150000
Dejan      ANALITICAR       130000
```

Isti uslov mogu}e je realizovati i na slede}i na~in:

```
WHERE PLATA >=120000 AND PLATA <= 150000
```

- *Prika`i ime, posao i sifru odeljenja radnika koji nisu analiti~ari, savetnici niti trgova~ki putnici (kori{}enje operatora IN koji menja vi{estruku primenu operatora OR)*

```
SELECT IME, POSAO, ODELJENJE#
FROM RADNIK
WHERE POSAO NOT IN ('ANALITICAR', 'SAVETNIK',
                    'TRG_PUTNIK');
```

```
IME        POSAO        ODELJENJE#
---------- ------------ ----------
Marko      RUKOVODILAC          20
Dragan     RUKOVODILAC          30
Ivan       RUKOVODILAC          10
Jovan      PREDSEDNIK           10
```

- *Prikazati ime, posao i {ifru odeljenja radnika ~ija imena po~inju sa M. (Kori{}enje klauzule LIKE)*

```
SELECT IME, POSAO, ODELJENJE#
FROM RADNIK
WHERE IME LIKE 'M%';

IME        POSAO        ODELJENJE#
---------- ------------ ----------
Milan      TRG_PUTNIK           30
Marko      RUKOVODILAC          20
```

- Klauzula LIKE omogu}uje pretra`ivanje na osnovu "uzorka" odnosno dobijanje informacija i kada ne znamo potpun naziv (tj. vrednost) odredjenog atributa tipa character. Ona koristi dva specijalna karaktera ("**%**", "_") sa slede}im zna~enjem:

  "**%**" predstavlja string od 0 ili vi{e karaktera
  "_" predstavlja poziciju ta~no jednog karaktera.

  Ostali karakteri imaju uobi~ajeno zna~enje.

- *Primeri:*

  ... gde se ime zavr{ava sa N.

  ```
  WHERE IME LIKE '%N'
  ```

  ... gde je tre}i karakter imena R.

  ```
  WHERE IME LIKE '__R%'
  ```

  ... gde je ime duga~ko 5 karaktera.

  ```
  WHERE IME LIKE '_____'
  ```

  ... gde ime nije duga~ko 5 karaktera.

  ```
  WHERE IME NOT LIKE '_____'
  ```

  ... gde je u imenu slovo G posle R.

  WHERE IME LIKE '%R%G%'

- Ponekad vrednosti tekstualnih kolona sadr`e i karaktere '%' i '_'. Klauzula ESCAPE omogu}uje nala`enje takvih kolona tako {to privremeno, unutar upita, poni{tava specijalni efekat karaktera '%' ili '_'. Escape karakter sami biramo navodjenjem u ESCAPE klauzuli i on poni{tava specijalni efekat karaktera ispred koga se neposredno nalazi u uzorku za pretra`ivanje.
- *Prikazati sve podatke o radnicima koji u nazivu posla imaju karakter '_'.*

```
SELECT *
FROM RADNIK
WHERE POSAO LIKE '%*_%' ESCAPE '*';
```

SRADNIK	IME	POSAO	SRUKOV	DATZAP	PLATA	PREMIJA	ODELJENJE#
3199	Milan	TRG_PUTNIK	3398	1981-02-20	160000	30000	30
3221	Petar	TRG_PUTNIK	3398	1981-02-22	125000	50000	30
3354	Josip	TRG_PUTNIK	3398	1981-09-28	125000	140000	30
3544	Goran	TRG_PUTNIK	3398	1981-09-08	150000	0	30

Kori{}enje NULL vrednosti

- dva osnovna tipa NULL vrednosti
  - jo{ nepoznata vrednost
  - neprimenjivo svojstvo
- za testiranje null vrednosti koristi se sintaksa:

IS NULL ili IS NOT NULL

- *Prikazati ime, posao i premiju radnika koji ne primaju premiju*

```
SELECT IME, POSAO, PREMIJA
FROM RADNIK
WHERE PREMIJA IS NULL;

IME        POSAO           PREMIJA
---------- ------------ ----------
Steva      ANALITICAR
Marko      RUKOVODILAC
Dragan     RUKOVODILAC
Ivan       RUKOVODILAC
Pavle      SAVETNIK
Jovan      PREDSEDNIK
Adam       ANALITICAR
Janko      ANALITICAR
Filip      SAVETNIK
Dejan      ANALITICAR
```

- *Prikazati ime, posao i premiju radnika koji primaju premiju*

```
SELECT IME, POSAO, PREMIJA
  FROM RADNIK
 WHERE PREMIJA IS NOT NULL;

IME        POSAO           PREMIJA
---------- ------------ ----------
Milan      TRG_PUTNIK        30000
Petar      TRG_PUTNIK        50000
Josip      TRG_PUTNIK       140000
Goran      TRG_PUTNIK            0
```

ORDER BY klauzula

- Kori{}enjem ORDER BY klauzule mogu}e je sortirati rezultuju}u tabelu po jednom ili vi{e atributa u rastu}em ili opadaju}em redosledu.
- Za specifikaciju rastu}eg redosleda koristi se klauzula ASC, a za specifikaciju opadaju}eg redosleda klauzula DESC. Rastu}i redosled se podrazumeva, pa klauzulu ASC nije neophodno navoditi, za razliku od klauzule DESC koju uvek treba navesti kada se sortira u opadaju}em redosledu.
- ORDER BY je uvek poslednja klauzula u SELECT bloku.

- *Prika`i ime, posao i platu radnika u odeljenju 30 uredjene u rastu}em redosledu poslova i opadaju}em redosledu plata.*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 30
ORDER BY POSAO ASC, PLATA DESC;

IME        POSAO             PLATA
---------- ------------ ----------
Janko      ANALITICAR         95000
Dragan     RUKOVODILAC       285000
Milan      TRG_PUTNIK        160000
Goran      TRG_PUTNIK        150000
Petar      TRG_PUTNIK        125000
Josip      TRG_PUTNIK        125000
```

Kada se sortiranje vr{i po koloni koja sadr`i NULL vrednosti, redosled prikaza n-torki sa NULL vrednostima u koloni sortiranja uslovljen je realizacijom SQL-a u konkretnom SUBP-u.

Upiti nad jednom tabelom uz prikaz modifikovanog sadr`aja tabele

- Kori{}enjem izraza, agregatnih funkcija i funkcija nad pojedina~nim redovima mogu}e je obraditi podatke baze podataka i prikazati izvedene podatke
- Funkcije i izrazi pi{u se u SELECT listi i ne menjaju sadr`aj tabela baze podataka.

Agregatne funkcije

- Pored prikazivanja prostih vrednosti memorisanih u tabelama baze podataka, SQL ima vi{e funkcija koje se koriste za dobijanje izvedenih, sumarnih informacija.
- Te funkcije se obi~no nazivaju agregatnim funkcijama.
- Primena agregatne funkcije zahteva da redovi tabele na koju se agregatna funkcija primenjuje budu grupisani na neki na~in.
- Svaka agregatna funkcija generi{e jedan rezultuju}i red za svaku grupu redova tabele. Ukoliko grupisanje redova tabele nije eksplicitno specificirano, ~itava tabela tretira se kao jedna grupa.
- Najzna~ajnije agregatne funkcije su:

  AVG (naziv_kolone) - izra~unava srednju vrednost

  SUM (naziv_kolone) - izra~unava ukupnu vrednost

  MIN (naziv_kolone) - nalazi minimalnu vrednost

  MAX (naziv_kolone) - nalazi maksimalnu vrednost

  COUNT (*) - nalazi broj redova u grupi

  COUNT ([**ALL**] naziv_kolone) - nalazi broj definisanih (not null) vrednosti kolone

  COUNT (**DISTINCT** naziv_kolone) - nalazi broj razli~itih definisanih (not null) vrednosti kolone

- *Na}i minimalnu, srednju i maksimalnu platu, kao i broj radnika u odeljenju 10*

```
SELECT MIN (PLATA), AVG (PLATA),
       MAX (PLATA), COUNT (*)
  FROM RADNIK
 WHERE ODELJENJE# = 10;

MIN(PLATA) AVG(PLATA) MAX(PLATA)   COUNT(*)
---------- ---------- ---------- ----------
    130000 291666.667     500000          3
```

- *Na}i ukupnu platu i ukupnu premiju za trgova~ke putnike*

```
SELECT SUM (PLATA), SUM (PREMIJA)
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

SUM(PLATA) SUM(PREMIJA)
---------- ------------
    560000       220000
```

- Pored toga {to argumenti agregatnih funkcija mogu biti izrazi, i same agregatne funkcije mogu biti operandi izraza. To je ilustrovano slede}im upitom:
- *Koliko je srednje godi{nje primanje trgova~kih putnika*

```
SELECT AVG (PLATA + PREMIJA) * 12
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

AVG(PLATA+PREMIJA)*12
---------------------
              2340000
```

GROUP BY klauzula

- Prouzrokuje dobijanje jednog rezultuju}eg reda za svaku razli~itu vrednost kolone po kojoj se vr{i grupisanje.
- *Na}i minimalnu, srednju i maksimalnu platu kao i ukupan broj radnika u svakom odeljenju*

```
SELECT MIN (PLATA), AVG (PLATA),
       MAX (PLATA), COUNT (*), ODELJENJE#
FROM RADNIK
GROUP BY ODELJENJE#;
```

```
MIN(PLATA) AVG(PLATA) MAX(PLATA)   COUNT(*) ODELJENJE#
---------- ---------- ---------- ---------- ----------
    130000 291666.667     500000          3         10
     80000     217500     300000          5         20
     95000 156666.667     285000          6         30
```

- Grupisanje se mo`e vr{iti po vi{e kolona. U tom slu~aju svaka razli~ita postoje}a kombinacija vrednosti kolona ~ini grupu.
- *Izra~unati broj radnika koji obavljaju razli~iti posao unutar svakog odeljenja*

```
SELECT ODELJENJE#, POSAO, COUNT(*)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO;

ODELJENJE# POSAO          COUNT(*)
---------- ------------ ----------
        10 ANALITICAR            1
        10 PREDSEDNIK            1
        10 RUKOVODILAC           1
        20 ANALITICAR            2
        20 RUKOVODILAC           1
        20 SAVETNIK              2
        30 ANALITICAR            1
        30 RUKOVODILAC           1
        30 TRG_PUTNIK            4
```

HAVING klauzula

- odredjuje kriterijume za selekciju grupa koje su prethodno specificirane GROUP BY klauzulom .
- *Prikazati koje poslove obavlja vi{e od dva radnika u svakom odeljenju*

```
SELECT ODELJENJE#, POSAO, COUNT(*)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO
HAVING COUNT (*) > 2;

ODELJENJE# POSAO          COUNT(*)
---------- ------------ ----------
        30 TRG_PUTNIK             4
```

- SQL naredbe mogu sadr`ati aritmeti~ke izraze sastavljene od imena kolona i konstantnih vrednosti povezanih aritmeti~kim operatorima (“+”, “*”, “-”, “/”)
- *Koliko je srednje godi{nje primanje svih trgova~kih putnika.*

```
SELECT AVG (PLATA + PREMIJA) * 12
FROM RADNIK
WHERE POSAO = 'TRG_PUTNIK';

AVG(PLATA + PREMIJA)*12
-----------------------
                2340000
```

- WHERE i GROUP BY klauzula mogu se koristiti zajedno, pri ~emu WHERE klauzula uvek ide pre GROUP BY klauzule.
- *Odrediti srednju godi{nju platu za svako odeljenje. Pri ra~unanju ne uzimati u obzir predsednika i rukovodioce odeljenja.*

```
SELECT ODELJENJE#, AVG (PLATA) * 12
FROM RADNIK
WHERE POSAO NOT IN ('RUKOVODILAC', 'PREDSEDNIK')
GROUP BY ODELJENJE#;

ODELJENJE# AVG(PLATA)*12
---------- -------------
        10       1560000
        20       2370000
        30       1572000
```

Funkcije EVERY, ANY i SOME

- Pored navedenih najzna~ajnijih agregatnih funkcija (AVG, SUM, MIN, MAX i COUNT), SQL:1999 standard uvodi i funkcije:
  - EVERY
  - ANY i
  - SOME
- Ove funkcije su definisane nad logi~kim (BOOLEAN) tipovima podataka.
  - Funkcija EVERY vrati}e vrednost TRUE (istinito) ako i samo ako je vrednost izraza, koji je argument funkcije, istinita (TRUE) u svakom redu grupe nad kojom je funkcija definisana.
  - Funkcije ANY i SOME su sinonimi. One vra}aju vrednost TRUE ako i samo ako je vrednost izraza, koji je argument funkcije, istinita bar u jednom redu grupe nad kojom je funkcija definisana.

Funkcije i izrazi za obradu pojedina~nih redova

- SQL:1999 standard sadr`i ~etiri tipa funkcija za obradu pojedina~nih redova. To su:
  - Numeri~ke funkcije
  - Tekstualne funkcije
  - Datumske funkcije
  - Intervalne funkcije
- Funkcije su grupisane prema tipu rezultuju}e vrednosti. Numeri~ka funkcija, na primer, mo`e imati argumente bilo kog tipa, ali kao rezultat uvek vra}a numeri~ku vrednost.

Numeri~ke funkcije za obradu pojedina~nih redova

- Najzna~ajnije numeri~ke funkcije su
  - POSITION
  - CHARACTER_LENGTH
  - OCTET_LENGTH
  - BIT_LENGTH
  - EXTRACT
  - ABS
  - MOD

Funkcija POSITION

- Funkcija POSITION ima slede}u sintaksu:

```
POSITION (string1 IN string2)
```

- String je bilo koji, eksplicitno ili implicitno odredjeni niz karaktera.
- Funkcija POSITION tra`i string1 u string2. Ukoliko ga nadje, vra}a njegovu po~etnu poziciju, ina~e, ukoliko ga ne nadje, vra}a vrednost 0.

- Slede neki primeri primene funkcije:

```
POSITION ('4' IN '48 sati')
```

vra}a vrednost 1

```
POSITION ('48' IN '48 sati')
```

vra}a vrednost 1

```
POSITION ('48' IN 'Narednih 48 sati')
```

vra}a vrednost 10

```
POSITION ('Zabava' IN 'Bolji zivot')
```

vra}a vrednost 0 ukazuju}i da tra`eni string nije pronadjen

```
POSITION ('' IN 'Istok Zapad')
```

vra}a vrednost 1; ako je string1 du`ine 0 funkcija uvek vra}a vrednost 1

```
POSITION (B'101' IN B'000101001010')
```

vra}a vrednost 4

- Oracle trenutno ne podr`ava funkciju POSITION.
- Umesto nje na raspolaganju je logi~ki ekvivalentna funkcija INSTR slede}e sintakse:

  `INSTR (string2, string1 [, pocetak])-` tra`i string1 u string2 polaze}i od pozicije "pocetak". Ako je nadjen, vra}a se njegova pozicija, ina~e 0
- Primer:

  `INSTR ('48 sati', '4') -` vra}a vrednost 1

Funkcija EXTRACT

- Funkcija EXTRACT ima slede}u sintaksu:

```
EXTRACT (polje_datuma FROM datum)
EXTRACT (polje_datuma FROM interval)
EXTRACT (vremenska_zona FROM datum)
```

- Datum uklju~uje vreme i vremensku zonu, polje_datuma je datumski ili vremenski deo datuma (godina, mesec, ..., sekund), a interval je godina-mesec (year-month) ili dan-vreme (day-time) interval.
- Funkcija izdvaja datumski ili vremenski deo iz datuma ili intervala, odnosno vremensku zonu iz datuma.

- Slede neki primeri primene funkcije:

```
EXTRACT (YEAR FROM DATE '1996-06-28')
```

vra}a vrednost 1996

```
EXTRACT (MONTH FROM DATE '1996-06-28')
```

vra}a vrednost 6

```
EXTRACT (DAY FROM DATE '1996-06-28')
```

vra}a vrednost 28

```
EXTRACT (HOUR FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 18

```
EXTRACT (MINUTE FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 0

```
EXTRACT (SECOND FROM TIMESTAMP '1996-06-28 18:00:01.99')
```

vra}a vrednost 1.99

```
EXTRACT (HOUR FROM INTERVAL '12 17:10:30.11' DAY(2) TO SECOND)
```

vra}a vrednost 17

```
EXTRACT (SECOND FROM INTERVAL '10:30' MINUTE TO SECOND)
```

vra}a vrednost 30

Funkcije CHARACTER_LENGTH i CHAR_LENGTH

- Funkcije CHARACTER_LENGTH i CHAR_LENGTH su sinonimi. Njihova sintaksa je:

  ```
  CHARACTER_LENGTH (string)
  CHAR_LENGTH (string).
  ```

- Funkcija vra}a numeri~ku vrednost koja odgovara du`ini stringa u karakterima.
- Slede neki primeri primene funkcija:

  ```
  CHARACTER_LENGTH ('Baze podataka')
  ```

  - vra}a vrednost 13

  ```
  CHAR_LENGTH ('Istinite lazi')
  ```

  - vra}a vrednost 13

- *Koliko su duga imena odeljenja ?*

```
SELECT NAZIV, CHAR_LENGTH (NAZIV) AS DUZINA
FROM ODELJENJE;

NAZIV                DUZINA
-------------- ----------
RAD_ZAJ                   7
PRIPREMA                  8
PROJEKTOVANJE            13
ISTRAZIVANJE             12
```

- Logi~ki ekvivalentna funkcija u Oracle-u je funkcija LENGTH slede}e sintakse:

```
LENGTH (string)
```

Funkcije OCTET_LENGTH i BIT_LENGTH

- Funkcije OCTET_LENGTH i BIT_LENGTH imaju slede}u sintaksu:

```
OCTET_LENGTH (string)
BIT_LENGTH (string)
```

- OCTET_LENGTH vra}a du`inu stringa u oktetima, a BIT_LENGTH u bitovima.
- Dalje slede neki primeri primene funkcija:

  `OCTET_LENGTH ('Baze podataka')`- vra}a vrednost 13, podrazumevaju}i jedan oktet po karakteru; za multioktet implementacije vra}ena vrednost bi bila razli~ita

  `OCTET_LENGTH (B'1011100001')`- vra}a vrednost 2, jer 10 bita zahteva bar 2 okteta za reprezentaciju (oktet = 8 bita)

  `BIT_LENGTH (B'01111110')` - vra}a vrednost 8
- Oracle trenutno ne podr`ava funkcije OCTET_LENGTH i BIT_LENGTH

Funkcije ABS i MOD

- Funkcija ABS ima slede}u sintaksu:

  ```
  ABS (numerik)
  ```

- Funkcija ABS nalazi apsolutnu vrednost broja.

- Funkcija MOD ima slede}u sintaksu:

  ```
  MOD (numerik1, numerik2).
  ```

- Funkcija nalazi ostatak deljenja prve numeri~ke vrednosti drugom
- Slede primeri:

  `MOD (35, 3)` - vra}a vrednost 2

  `MOD (35, -8)` - vra}a vrednost 3

- Komercijalni relacioni sistemi obi~no imaju {iri skup funkcija u odnosu na funkcije definisane SQL standardom.
- Njima se zna~ajno pove}ava efikasnost rada u konkretnom implementacionom okru`enju.
- Dalje sledi spisak nekih numeri~kih funkcija koje su raspolo`ive u Oracle okru`enju:

  `POWER (broj, e)` - di`e broj na e-ti stepen

  `ROUND (broj [,d])`- zaokru`uje broj na d decimala

  `TRUNC (broj [,d])`- odbacuje ostatak od d-tog decimalnog mesta

- `SIGN (broj)` - vra}a +1 ako je broj >0, 0 ako je broj =0, a -1 ako je broj <0.

  `SQRT (broj)` - nalazi pozitivan kvadratni koren broja

- Broj, e i d su eksplicitne ili implicitne numeri~ke vrednosti.

- Slede}i primer ilustruje primenu funkcije ROUND na numeri~ki izraz
- *Koji radnici zaradjuju vi{e od 1000 dinara po satu. Zaradu po satu zaokru`iti na 2 decimale. (Podrazumeva se da postoje 22 radna dana u mesecu i 8 radnih sati u danu)*

```
SELECT IME,
       ROUND (PLATA / (22 * 8), 2) AS "ZARADA PO CASU"
FROM RADNIK
WHERE PLATA / (22 * 8) > 1000;

IME        ZARADA PO CASU
---------- --------------
Marko              1690.34
Dragan             1619.32
Ivan               1392.05
Pavle              1704.55
Jovan              2840.91
Filip              1704.55
```

Tekstualne funkcije za obradu pojedina~nih redova

- Najzna~ajnije tekstualne funkcije su:
  - SUBSTRING
  - UPPER
  - LOWER
  - TRIM i
  - OVERLAY
- Funkcija SUBSTRING ima slede}u sintaksu:

```
SUBSTRING (string FROM pocetak [FOR duzina])
```

  - String je bilo koji, eksplicitno ili implicitno odredjeni niz karaktera ili niz bitova.
  - Funkcija SUBSTRING izdvaja podniz iz datog niza (string) po~ev od startne pozicije (pocetak) u definisanoj du`ini (duzina).
  - To je mo}na funkcija jer za tekstualne kolone omogu}uje kontrolu vrednosti do nivoa pojedina~nog karaktera, a za binarne do nivoa pojedina~nog bita.

- Dalje slede neki primeri primene funkcije :

```
SUBSTRING ('Narednih 48 sati FROM 1 FOR 10)
```

vra}a vrednost 'Narednih 4'

```
SUBSTRING ('abcdef' FROM -8 FOR 2)
```

vra}a vrednost '' (prazan string)

```
SUBSTRING ('abcdef' FROM -2 FOR 6)
```

vra}a vrednost 'abc'

```
SUBSTRING ('abcdef' FROM 0 FOR 4)
```

vra}a vrednost 'abc'

```
SUBSTRING ('abcdef' FROM 3 FOR -2)
```

vra}a poruku o gre{ci

```
SUBSTRING ('abcdef' FROM 7 FOR 3)
```

vra}a vrednost '' (prazan string)

```
SUBSTRING ('abcdef' FROM 3)
```

vra}a vrednost 'cdef'

```
SUBSTRING (B'101101' FROM 3 FOR 2)
```

vra}a vrednost B'11'

- Logi~ki ekvivalentna u Oracle-u je funkcija SUBSTR slede}e sintakse:

```
SUBSTR (string, pocetak [, duzina])
SUBSTR ('PROJEKTOVANJE', 1, 4) -
```

vra}a vrednost 'PROJ'.

- Funkcija UPPER ima slede}u sintaksu:

  `UPPER (string)`

  – Funkcija menja sva mala slova u velika, kao u slede}em primeru:

  `UPPER ('Hocu da sam veliki !!!')`

  vra}a 'HOCU DA SAM VELIKI !!!'.

- Funkcija LOWER ima slede}u sintaksu:

  `LOWER (string)`

  – Funkcija menja sva velika slova u mala, kao u slede}em primeru:

  `LOWER ('LAKSE JE BITI MALI !!!')`

  vra}a 'lakse je biti mali !!!'.

- Funkcija TRIM ima slede}u sintaksu:

  `TRIM ([BOTH|LEADING|TRAILING] karakter FROM string)`

  – Funkcija elimini{e navedeni karakter sa po~etka i kraja (opcija BOTH), samo sa po~etka (opcija LEADING) ili samo sa kraja (opcija TRAILING) datog stringa.

  `TRIM (LEADING ' ' FROM ' TEST ')-` vra}a vrednost 'TEST ‘

  `TRIM (TRAILING ' ' FROM ' TEST ')-` vra}a vrednost ' TEST‘

  `TRIM (BOTH ' ' FROM ' TEST ')-` vra}a vrednost 'TEST‘

  `TRIM (BOTH 'T' FROM 'TEST')-` vra}a vrednost 'ES'.

- Funkcija OVERLAY ima slede}u sintaksu:

```
OVERLAY (string1 PLACING
          string2 FROM pocetak [FOR duzina]).
```

  – Funkcija omogu}uje zamenu dela niza karaktera ili niza bitova (string1) novim nizom (string2) od date startne pozicije (pocetak) u navedenoj du`ini (duzina), kao u slede}em primeru :

```
OVERLAY ('Nikad subotom' PLACING 'Uvek' FROM 1 FOR 5)
```

  vra}a vrednost 'Uvek subotom'.

- Pored navedenih funkcija za rad sa tekstualnim i binarnim podacima na raspolaganju je i operator konkatenacije (||).
  – On omogu}uje formiranje izraza kojima se spajaju nizovi karaktera ili nizovi bitova.
  – Sintaksa operatora konkatenacije:

```
string1 || string2
```

- *Prika`i imena radnika odeljenja 30 iza kojih neposredno treba da dodje posao koji obavljaju. Sortirati rezultuju}u tabelu u rastu}em redosledu vrednosti kolone POSAO*

```
SELECT IME||', '||POSAO AS RADNIK
FROM RADNIK
WHERE ODELJENJE# = 30
ORDER BY POSAO;

RADNIK
-------------------------
Janko, ANALITICAR
Dragan, RUKOVODILAC
Milan, TRG_PUTNIK
Petar, TRG_PUTNIK
Goran, TRG_PUTNIK
Josip, TRG_PUTNIK
```

Datumske funkcije za obradu pojedina~nih redova

- SQL standard podr`ava slede}e datumske funkcije:
  - CURRENT_DATE
  - CURRENT_TIME
  - CURRENT_TIMESTAMP
  - LOCALTIME i
  - LOCALTIMESTAMP.
- Prve tri funkcije definisane su SQL-92 standardom, a poslednje dve su uvedene u SQL:1999 standardu.
- Funkcija CURRENT_DATE je bez argumenata. Ona prikazuje teku}i datum, na primer 2002-01-07. Vra}ena vrednost je tipa DATE.
- Ostale funkcije imaju jedan argument, preciznost, kojim je odredjen deo sekunde (desetinka, stotinka, hiljaditi deo itd.) sa kojim je potrebno iskazati ta~nost vremena.

- Funkcije CURRENT_TIME i LOCALTIME prikazuju teku}e vreme
  - Prva funkcija daje teku}e vreme sa vremenskim odstupanjem od univerzalno koordinisanog vremena (UTC) za datu vremensku zonu, a druga je bez odstupanja za vremensku zonu.
  - Rezultuju}a vrednost prve funkcije je tipa TIME WITH TIME ZONE, a druge TIME.
- Sledi sintaksa i primeri primene navedenih funkcija:

```
CURRENT_TIME [(preciznost)]
LOCALTIME [(preciznost)]
```

`CURRENT_TIME (2)` - vra}a teku}e vreme, npr. 17:12:30.32 +01:00, sa ta~no{}u na stote delove sekunda i sa vremenskim odstupanjem

`LOCALTIME (2)` - vra}a teku}e vreme, npr. 17:12:30.32, sa ta~no{}u na stote delove sekunda.

- Funkcije CURRENT_TIMESTAMP i LOCALTIMESTAMP
  - prikazuju teku}e "datum + vreme“
  - prva funkcija vr{i prikaz sa odstupanjem od univerzalno koordinisanog vremena (UTC) za datu vremensku zonu
  - druga je bez odstupanja za vremensku zonu.
  - Rezultuju}a vrednost prve funkcije je tipa TIMESTAMP WITH TIME ZONE, a druge TIMESTAMP.
- Sledi sintaksa i primeri primene navedenih funkcija:

  `CURRENT_TIMESTAMP [(preciznost)]`

  `LOCALTIMESTAMP [(preciznost)]`

  `CURRENT_TIMESTAMP (1)` - vra}a teku}e "datum + vreme", npr. 2002-01-07:17:12:30.3 +01:00, sa ta~no{}u na desete delove sekunda i sa vremenskim odstupanjem

  `LOCALTIMESTAMP (1)`- vra}a teku}e "datum + vreme", npr. 2002-01-07:17:12:30.3, sa ta~no{}u na desete delove sekunda.

- Navedene datumske funkcije, kombinovane sa vrednostima intervalnog tipa, mogu formirati izraze sa rezultuju}om vredno{}u datumskog tipa.
- Na primer, izraz

  ```
  CURRENT_DATE + INTERVAL '1' DAY
  ```

  vrati}e istu vrednost koju }e funkcija CURRENT_DATE imati sutra.
- Medjutim, ako od jednog datuma oduzmemo drugi datum, dobi}emo rezultuju}u vrednost intervalnog tipa, kao u slede}em slu~aju

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

Intervalne funkcije za obradu pojedina~nih redova

- Postoji samo jedna intervalna funkcija, a uvedena je u SQL:1999 standardu.
- To je funkcija ABS i potpuno je analogna funkciji ABS koja se primenjuje na numeri~ke vrednosti.
- Funkcija ABS ima jedan operand, koji mora biti intervalnog tipa, i vra}a rezultuju}u vrednost koja je istog tipa i iste preciznosti kao i operand funkcije.
- Na primer, funkcija :

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

vrati}e slede}u rezultuju}u vrednost:

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

Izrazi CASE, NULLIF, COALESCE i CAST

- Pored prethodno navedenih osnovnih funkcija i izraza, SQL:1999 standard podr`ava i kompleksne uslovne izraze (CASE), izraze kojima se konvertuju podaci jednog tipa u drugi (CAST) i specijalne slu~ajeve uslovnih, CASE izraza (NULLIF i COALESCE).
- CASE izraz ima slede}u sintaksu:

```
CASE
   WHEN uslov1 THEN rezultat1
   WHEN uslov2 THEN rezultat2
   ...
   WHEN uslovn THEN rezultatn
   ELSE rezultatx
END.
```

- CASE izraz je sli~an CASE naredbi nekog klasi~nog programskog jezika.
  - Su{tinska razlika je u tome da CASE izraz nije izvr{na naredba, ve} uslovni izraz, koji se u SQL naredbama mo`e koristiti na bilo kom mestu gde je dozvoljeno kori{}enje vrednosnog izraza.
- Jedna od va`nih primena CASE izraza je za transformaciju nedefinisane (NULL) vrednosti u odredjenu konkretnu vrednost.
  - NULL vrednost se ne koristi pri izra~unavanju izraza i funkcija. Da bi se izra~unavanje ipak omogu}ilo koristi se CASE izraz koji privremeno, unutar upita, menja NULL vrednost sa vredno{}u za koju se sami odlu~imo, naj~e{}e vredno{}u koja je neutralna u odnosu na `eljenu operaciju.

- *Prika`i ukupnu mese~nu zaradu radnika u odeljenju 30*

```
SELECT IME, POSAO, PLATA, PREMIJA,
       PLATA + (CASE WHEN PREMIJA IS NULL THEN 0
                     ELSE PREMIJA
                END) AS PRIMANJA
FROM RADNIK
WHERE ODELJENJE# = 30;
```

IME	POSAO	PLATA	PREMIJA	PRIMANJA
Milan	TRG_PUTNIK	160000	30000	190000
Petar	TRG_PUTNIK	125000	50000	175000
Josip	TRG_PUTNIK	125000	140000	265000
Dragan	RUKOVODILAC	285000		285000
Goran	TRG_PUTNIK	150000	0	150000
Janko	ANALITICAR	95000		95000

- Sli~an efekat CASE izraz ima i kao argument funkcija
- *Za odeljenje 30 izra~unaj srednju platu, srednju premiju, srednju mese~nu zaradu za radnike koji primaju premiju i srednju mese~nu zaradu za sve radnike.*

```
SELECT AVG (PLATA), AVG (PREMIJA),
       AVG (PLATA + PREMIJA),
       AVG (PLATA + (CASE WHEN PREMIJA IS NULL THEN 0
                          ELSE PREMIJA
                     END)) AS "PROSEK SVIH"
FROM RADNIK
WHERE ODELJENJE# = 30;

AVG(PLATA) AVG(PREMIJA) AVG(PLATA+PREMIJA) PROSEK SVIH
---------- ------------ ------------------ -----------
156666.667        55000             195000  193333.333
```

- Mogu}e je koristiti i skra}eni oblik CASE izraza, koji je slede}eg izgleda :

```
CASE izraz
   WHEN izraz1 THEN rezultat1
   WHEN izraz2 THEN rezultat2
   ...
   WHEN izrazn THEN rezultatn
   ELSE rezultatx
END
```

- Navedeni CASE izraz je samo skra}ena verzija slede}eg op{teg CASE izraza:

```
CASE
   WHEN izraz = izraz1 THEN rezultat1
   WHEN izraz = izraz2 THEN rezultat2
   ...
   WHEN izraz = izrazn THEN rezultatn
   ELSE rezultatx
END.
```

- Primenu skra}enog oblika CASE izraza ilustrova}emo slede}im primerom:
- *Koriste}i kolonu POSAO formiraj kolonu KLASA tako {to za posao analiti~ara vrednost klase treba da bude 1, rukovodioca - 3, predsednika - 5, a za svaki drugi - 2*

```
SELECT IME, POSAO,
   CASE POSAO WHEN 'ANALITICAR' THEN 1
              WHEN 'RUKOVODILAC' THEN 3
              WHEN 'PREDSEDNIK' THEN 5
              ELSE 2
   END AS KLASA
FROM RADNIK;
```

IME	POSAO	KLASA
Steva	ANALITICAR	1
Milan	TRG_PUTNIK	2
Petar	TRG_PUTNIK	2
Marko	RUKOVODILAC	3
Josip	TRG_PUTNIK	2
Pavle	SAVETNIK	2
Jovan	PREDSEDNIK	5
...		

- Korisnici se ponekad, i pored toga {to SQL podr`ava rad sa NULL vrednostima, odlu~uju da NULL vrednosti predstave na tradicionalan na~in, nekom konkretnom vredno{}u, recimo -1, koja se ne pojavljuje kao mogu}a vrednost posmatrane kolone.
- Ako `elimo da takvoj koloni vratimo SQL funkcionalnost rada sa NULL vrednostima, mo`emo koristiti slede}i CASE izraz:

```
CASE
   WHEN naziv_kolone = -1 THEN NULL
   ELSE naziv_kolone
END.
```

- Kako je procenjeno da se potreba za prethodnom transformacijom relativno ~esto javlja, uveden je NULLIF izraz slede}e sintakse:

  NULLIF ($izraz_1$, $izraz_2$).

- Semantika NULLIF izraza je:
  - ako je vrednost izraza1 jednaka vrednosti izraza2, vrati NULL vrednost, ina~e vrati vrednost izraza1.
  - NULLIF izraz za prikazani CASE izraz bio bi slede}eg izgleda:

```
NULLIF (naziv_kolone, -1).
```

- COALESCE izraz ima slede}u sintaksu:

  COALESCE ($izraz_1$, $izraz_2$, ..., $izraz_n$).

  – Ako je vrednost izraza1 definisana, not null vrednost, rezultat COALESCE izraza je vrednost izraza1.
  – Ako je vrednost izraza1 nedefinisana, null vrednost, vr{i se provera vrednosti izraza2 na identi~an na~in.
  – COALESCE izraz vra}a vrednost NULL ako nijedna od vrednosti izraza iz liste nije definisana, not null vrednost.
  – COALESCE izraz je samo specijalan slu~aj CASE izraza.
  – COALESCE ($izraz_1$, $izraz_2$, $izraz_3$) je ekvivalent slede}oj CASE naredbi:

```
CASE
   WHEN izraz1 IS NOT NULL THEN izraz1
   WHEN izraz2 IS NOT NULL THEN izraz2
   ELSE izraz3
END.
```

- Prikaz ukupne mese~ne zarade radnika u odeljenju 30 mogli smo, kori{}enjem COALESCE izraza, re{iti i na slede}i na~in:

```
SELECT IME, POSAO, PLATA, PREMIJA,
   COALESCE (PLATA + PREMIJA, PLATA) AS PRIMANJA
FROM RADNIK
WHERE ODELJENJE# = 30;
```

IME	POSAO	PLATA	PREMIJA	PRIMANJA
Milan	TRG_PUTNIK	160000	30000	190000
Petar	TRG_PUTNIK	125000	50000	175000
Josip	TRG_PUTNIK	125000	140000	265000
Dragan	RUKOVODILAC	285000		285000
Goran	TRG_PUTNIK	150000	0	150000
Janko	ANALITICAR	95000		95000

- CAST izraz ima slede}u sintaksu
  `CAST (izraz AS tip_podataka).`
  - Ona omogu}uje transformaciju vrednosti jednog tipa podataka u drugi.
  - Dalje sledi primer primene funkcije:
- *Za svakog radnika odeljenja 30 prika`i {ifru, iza koje neposredno treba da dodje ime, a iza imena posao koji obavlja.*

```
SELECT CAST (SRADNIK AS CHAR(4))||', '||IME||', '||POSAO
       AS RADNIK
FROM RADNIK
WHERE ODELJENJE# = 30;
 RADNIK
------------------------------
3199, Milan, TRG_PUTNIK
3221, Petar, TRG_PUTNIK
3354, Josip, TRG_PUTNIK
3398, Dragan, RUKOVODILAC
3544, Goran, TRG_PUTNIK
```

Ulaganje upita nad jednom relacijom u upit nad drugom relacijom

- U SQL-u rezultat jednog upita mo`e biti dinami~ki zamenjen u WHERE klauzuli drugog upita.
- *Prikazati ime i posao svakog radnika koji ima isti posao kao Dejan*

```
SELECT IME, POSAO
  FROM RADNIK
 WHERE POSAO = (SELECT POSAO FROM RADNIK
                WHERE IME = 'Dejan');
```

```
IME        POSAO
---------- ------------
Steva      ANALITICAR
Adam       ANALITICAR
Janko      ANALITICAR
Dejan      ANALITICAR
```

- *Prikazati ime i posao radnika koji rade u Beogradu*

```
SELECT IME, POSAO
FROM RADNIK
WHERE ODELJENJE# = (SELECT ODELJENJE#
                    FROM ODELJENJE
                    WHERE GRAD = 'BEOGRAD');
```

```
IME        POSAO
---------- ------------
Ivan       RUKOVODILAC
Jovan      PREDSEDNIK
Dejan      ANALITICAR
```

- *Prikazati ime, posao i platu radnika u odeljenju 20 koji imaju isti posao kao radnici odeljenja projektovanje*

```
SELECT IME, POSAO, PLATA
FROM RADNIK
WHERE ODELJENJE# = 20
  AND POSAO IN
         (SELECT POSAO FROM RADNIK
          WHERE ODELJENJE# IN
                 (SELECT ODELJENJE# FROM ODELJENJE
                  WHERE NAZIV = 'PROJEKTOVANJE'));
```

```
IME        POSAO              PLATA
---------- ------------ ----------
Steva      ANALITICAR          80000
Adam       ANALITICAR         110000
Marko      RUKOVODILAC        297500
```

- *Ko je najbolje pla}eni radnik u svakom odeljenju?*

```
SELECT IME, ODELJENJE#, POSAO, PLATA
FROM RADNIK
WHERE (ODELJENJE#, PLATA) IN
          (SELECT ODELJENJE#, MAX (PLATA)
           FROM RADNIK
           GROUP BY ODELJENJE#);
```

IME	ODELJENJE#	POSAO	PLATA
Jovan	10	PREDSEDNIK	500000
Pavle	20	SAVETNIK	300000
Filip	20	SAVETNIK	300000
Dragan	30	RUKOVODILAC	285000

- Upit koji se izvr{ava jedanput za svaki selektovani red spoljnog upita
- *Prika`i {ifru odeljenja, ime i platu svakog radnika ~ija je plata ve}a od prose~ne plate svog odeljenja.*

```
SELECT ODELJENJE#, IME, PLATA
FROM RADNIK R
WHERE PLATA > (SELECT AVG(PLATA)
               FROM RADNIK
               WHERE ODELJENJE# = R.ODELJENJE#);
```

```
ODELJENJE# IME          PLATA
---------- ---------- -------
        30 Milan       160000
        20 Marko       297500
        30 Dragan      285000
        20 Pavle       300000
        10 Jovan       500000
        20 Filip       300000
```

Primena operatora unije (UNION), preseka (INTERSECT) i razlike (EXCEPT)

- Da bi skupovne operacije mogle da se primene SELECT blokovi (upiti) na koje se operacije odnose moraju imati isti broj rezultuju}ih kolona i rezultuju}e kolone moraju odgovarati po tipu.
- U cilju ilustracije da}emo primer primene operacije EXCEPT na dve tabele. Pretpostavimo da je kreirana i tabela o trgova~kim putnicima ~ija bi shema bila slede}a:

  TRGOVACKI_PUTNIK (SRADNIK, PREMIJA).
- *Zahtev da se prika`u {ifre radnika koji nisu trgova~ki putnici mogao bi da se re{i na slede}i na~in*

```
SELECT SRADNIK                  SRADNIK
FROM RADNIK                    ----------
EXCEPT                            3069
SELECT SRADNIK                    3266
FROM TRGOVACKI_PUTNIK;            3398
                                   ...
```

- SQL standard je odstupio od relacione teorije i dozvoljava postojanje duplikata u rezultatu primene navedenih skupovnih operacija.
- O~uvanje duplikata - klju~na re~ ALL odmah iza naziva operatora.
- Eliminisanje duplikata - klju~na re~ DISTINCT (DISTINCT se podrazumeva po default-u).
- U cilju potpunog razja{njenja pokaza}emo primenu ovih operatora sa klauzulama ALL i DISTINCT na slede}e dve tabele

```
T1                  T2

ID NAZIV            ID NAZIV
-- -----            -- -----
10 AAA              10 AAA
10 AAA              10 AAA
10 AAA              40 DDD
20 BBB              50 EEE
30 CCC              60 FFF
40 DDD
```

```
SELECT *
FROM T1
UNION ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
10 AAA
10 AAA
10 AAA
10 AAA
10 AAA
20 BBB
30 CCC
40 DDD
40 DDD
50 EEE
60 FFF
```

```
SELECT *
FROM T1
UNION DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
40 DDD
50 EEE
60 FFF
```

```
SELECT *
FROM T1
INTERSECT ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
10 AAA
40 DDD
```

```
SELECT *
FROM T1
INTERSECT DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
40 DDD
```

```
SELECT *
FROM T1
EXCEPT ALL
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
```

```
SELECT *
FROM T1
EXCEPT DISTINCT
SELECT *
FROM T2;

ID NAZIV
-- -----
20 BBB
30 CCC
```

- Pored ALL i DISTINCT neposredno iza naziva operatora mo`e se navesti i klju~na re~ CORRESPONDING.
- Ona omogu}uje primenu operatora UNION, INTERSECT i EXCEPT na tabele koje nisu kompatibilne na uniju, ali imaju neke 'zajedni~ke' kolone (kolone identi~nih naziva).
- CORRESPONDING mo`e biti sa listom ili bez liste zajedni~kih kolona.
- Za ilustraciju, posmatrajmo slede}e dve tabele:

T1

ID	NAZIV	TEZINA
10	AAA	100
10	AAA	100
10	AAA	100
20	BBB	200
30	CCC	400
40	DDD	300

T2

ID	NAZIV	BOJA
10	AAA	PLAVA
10	AAA	PLAVA
40	DDD	ZELENA
50	EEE	BELA
60	FFF	CRNA

- Mada tabele T1 i T2 nisu kompatibilne na uniju, kori{}enjem klauzule CORRESPONDING operator UNION (takodje i INTERSECT i EXCEPT) se mo`e primeniti na njih na slede}i na~in:

```
SELECT *
FROM T1
UNION CORRESPONDING
SELECT *
FROM T2;

ID NAZIV
-- -----
10 AAA
20 BBB
30 CCC
40 DDD
50 EEE
60 FFF
```

- Pri obradi i izvr{avanju prikazanog upita odigrava se slede}a logi~ka sekvenca akcija:
  - Zvezdica (*) iz SELECT liste prevodi se u kompletnu listu kolona tabele.
  - Lista se redukuje tako da ostaju samo kolone koje su implicitno ili eksplicitno odredjene CORRESPONDING klauzulom.
  - Rezultuju}a tabela ne sadr`i duplikate, jer je DISTINCT default opcija.
- Napomenimo na kraju da bi se identi~an rezultat dobio izvr{avanjem slede}e naredbe:

```
SELECT ID, NAZIV
FROM T1
UNION
SELECT ID, NAZIV
FROM T2;
```

JOIN - spajanje dve ili vi{e tabela

- Kada su nam potrebni podaci iz vi{e tabela, koristi se JOIN. JOIN povezuje n-torke razli~itih tabela koriste}i zajedni~ke atribute.
- U skladu sa relacionom teorijom podr`ani su razli~iti tipovi spajanja:
  - dekartov proizvod
  - ekvispajanje (spajanje na jednakost)
  - prirodno spajanje i
  - spoljno spajanje.
- Operacija spajanja mo`e se zahtevati:
  - implicitno - zadavanjem uslova spajanja u WHERE klauzuli upita
    - Ovaj na~in je klasi~an na~in spajanja, koji je bio i jedini u SQL-86 i SQL-89 standardu.
  - eksplicitno - navodjenjem tipa i uslova spajanja u FROM klauzuli upita
    - SQL-92 i SQL:1999 standard, pored implicitnog, omogu}uju i eksplicitno navodjenje tipa spajanja.

Ekvispajanje

- *Prika`i za svakog radnika ime, posao i podatke o odeljenju u kom radi.*

```
SELECT IME, POSAO, RADNIK.ODELJENJE#,
       ODELJENJE.ODELJENJE#, NAZIV, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;
```

IME	POSAO	ODELJENJE#	ODELJENJE#	NAZIV	GRAD
Ivan	RUKOVODILAC	10	10	RAD_ZAJ	BEOGRAD
Jovan	PREDSEDNIK	10	10	RAD_ZAJ	BEOGRAD
Dejan	ANALITICAR	10	10	RAD_ZAJ	BEOGRAD
Steva	ANALITICAR	20	20	PRIPREMA	NIS
Adam	ANALITICAR	20	20	PRIPREMA	NIS
Filip	SAVETNIK	20	20	PRIPREMA	NIS
Pavle	SAVETNIK	20	20	PRIPREMA	NIS
Marko	RUKOVODILAC	20	20	PRIPREMA	NIS
Milan	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Dragan	RUKOVODILAC	30	30	PROJEKTOVANJE	BOR
Josip	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Janko	ANALITICAR	30	30	PROJEKTOVANJE	BOR
Goran	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR
Petar	TRG_PUTNIK	30	30	PROJEKTOVANJE	BOR

Pored prikazane klasi~ne sintaksne podr{ke operacije ekvispajanja, SQL-92 i SQL:1999 standard omogu}uju eksplicitno navodjenje operacije spajanja u FROM klauzuli na jedan od slede}a dva na~ina:

```
SELECT *
FROM RADNIK JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK JOIN ODELJENJE
     USING (ODELJENJE#);
```

Prirodno spajanje

- Ekvispajanje uvek daje rezultuju}u tabelu sa dve identi~ne kolone, odnosno kolone sa identi~nim sadr`ajem.
- Kada se iz rezultata ekvispajanja izbaci jedna od dve identi~ne kolone dobija se prirodno spajanje.
- Prirodno spajanje tabela RADNIK i ODELJENJE kori{}enjem klasi~ne sintakse mo`e se prikazati slede}om SQL naredbom:

```
SELECT RADNIK.*, NAZIV, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;
```

- Eksplicitnim navodjenjem operacije prirodnog spajanja dobija se slede}a SQL naredba:

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

- Uslov spajanja nije eksplicitno naveden. Spajanje se vr{i po svim kolonama sa identi~nim nazivima u obe tabele.

Uslov spajanja i uslov selekcije u istom upitu

- *Za svakog analiti~ara prika`i ime i grad u kome radi.*

  Ako se upit realizuje klasi~nom sintaksom i uslov spajanja i uslov selekcije bi}e u WHERE klauzuli.

```
SELECT IME, GRAD
FROM RADNIK, ODELJENJE
WHERE RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#
  AND POSAO = 'ANALITICAR';

IME        GRAD
---------- -------------
Dejan      BEOGRAD
Steva      NIS
Adam       NIS
Janko      BOR
```

Ako se upit realizuje eksplicitnim navodjenjem operacije spajanja u FROM klauzuli, u WHERE klauzuli osta}e samo uslov selekcije.

```
SELECT IME, GRAD
FROM RADNIK JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#
WHERE POSAO = 'ANALITICAR';

ili
SELECT IME, GRAD
FROM RADNIK JOIN ODELJENJE
     USING (ODELJENJE#)
WHERE POSAO = 'ANALITICAR';
```

Dekartov proizvod

- Ukoliko se u upitu izostavi uslov spajanja dobija se dekartov proizod
- *Ponovimo predhodni primer zapisan klasi~nom sintaksom izostavljaju}i uslov spajanja .*

```
SELECT IME, GRAD
FROM RADNIK, ODELJENJE
WHERE POSAO = 'ANALITICAR';
```

```
 IME        GRAD
---------- -------------
Steva      BEOGRAD
Adam       BEOGRAD
Janko      BEOGRAD
Dejan      BEOGRAD
Steva      NIS
Adam       NIS
Janko      NIS
Dejan      NIS
Steva      BOR
Adam       BOR
Janko      BOR
Dejan      BOR
Steva      SARAJEVO
Adam       SARAJEVO
Janko      SARAJEVO
Dejan      SARAJEVO
```

- Eksplicitnim navodjenjem operacije dekartovog proizvoda prethodni upit bi se realizovao SQL naredbom slede}eg izgleda:

```
SELECT IME, GRAD
FROM RADNIK CROSS JOIN ODELJENJE
WHERE POSAO = 'ANALITICAR';
```

Spajanje tabele sa samom sobom (SELF JOIN)

- Tabela mo`e biti spojena sa samom sobom po kolonama koje sadr`e isti tip informacija. SELF JOIN spaja redove tabele sa ostalim ili istim redovima te iste tabele.
- *Prika`i ime i posao svakog radnika i njegovog pretpostavljenog.*

```
SELECT PODR.IME, PODR.POSAO, PODR.SRUKOV,
       NADR.SRADNIK AS SEF,
       NADR.IME AS SEF_IME, NADR.POSAO AS SEF_POSAO
FROM RADNIK AS PODR, RADNIK AS NADR
WHERE PODR.SRUKOV = NADR.SRADNIK;
```

IME	POSAO	SRUKOV	SEF	SEF_IME	SEF_POSAO
Pavle	SAVETNIK	3266	3266	Marko	RUKOVODILAC
Filip	SAVETNIK	3266	3266	Marko	RUKOVODILAC
Milan	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Petar	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Janko	ANALITICAR	3398	3398	Dragan	RUKOVODILAC
Goran	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Josip	TRG_PUTNIK	3398	3398	Dragan	RUKOVODILAC
Dejan	ANALITICAR	3482	3482	Ivan	RUKOVODILAC
Adam	ANALITICAR	3488	3488	Pavle	SAVETNIK
Marko	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Ivan	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Dragan	RUKOVODILAC	3539	3539	Jovan	PREDSEDNIK
Steva	ANALITICAR	3602	3602	Filip	SAVETNIK

- Eksplicitnim navodjenjem operacije spajanja prethodni upit bi se realizovao SQL naredbom slede}eg izgleda:

```
SELECT PODR.IME, PODR.POSAO, PODR.SRUKOV,
       NADR.SRADNIK AS SEF,
       NADR.IME AS SEF_IME, NADR.POSAO AS
  SEF_POSAO
FROM RADNIK AS PODR JOIN RADNIK AS NADR
     ON PODR.SRUKOV = NADR.SRADNIK;
```

Spoljno spajanje (OUTER JOIN)

- Spajanje tabela mo`e biti unutra{nje (INNER) ili spoljno (OUTER).
- Ukoliko se vrsta spajanja ne navede eksplicitno podrazumeva se unutra{nje spajanje.
- Ponovimo primere za ekvispajanje i prirodno spajanje sa eksplicitnim navodjenjem klauzule za unutra{nje (INNER) spajanje:

```
SELECT *
FROM RADNIK INNER JOIN ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK NATURAL INNER JOIN ODELJENJE;
```

- Spoljno spajanje mo`e biti levo (LEFT), desno (RIGHT) i centralno (FULL).
  - Levo spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova tabele sa leve strane JOIN klauzule tako {to se praznim redom pro{iruje tabela sa desne strane.
  - Desno spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova tabele sa desne strane JOIN klauzule tako {to se praznim redom pro{iruje tabela sa leve strane.
  - Centralno spoljno spajanje omogu}uje uklju~ivanje u rezultuju}u tabelu svih redova i leve i desne tabele tako {to se obe tabele pro{iruju praznim redom.

- Levo, desno ili centralno spoljno spajanje tabela RADNIK i ODELJENJE mo`e se realizovati jednom od slede}e dve SQL naredbe:

```
SELECT *
FROM RADNIK {LEFT | RIGHT | FULL} [OUTER] JOIN
     ODELJENJE
     ON RADNIK.ODELJENJE# = ODELJENJE.ODELJENJE#;

SELECT *
FROM RADNIK {LEFT | RIGHT | FULL} [OUTER] JOIN
     ODELJENJE
     USING (ODELJENJE#);
```

- Prika`i naziv odeljenja kao i ime i posao svakog radnika koji u njemu radi, uklju~uju}i i odeljenja bez radnika.

  SQL naredba zapisana originalnim Oracle dijalektom bila bi slede}eg izgleda:

```
SELECT NAZIV, IME, POSAO
FROM ODELJENJE, RADNIK
WHERE ODELJENJE.ODELJENJE# = RADNIK.ODELJENJE#(+);
NAZIV          IME        POSAO
-------------- ---------- ------------
RAD_ZAJ        Ivan       RUKOVODILAC
RAD_ZAJ        Jovan      PREDSEDNIK
RAD_ZAJ        Dejan      ANALITICAR
PRIPREMA       Steva      ANALITICAR
PRIPREMA       Adam       ANALITICAR
PRIPREMA       Filip      SAVETNIK
PRIPREMA       Pavle      SAVETNIK
PRIPREMA       Marko      RUKOVODILAC
PROJEKTOVANJE  Milan      TRG_PUTNIK
PROJEKTOVANJE  Dragan     RUKOVODILAC
PROJEKTOVANJE  Josip      TRG_PUTNIK
PROJEKTOVANJE  Janko      ANALITICAR
PROJEKTOVANJE  Goran      TRG_PUTNIK
PROJEKTOVANJE  Petar      TRG_PUTNIK
ISTRAZIVANJE
```

- SQL naredba sa eksplicitnim navodjenjem spoljnog spajanja bila bi slede}eg izgleda:

```
SELECT NAZIV, IME, POSAO
FROM ODELJENJE LEFT [OUTER] JOIN RADNIK
     ON ODELJENJE.ODELJENJE# = RADNIK.ODELJENJE#;
ili
SELECT NAZIV, IME, POSAO
FROM ODELJENJE LEFT [OUTER] JOIN RADNIK
     USING (ODELJENJE#);
```

A`uriranje baze podataka
(naredbe INSERT, UPDATE i DELETE)

- A`uriranje u {irem smislu zna~enja te re~i obuhvata dodavanje, izmenu sadr`aja i brisanje reda ili redova tabele. Te osnovne operacije realizuju se SQL naredbama INSERT, UPDATE i DELETE sa slede}im zna~enjem:

  - **INSERT-** dodavanje reda ili redova u tabelu
  - **UPDATE-** izmena sadr`aja postoje}eg reda ili redova tabele
  - **DELETE-** brisanje postoje}eg reda ili redova tabele

Dodavanje novih redova - INSERT

- Razmatra}e se slede}i tipovi insert naredbe:
  1. Ubacivanje vrednosti svih atributa n-torke;
  2. Ubacivanje vrednosti nekih atributa n-torke;
  3. Ubacivanje podataka iz jedne tabele u drugu.

1. U prvom slu~aju nije potrebno specificirati nazive atributa, pa insert naredba ima oblik:

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

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

- *Ubaciti podatke o Bojanu, pravniku, koji po~inje da radi u odeljenju 10 7.04.1984 sa platom od 450000 i bez prava na premiju. Neposredni rukovodilac jo{ nije poznat.*

```
INSERT INTO RADNIK VALUES
     (3545, 'Bojan', 'PRAVNIK', NULL,
     '1984-04-07', 450000,      NULL, 10)
```

- *Prika`imo rezultat insert naredbe*

```
SELECT *
FROM RADNIK
WHERE IME = 'Bojan';
```

```
SRADNIK IME   POSAO   SRUKOV DATZAP     PLATA PREMIJA ODELJENJE#
------- ----- ------- ------ ---------- ----- ------- ----------
   3545 Bojan PRAVNIK        1984-04-07 450000                10
```

2. Ako `elimo da unesemo vrednost za samo neke atribute, nazivi tih atributa moraju se eksplicitno navesti. U tom slu~aju naredba insert ima oblik:

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

Vrednosti za NOT NULL atribute moraju biti unete.

- *Uneti podatke o Srdjanu, planeru, koji je primljen u odeljenje 20.*

```
INSERT INTO RADNIK
 (IME, SRADNIK, ODELJENJE#, POSAO)
VALUES ('Srdjan', 3562, 20, 'PLANER')
```

- *Prika`imo rezultat insert naredbe*

```
SELECT *
FROM RADNIK
WHERE IME = 'Srdjan';

 SRADNIK IME    POSAO    SRUKOV DATZAP  PLATA  PREMIJA ODELJENJE#
------- ------ ------- ------- ------ ------ -------- ----------
   3562 Srdjan PLANER                                         20
```

3. Ukoliko obe tabele imaju isti broj atributa i ukoliko su atributi identi~no definisani, naredba INSERT je oblika:

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

ina~e:

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

- *Dati svim analiti~arima i savetnicima premiju u iznosu od 10% njihove plate. Te informacije uneti u tabelu premija zajedno sa datumom zaposlenja.*

```
INSERT INTO PREMIJA (SRADNIK, PREMIJA, POSAO, DATZAP)
SELECT SRADNIK, PLATA * .10, POSAO, DATZAP
FROM RADNIK
WHERE POSAO IN ('ANALITICAR', 'SAVETNIK');
```

- *Prikazati izgled tabele premija.*

```
SELECT * FROM PREMIJA;
```

SRADNIK	POSAO	PLATA	PREMIJA	DATZAP
3069	ANALITICAR		8000	1980-12-17
3488	SAVETNIK		30000	1981-11-09
3576	ANALITICAR		11000	1981-09-23
3600	ANALITICAR		9500	1981-12-03
3602	SAVETNIK		30000	1981-12-03
3634	ANALITICAR		13000	1982-01-23

Izmena sadr`aja - UPDATE

- Op{ti oblik naredbe je:

```
UPDATE tabela
SET atribut1=izraz1[,atribut2=izraz2]
[WHERE kriterijum selekcije n-torki],
```

odnosno:

```
UPDATE tabela
SET(atribut1, atribut2, ...)=(podupit)
[WHERE kriterijum selekcije n-torki].
```

- Podupit mora vratiti najvi{e po jednu vrednost za svaki od atributa u listi atributa iza SET klauzule.

- *Novoprimljeni pravnik Bojan ima}e predsednika za neposrednog rukovodioca. Realizovati tu odluku.*

```
UPDATE RADNIK
SET SRUKOV = 3539
WHERE IME = 'Bojan';
```

- *Srdjan po~inje sa radom 19.09.84 sa platom od 350000 i ima neposrednog rukovodioca sa {ifrom 3266. Realizovati te informacije.*

```
UPDATE RADNIK
SET SRUKOV = 3266,
    DATZAP = '1984-09-19',
    PLATA = 350000
WHERE IME = 'Srdjan';
```

- *Dati svim analiti~arima i savetnicima u odeljenju 20 15% povi{icu na platu .*

```
UPDATE RADNIK
SET PLATA = PLATA * 1.15
WHERE POSAO IN ('ANALITICAR', 'SAVETNIKK')
  AND ODELJENJE# = 20;
```

Brisanje n-torki tabele - DELETE

- Op{ti oblik naredbe je:

```
DELETE [FROM] tabela
[WHERE kriterijum selekcije n-torki].
```

- *Izbaciti podatke o Bojanu, jer je on napustio radnu organizaciju.*

```
DELETE FROM RADNIK
WHERE IME = 'Bojan';
```

- *Izbaciti podatke o svim radnicima odeljenja priprema .*

```
DELETE RADNIK
WHERE ODELJENJE# IN (SELECT ODELJENJE#
                     FROM ODELJENJE
                     WHERE NAZIV = 'PRIPREMA');
```

- Mogu}e je kontrolisati vremenski trenutak prenosa dejstava transakcije na bazu podataka pomo}u SQL naredbi COMMIT i ROLLBACK.
- Pod transakcijom se podrazumevaju sve operacije (INSERT, UPDATE, DELETE) od poslednjeg prenosa dejstava na bazu, tj. od poslednjeg izdavanja naredbe COMMIT.
- Naredbom COMMIT se, dakle, prenose dejstva transakcije na bazu podataka.
- Naredbom ROLLBACK poni{tavaju se sva dejstva od poslednjeg izdavanja naredbe COMMIT.

Pogled - VIEW

- [ta je pogled ?
  - Pogled je "prozor" kroz koji se vide podaci baze podataka,
  - Pogled je virtuelna tabela (sa njim se radi gotovo kao sa baznom tabelom, mada nema svoje podatke i ne zauzima nikakav memorijski prostor).
- Preciznije re~eno, pogled se koristi kao bilo koja druga tabela pri izve{tavanju.
- A`uriranje baze podataka preko pogleda ima brojna ograni~enja.
  - Ne mo`e se vr{iti a`uriranje preko pogleda ukoliko je pogled definisan nad vi{e tabela. Takvo a`uriranje bi zna~ilo da se jednom naredbom vr{i a`uriranje vi{e tabela baze podataka, {to je suprotno definiciji INSERT, DELETE i UPDATE naredbi.
  - Zatim se ne mo`e vr{iti a`uriranje preko pogleda ukoliko se u definiciji pogleda iza SELECT klauzule nalaze funkcije i aritmeti~ki izrazi.
  - Isto tako, a`uriranje se ne mo`e vr{iti ukoliko u definiciju pogleda nisu uklju~ene sve NOT NULL kolone tabele nad kojom je pogled definisan.

- Dakle, da bi mogli vr{iti a`uriranje preko pogleda:
  - pogled mora biti definisan nad jednom tabelom
  - u definiciju pogleda moraju biti uklju~ene sve NOT NULL kolone te tabele i
  - kolone pogleda moraju sadr`ati samo prost, neizmenjen sadr`aj kolona tabele nad kojom je pogled definisan.
- Za{to koristiti pogled ?
  - Jednostavnost kori{}enja - upro{}ava upite,
  - Tajnost - mo}an mehanizam kontrole pristupa podacima,
  - Performanse - ~uva se u kompajliranom obliku,
  - Nezavisnost podataka - menjaju se definicije pogleda, a ne aplikacioni programi koji koriste podatke baze podataka preko pogleda.

- Op{ti oblik naredbe za kreiranje pogleda je:

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

- *Kreirati pogled koji se sastoji od imena, {ifara i poslova zaposlenih u odeljenju 30.*

```
CREATE VIEW ODELJENJE30 AS
SELECT SRADNIK, IME, POSAO
FROM RADNIK
WHERE ODELJENJE# = 30;
```

- *Kreirati pogled koji }e davati informacije o platama i premijama radnika bez navodjenja imena radnika.*

```
CREATE VIEW ZARADA AS
SELECT SRADNIK, ODELJENJE#, PLATA, PREMIJA
FROM RADNIK;
```

- *Pogledajmo {ta se mo`e videti kroz taj pogled.*

```
SELECT *
FROM ZARADA;
```

SRADNIK	ODELJENJE#	PLATA	PREMIJA
3069	20	80000	
3199	30	160000	30000
3221	30	125000	50000
3266	20	297500	
3354	30	125000	140000
3398	30	285000	
3482	10	245000	
3488	20	300000	
3539	10	500000	
3544	30	150000	0
3576	20	110000	
3600	30	95000	
3602	20	300000	
3634	10	130000	
3545	10	450000	
3562	20		

- Pored originalnih vrednosti kolona baznih tabela pogled mo`e sadr`ati i izvedene vrednosti svojih virtuelnih kolona.
- *Kreirati pogled sa atributima odeljenje, posao, ukupna i srednja primanja radnika odredjenog zanimanja u odredjenom odeljenju.*

```
CREATE VIEW FINANSIJE
  (ODELJENJE#, POSAO, UKUPNAPLATA, SREDNJAPLATA) AS
SELECT ODELJENJE#, POSAO, SUM (PLATA), AVG (PLATA)
FROM RADNIK
GROUP BY ODELJENJE#, POSAO;
```

- *[ta se vidi kroz ovaj pogled ?*

```
SELECT * FROM FINANSIJE;
ODELJENJE# POSAO        UKUPNAPLATA SREDNJAPLATA
---------- ------------ ----------- ------------
        10 ANALITICAR        130000       130000
        10 PRAVNIK           450000       450000
        10 PREDSEDNIK        500000       500000
        10 RUKOVODILAC       245000       245000
        20 ANALITICAR        190000        95000
        20 PLANER
        20 RUKOVODILAC       297500       297500
        20 SAVETNIK          600000       300000
        30 ANALITICAR         95000        95000
        30 RUKOVODILAC       285000       285000
        30 TRG_PUTNIK        560000       140000
```

- Izbacivanje definicije pogleda (DROP VIEW)

```
DROP VIEW FINANSIJE;
```

- Kako pogled nema svoje podatke, ovom naredbom se izbacuje definicija pogleda iz re~nika podataka.

Ograni~enja

- SQL standard podr`ava slede}e vrste ograni~enja:
  - Ograni~enja domena
  - Ograni~enja tabela i kolona
  - Op{ta ograni~enja
- **Ograni~enje domena** je CHECK ograni~enje. Primenjuje se na sve kolone definisane nad posmatranim domenom.
- **Ograni~enje tabele** definisano je za jednu baznu tabelu. Ograni~enje tabele je:
  - UNIQUE ograni~enje,
  - PRIMARY KEY ograni~enje,
  - referencijalno (FOREIGN KEY) ograni~enje ili
  - CHECK ograni~enje.

- UNIQUE ograni~enje obezbedjuje jedinstvenost vrednosti jedne ili vi{e kolona bazne tabele. UNIQUE ograni~enje je zadovoljeno ako i samo ako ne postoje dva reda bazne tabele sa istim definisanim (NOT NULL) vrednostima u kolonama nad kojima je ograni~enje specificirano.
- PRIMARY KEY ograni~enje defini{e primarni klju~ tabele. Ono je zadovoljeno ako i samo ako ne postoje dva reda bazne tabele sa istim vrednostima u kolonama nad kojima je ograni~enje specificirano i ne postoji ni jedna nedefinisana vrednost ni u jednoj od navedenih kolona.
- Referencijalno ograni~enje realizuje referencijalni integritet na taj na~in {to specificira jednu ili vi{e kolona bazne tabele kao **referenciraju}e kolone** i njima odgovaraju}e **referencirane kolone** u nekoj (ne neophodno razli~itoj) baznoj tabeli.
  - Tabela sa referenciraju}im kolonama naziva se **referenciraju}a tabela**, a tabela sa referenciranim kolonama - **referencirana tabela**.
  - Nad referenciranim kolonama referencirane tabele definisano je PRIMARY KEY ili UNIQUE ograni~enje.

- Referencijalno ograni~enje je zadovoljeno ako su, za svaki red referenciraju}e tabele, vrednosti referenciraju}ih kolona jednake vrednostima odgovaraju}ih referenciranih kolona nekog reda referencirane tabele.
- Ako neka od referenciraju}ih kolona sadr`i null vrednost, zadovoljenje referencijalnog integriteta zavisi od tretmana null vrednosti (***match*** opcija).
- Kao deo specifikacije referencijalnog ograni~enja mogu se navesti i akcije za zadovoljavanje ograni~enja, kojima se defini{u promene koje treba sprovesti nad referenciraju}om tabelom, a bez kojih bi promene nad referenciranom tabelom dovele do naru{avanja referencijalnog ograni~enja.

– CHECK ograni~enje defini{e uslov. Ograni~enje je naru{eno ako je za bilo koji red tabele rezultat evaluacije uslova FALSE (la`no), a zadovoljeno ako je rezultat TRUE (istinito) ili UNKNOWN (nepoznato).

- **Op{te ograni~enje** (assertion) je CHECK ograni~enje, kojim se defini{e uslov nad podacima vi{e tabela baze podataka. Ograni~enje je naru{eno ukoliko je rezultat evaluacije uslova FALSE (la`no), a zadovoljeno ako je rezultat TRUE (istinito) ili UNKNOWN (nepoznato).
- SQL:1999 standard, kao i SQL-92, defini{e ograni~enje kolone kao sintaksnu skra}enicu ograni~enja tabele . Medjutim, pored UNIQUE, PRIMARY KEY, FOREIGN KEY i CHECK ograni~enja, ograni~enje kolone mo`e biti i NOT NULL.

UNIQUE ograni~enje

- Pretpostavimo da tabela RADNIK ima i kolonu MLB (mati~ni li~ni broj). UNIQUE ograni~enje obezbedi}e jedinstvenost vrednosti navedene kolone. Ako bi se UNIQUE ograni~enje specificiralo kao ograni~enje kolone, onda bi kolona MLB bila definisana na slede}i na~in:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) UNIQUE,
   ...
   ...);
```

- UNIQUE ograni~enje ne podrazumeva not null karakteristiku, odnosno dozvoljava da kolone nad kojima je definisano imaju null vrednosti.
- U SQL:1999 terminologiji UNIQUE ima zna~enje "nije jednako".

- Ako ho}emo da kolona MLB ima definisanu vrednost u svakom redu tabele RADNIK, neophodno je da se NOT NULL ograni~enje eksplicitno navede:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) NOT NULL UNIQUE,
   ...
 ...);
```

- Ako bi se UNIQUE ograni~enje specificiralo kao ograni~enje tabele, bilo bi zapisano na slede}i na~in:

```
CREATE TABLE RADNIK
  (...
   MLB CHAR (13) NOT NULL,
   ...
   UNIQUE (MLB),
 ... );
```

- UNIQUE ograni~enje nad vi{e kolona, kojim se obezbedjuje da ne postoje dva reda sa identi~nom kombinacijom vrednosti navedenih kolona, specificira se isklju~ivo kao ograni~enje tabele:

```
UNIQUE (<naziv kolone1>,
        < naziv kolone2>,
        ...,
        < naziv kolonen>).
```

- Ako se zahteva da svi redovi tabele budu razli~iti, odnosno da budu razli~ite kombinacije vrednosti svih kolona tabele, UNIQUE ograni~enje ima slede}i skra}eni oblik:

```
UNIQUE (VALUE).
```

PRIMARY KEY ograni~enje

- Primarni klju~evi tabela RADNIK i ODELJENJE mogu se specificirati kao ograni~enja kolona SRADNIK i ODELJENJE#:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER PRIMARY KEY,
   ...
   ...);

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER PRIMARY KEY,
   ...
   ...);
```

- SQL-89 standard zahtevao je da se specificira NOT NULL PRIMARY KEY.
- SQL:1999 standard, kao i SQL-92, dozvoljava da se, sa istom semantikom, specificira samo PRIMARY KEY. To zna~i da je u PRIMARY KEY ograni~enje uklju~eno i NOT NULL ograni~enje.

- Ako bi se prethodna PRIMARY KEY ograni~enja specificirala kao ograni~enja tabela, bila bi zapisana na slede}i na~in:

```
CREATE TABLE RADNIK
  (SRADNIK INTEGER NOT NULL,
   ...
   CONSTRAINT RADNIK_PK PRIMARY KEY (SRADNIK),
   ... );

CREATE TABLE ODELJENJE
  (ODELJENJE# INTEGER NOT NULL,
   ...
   CONSTRAINT ODELJENJE_PK PRIMARY KEY (ODELJENJE#),
  ... );
```

- Svako ograni~enje ima naziv
  - korisnik mo`e eksplicitno da imenuje svako ograni~enje
  - ukoliko korisnik to ne uradi, sistem sam dodeljuje naziv svakom ograni~enju.
- Kada je primarni klju~ slo`en, odnosno sastavljen od vi{e kolona, specificira se isklju~ivo kao ograni~enje tabele:

```
[CONSTRAINT <naziv ogranicenja>]
PRIMARY KEY (<naziv kolone1>, ..., < naziv kolonen>).
```

Referencijalno (FOREIGN KEY) ograni~enje

- Referenciraju}a tabela u na{em primeru je tabela RADNIK sa referenciraju}om kolonom RADNIK.ODELJENJE#, a referencirana tabela je tabela ODELJENJE sa referenciranom kolonom ODELJENJE.ODELJENJE#.
- Referenciraju}a kolona RADNIK.ODELJENJE# je spoljni klju~ tabele RADNIK, a referencirana kolona ODELJENJE.ODELJENJE# je primarni klju~ tabele ODELJENJE.
- Referencijalno ograni~enje specificira se kao deo definicije tabele RADNIK. Ono se zadaje ili kao ograni~enje kolone, kroz definiciju njene kolone ODELJENJE#, ili kao ograni~enje tabele.
- Osnovna sintaksa specifikacije referencijalnog ograni~enja kao ograni~enja kolone mo`e se ilustrovati slede}im primerom:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE (ODELJENJE#),
   ... );
```

- U prikazanom primeru ograni~enja nisu eksplicitno imenovana. Eksplicitnim imenovanjem referencijalnog i NOT NULL ograni~enja imali bi slede}i izgled naredbe:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER
     CONSTRAINT ODELJENJE_NOT_NULL NOT NULL
     CONSTRAINT ODELJENJE_FK
       REFERENCES ODELJENJE (ODELJENJE#),
   ... );
```

- Navedeno referencijalno ograni~enje mo`emo da zapi{emo i kao ograni~enje tabele:

```
CREATE TABLE RADNIK
  (...
   ...
   ...
   CONSTRAINT ODELJENJE_FK FOREIGN KEY (ODELJENJE#)
       REFERENCES ODELJENJE (ODELJENJE#),
  ... );
```

- Kada je referencirana kolona primarni klju~ referencirane tabele, ne mora se eksplicitno navesti u specifikaciji referencijalnog ograni~enja:

```
CREATE TABLE RADNIK
  (...
   ...
   ...
   CONSTRAINT ODELJENJE_FK FOREIGN KEY (ODELJENJE#)
       REFERENCES ODELJENJE,
   ... );
```

- Kada je spoljni klju~ slo`en, odnosno sastavljen od vi{e kolona, specificira se isklju~ivo kao ograni~enje tabele

- Svi do sada navedeni primeri spre~avali su izvr{avanje svake SQL naredbe koja bi naru{ila definisano referencijalno ograni~enje.
- To je bio i jedini na~in zadovoljavanja referencijalnog integriteta u SQL-89 standardu.
- SQL:1999 standard podr`ava pet referencijalnih akcija kao odgovor na operacije nad referenciranom tabelom koje potencijalno naru{avaju referencijalni integritet.
- Operacije koje potencijalno naru{avaju referencijalni integritet su:
  - brisanje (DELETE) ili
  - izmena (UPDATE) redova referencirane tabele.
- Referencijalne akcije su
  - NO ACTION,
  - RESTRICT,
  - CASCADE,
  - SET NULL i
  - SET DEFAULT.
- Pri specifikaciji referencijalnog ograni~enja mogu}e je izabrati jednu referencijalnu akciju za operaciju DELETE, a drugu za operaciju UPDATE.

- NO ACTION referencijalna akcija poni{tava efekte SQL naredbe ako je **na kraju** njenog izvr{avanja naru{eno referencijalno ograni~enje.
- SUBP prikazuje poruku o gre{ci.
- U slede}em primeru specificirano je referencijalno ograni~enje sa NO ACTION referencijalnom akcijom i za brisanje i za izmenu redova referencirane tabele:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE
         ON DELETE NO ACTION
         ON UPDATE NO ACTION,
   ... );
```

- NO ACTION je default referencijalna akcija, {to zna~i da se primenjuje ukoliko nijedna akcija nije eksplicitno navedena.

- RESTRICT i NO ACTION su veoma sli~ne referencijalne akcije. Razlika je u tome {to NO ACTION dozvoljava da referencijalno ograni~enje bude naru{eno **u toku** izvr{avanja SQL naredbe, a RESTRICT ne.
- CASCADE referencijalna akcija zadovoljava referencijalno ograni~enje tako {to nastavlja zapo~etu operaciju (DELETE ili UPDATE) nad odgovaraju}im redovima referenciraju}e tabele.
- U slede}em primeru specificirano je referencijalno ograni~enje koje obezbedjuje da brisanje reda tabele ODELJENJE ima za posledicu brisanje svih redova tabele RADNIK koji su referencirali taj red tabele ODELJENJE:

```
CREATE TABLE RADNIK
  (...
   ...
   ODELJENJE# INTEGER NOT NULL
       REFERENCES ODELJENJE
         ON DELETE CASCADE,
   ... );
```

- SET NULL referencijalna akcija zadovoljava referencijalno ograni~enje tako {to postavlja vrednosti referenciraju}ih kolona na NULL, a SET DEFAULT na default vrednosti. Naravno, SET NULL akcija je mogu}a samo ukoliko referenciraju}e kolone nisu definisane kao NOT NULL.

CHECK ograni~enje

- CHECK ograni~enje je najfleksibilnije i verovatno najkorisnije ograni~enje.
- Njime se defini{e uslov, koji mo`e ograni~avati dozvoljene vrednosti domena ili kolone, odrediti medjuzavisnosti vrednosti kolona i redova jedne tabele ili definisati op{te ograni~enje, koje se prostire na vi{e tabela baze podataka.
- Slede}e CHECK ograni~enje je specificirano kao ograni~enje kolone IME i obezbedjuje da sva imena budu zapisana velikim slovima u bazi podataka:

```
CREATE TABLE RADNIK
  (...
   IME VARCHAR (20) NOT NULL CHECK(IME = UPPER (IME)),
   ...
   ...);
```

- Slede}e ograni~enje specificirano je kao ograni~enje tabele RADNIK. Ono ne dozvoljava da ukupan broj radnika bude ve}i od 100:

```
CREATE TABLE RADNIK
  (...
   ...
   CONSTRAINT BROJ_RADNIKA
     CHECK((SELECT COUNT(*)
            FROM RADNIK) <= 100),
   ...);
```

- Op{te ograni~enje specificira se slede}om sintaksom:

```
CREATE ASSERTION <naziv ogranicenja>
CHECK ( <uslov> ).
```

Provera ograni~enja

- Ograni~enja se, po default-u, proveravaju na kraju izvr{avanja svake SQL naredbe.
- SQL:1999 standard dozvoljava da se provera ograni~enja odlo`i i izvr{i na kraju transakcije.
- Za svako ograni~enje je mogu}e specificirati da li je ili ne dozvoljeno odlaganje provere do kraja transakcije, {to se zapisuje navodjenjem opcije [NOT] DEFERRABLE.
- Ako je izabrana opcija DEFERRABLE, tada je neophodno navesti da li je na po~etku transakcije provera ograni~enja odlo`ena (INITIALLY DEFERRED) ili ne (INITIALLY IMMEDIATE).
- Provera ograni~enja, koje je specificirano kao INITIALLY IMMEDIATE DEFERRABLE, mo`e se odlo`iti do kraja transakcije kori{}enjem naredbe SET CONSTRAINTS.