Elektrotehnički fakultet
Univerzitet u Beogradu
Katedra za računarsku tehniku i informatiku

Baze podataka 1

SQL

Autori:
Miloš Cvetanović
Stefan Tubić
Filip Hadžić
Tamara Šekularac

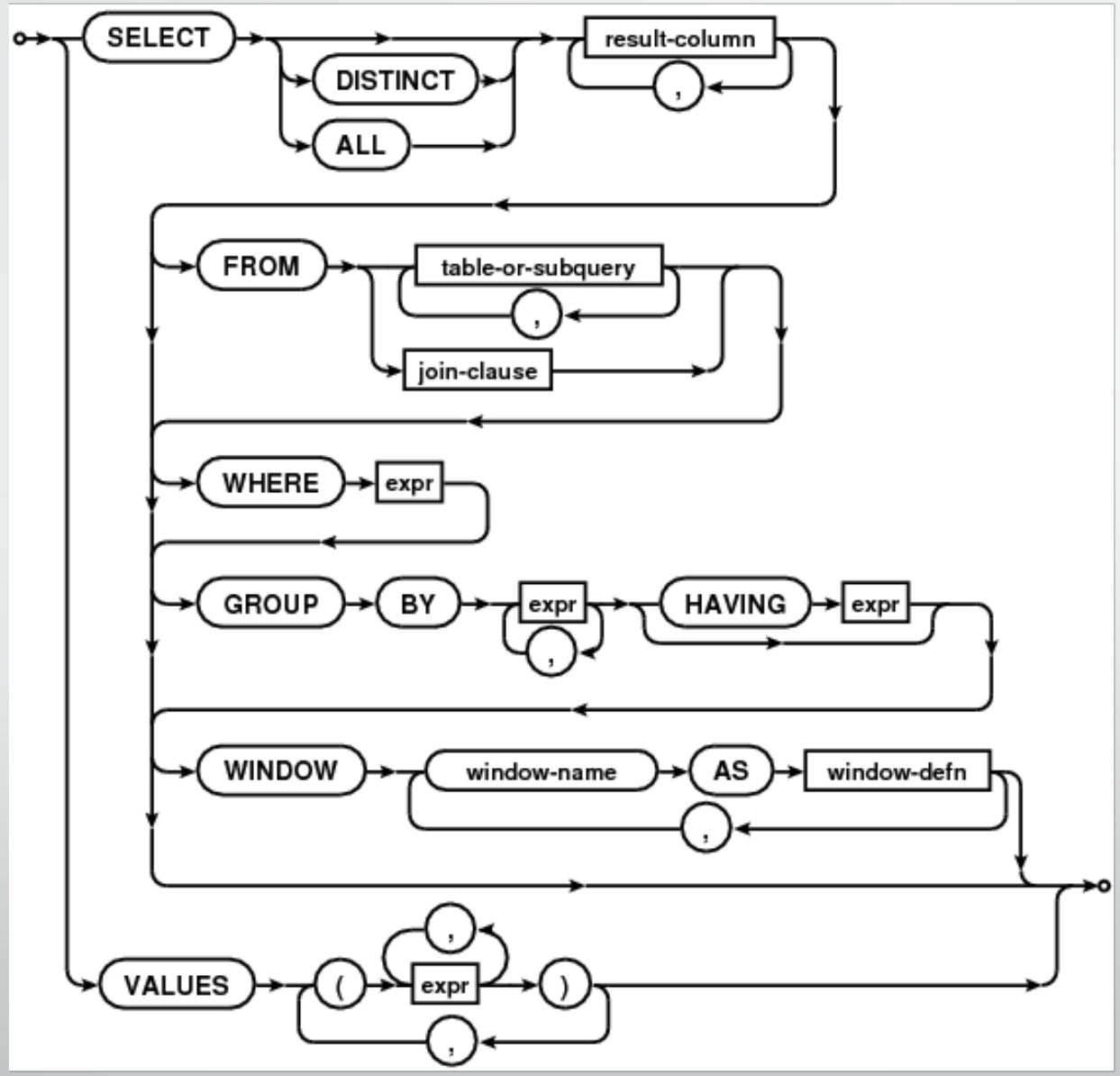
2018/2019

SQL (Structured Query Language)

- SQL je struktuirani upitni jezik koji omogućava pristup podacima u sistemima za upravljanje relacionim bazama podataka.
- SQL možemo podeliti na četiri dela:
  1. DDL (Data Definition Language):
     CREATE, DROP, ALTER, RENAME,...
  2. DML (Data Manipulation Language):
     SELECT, INSERT, UPDATE, DELETE
  3. DCL (Data Control Language)
     GRANT, REVOKE
  4. TCL (Transaction Control Language)
     COMMIT, ROLLBACK,...
- Svaka kompanija definiše svoj SQL za potrebe svoje baze:

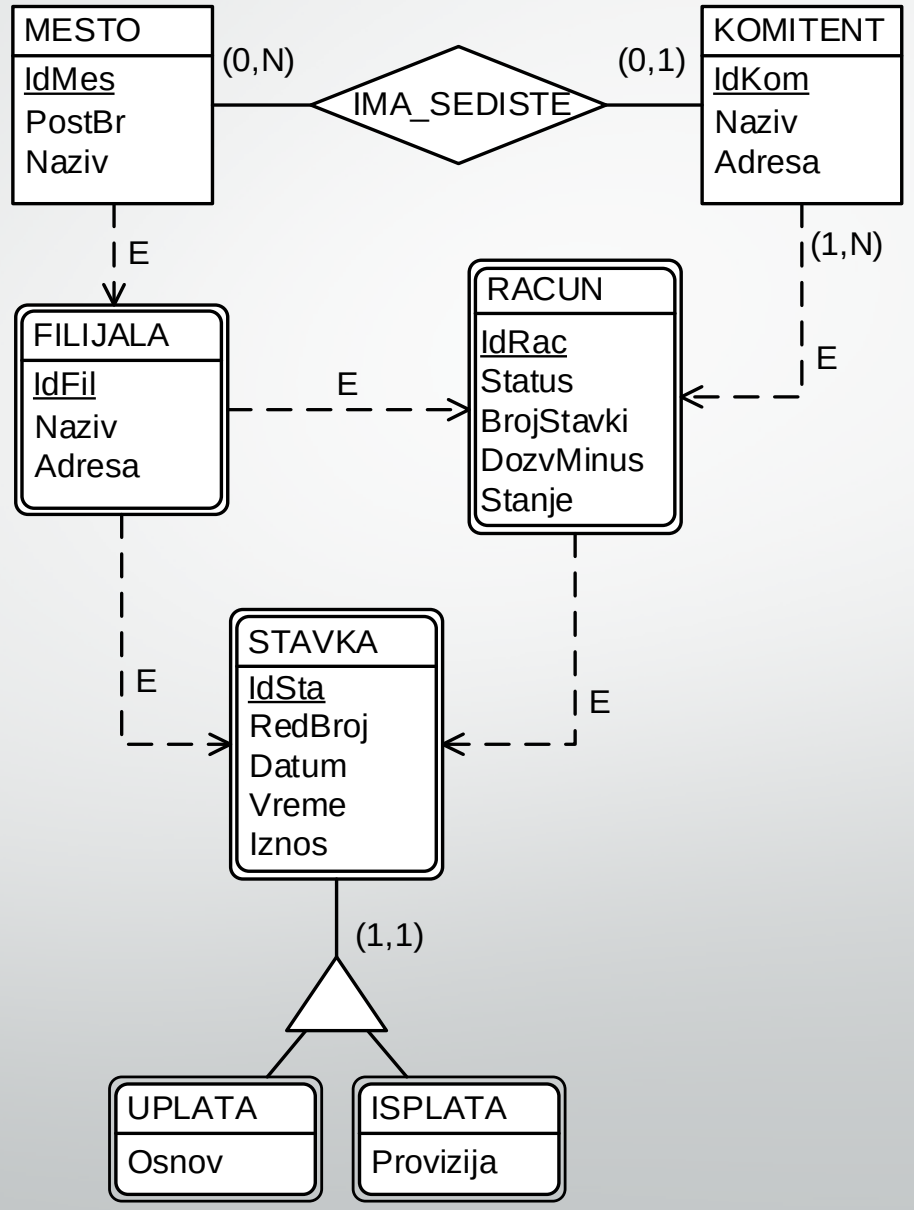
  SQLite, MSSQL, MySql, PostgreSQL, Oracle, ...

SELECT upit



Preuzeto: https://www.sqlite.org

Model baze nad kojom se rade upiti



Opis baze nad kojom se rade upiti

Banka putem svojih filijala (prati se naziv i adresa) u raznim mestima (prate se poštanski broj i naziv) opslužuje svoje komitente (prati se naziv i adresa) koji mogu biti bez mesta, a u trenutku prvog pojavljivanja u banci prijavljuju sedište u određenom mestu.

Svaki komitent može da ima više računa u svakoj od filijala (prate se status, broj stavki, dozvoljeni minus, i stanje), a mora imati bar jedan račun. Status računa može biti aktivan, blokiran ili ugašen. Račun postaje blokiran kada pređe u nedozvoljeni minus, a aktivira se kada stanje pređe u dozvoljeni minus.

Komitenti sa svojih računa vrše transakcije putem stavki prometa (prate se redni broj, datum i vreme) koje mogu biti uplate (prati se osnov i iznos) ili isplate (prati se iznos i provizija), pri čemu je to moguće u bilo kojoj filijali.

Primeri (prikaz cele tabele)

1. Napisati SQL upit koji ispisuje sve podatke iz tabele Komitent.

```
SELECT *
FROM Komitent
```

2. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun.

```
SELECT *
FROM Racun
```

Primeri (prikaz određenih kolona)

3. Napisati SQL upit koji ispisuje nazive svih komitenata.

```
SELECT Naziv
FROM Komitent
```

4. Napisati SQL upit koji ispisuje naziv i adresu svakog komitenta.

```
SELECT Naziv, Adresa
FROM Komitent
```

Primeri (sortiranje)

5. Napisati SQL upit koji ispisuje sve podatke iz tabele Komitent sortirane po nazivu u rastućem poretku.

```
SELECT *
FROM Komitent
ORDER BY Naziv
```

6. Napisati SQL upit koji ispisuje sve podatke iz tabele Komitent sortirano po nazivu u opadajućem poretku.

```
SELECT *
FROM Komitent
ORDER BY Naziv DESC
```

ASC sortira vrednosti u rastućem poretku i ona je podrazumevana.
DESC sortira vrednosti u opadajućem poretku

Primeri (sortiranje)

7. Napisati SQL upit koji ispisuje sve podatke iz tabele Komitent sortirane po nazivu u rastućem poretku, a zatim po adresi isto u rastućem poretku.

```
SELECT *
FROM Komitent
ORDER BY Naziv, Adresa
```

8. Napisati SQL upit koji ispisuje ispisuje naziv i adresu svakog komitenta sortirano po nazivu u rastućem poretku, a zatim po adresi u opadajućem poretku.

```
SELECT Naziv, Adresa
FROM Komitent
ORDER BY Naziv, Adresa DESC
```

Primeri (WHERE)

9. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun, za račune koji na stanju imaju tačno -55000 dinara.

```
SELECT *
FROM Racun
WHERE Stanje = -55000
```

10. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun, za račune kod kojih je stanje pozitivno.

```
SELECT *
FROM Racun
WHERE Stanje>0
```

Moguće operacije poređenja:
<, <=, >, >=, =, !=

Primeri (WHERE)

11. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun samo za blokirane račune.

```
SELECT *
FROM Racun
WHERE Status = 'B'
```

12. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun, za blokirane račune koji imaju stanje manje od -50000 dinara.

```
SELECT *
FROM Racun
WHERE Status = 'B' AND Stanje < -50000
```

Primeri (WHERE)

13. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun, za blokirane račune ili račune koji imaju stanje manje od -50000 dinara.

```
SELECT *
FROM Racun
WHERE Status = 'B' OR Stanje < -50000
```

14. Napisati SQL upit koji ispisuje stanje računa, za blokirane račune koji imaju stanje manje od -50000 dinara.

```
SELECT Stanje
FROM Racun
WHERE Status = 'B' AND Stanje<-50000
```

Primeri (WHERE)

15. Napisati SQL upit koji ispisuje sve podatke iz tabele Racun, za račune koji imaju stanje izmedju -12000 i 10000 dinara.

I varijanta:

```
SELECT *
FROM Racun
WHERE Stanje>=-12000 AND Stanje<=10000
```

II varijanta:

```
SELECT *
FROM Racun
WHERE Stanje BETWEEN -12000 AND 10000
```

Between obuhvata obe granične vrednosti.

Primeri (prikaz izračunatih kolona)

16. Napisati SQL upit koji ispisuje stanje računa, kamatnu stopu i vrednost kamate za sve račune koji su u minusu. Kamatna stopa ima vrednost od 3%.

```
SELECT Stanje, 3, Stanje*-0.03
FROM Racun
WHERE Stanje<0
```

17. Napisati SQL upit koja za svaki račun ispisuje sve podatke i dodatno informaciju o tome da li će biti izvan dozvoljenog minusa kada mu se obračuna kamata za račune koji su trenutno u minusu.

```
SELECT *, Stanje*1.03<-DozvMinus
FROM Racun
WHERE Stanje<0
```

Nazivi kolona u tabeli su uneti izrazi.

Primeri (preimenovanje kolona)

18. Napisati SQL upit koji ispisuje stanje računa, kamatnu stopu i vrednost kamate za sve račune koji su u minusu. Kamatna stopa ima vrednost od 3%. Potrebno je da kolone imaju sledeće nazive: Stanje, Kamatna stopa i Kamata.

```
SELECT Stanje , 3 AS "Kamatna stopa", Stanje*-0.03 AS Kamata
FROM Racun
WHERE Stanje<0
```

String za preimenovanje kolone se piše u duplim navodnicima.
U slučaju jedne reči navodnici mogu da se izostave.

Primeri (DISTINCT)

19. Napisati SQL upit koja za račune koji nisu u nedozvoljenom minusu ispisuje koliko mogu još maksimalno para da podignu.

```
SELECT IdRac, Stanje +dozvMinus
FROM Racun
WHERE Stanje > -dozvMinus
```

20. Napisati SQL upit koji ispisuje sve različite nazive komitenata (nazivi bez ponavljanja).

```
SELECT DISTINCT Naziv
FROM Komitent
```

Primeri (NULL vrednosti)

21. Napisati SQL upit koji ispisuje račune koji su ugašeni (ugašeni računi imaju Stanje NULL).

```
SELECT *
FROM Racun
WHERE Stanje IS NULL
```

22. Napisati SQL upit koji ispisuje račune koji nisu ugašeni (ugašeni računi imaju Stanje NULL).

```
SELECT *
FROM Racun
WHERE Stanje IS NOT NULL
```

Upoređivanje NULL vrednosti sa <, <=, >=, >, =, != uvek vraća false.

Primeri (agregatne funkcije)

23. Napisati SQL upit koji ispisuje sumu stanja na svim računima.

```
SELECT SUM(Stanje)
FROM Racun
```

24. Napisati SQL upit koji ispisuje minimalno stanja na računima koji su u plusu.

```
SELECT MIN(Stanje) AS Minimum
FROM Racun
WHERE Stanje>0
```

Agregatne funkcije: COUNT, MIN, MAX, SUM, AVG

Primeri (agregatne funkcije)

25. Napisati SQL upit koji ispisuje prosečno stanje na računima.

```
SELECT AVG(Stanje), SUM(Stanje)/COUNT(Stanje),
    SUM(Stanje)/COUNT(*)
FROM Racun
```

AVG računa prosečnu vrednost samo onih kolona koje nisu NULL
COUNT broji samo one kolone koje nisu NULL

Primeri (spajanje dve tabele)

26. Napisati SQL upit koji za svaki račun ispisuje naziv komitenta čiji je račun i stanje na računu.

```
SELECT Naziv, Stanje
FROM Komitent, Racun
WHERE Komitent.IdKom = Racun.IdKom
ili
SELECT Naziv, Stanje
FROM Komitent K, Racun R
WHERE K.IdKom = R.IdKom
```

Primeri (spajanje više tabela)

27. Napisati SQL upit koji za svaku stavku računa ispisuje u kojoj filijali je bila uplata/isplata, njen iznos i id racuna.

```
SELECT F.Naziv AS "Naziv filijale", S.Iznos, R.IdRac
FROM Filijala F, Stavka S, Racun R
WHERE F.idFIl=S.IdFIl AND S.IdRac=R.IdRac
```

28. Napisati SQL upit koji za svaku stavku računa ispisuje u kojoj Filijali je bila uplata/isplata, njen iznos i id racuna, kao i naziv filijale u kojoj je otvoren račun.

```
SELECT FSt.Naziv AS "Mesto uplate/isplate", S.Iznos, R.IdRac,
    FRac.Naziv AS "Matična filijala"
FROM Filijala FSt, Stavka S, Racun R, Filijala FRac
WHERE FSt.idFIl=S.IdFIl AND S.IdRac=R.IdRac AND FRac.IdFil=R.IdFil
```

Primeri (spajanje više tabela)

29. Napisati SQL upit koji za svaku platu ispisuje u kojoj Filijali je bila uplata.

```
SELECT F.Naziv AS "Naziv filijale", S.Iznos, R.IdRac
FROM Filijala F, Stavka S, Racun R, Uplata U
WHERE F.idFIl=S.IdFIl AND S.IdRac=R.IdRac
    AND S.idSta=U.idSta AND U.Osnov='Plata'
```

Primeri (GROUP BY)

30. Napisati SQL upit koji za svaki id komitenta ispisuje ukupno stanje na njihovim računima.

```
SELECT idKom, SUM(Stanje) AS "Suma na racunima"
FROM Racun R
GROUP BY idKom
```

Primeri (GROUP BY)

31. Napisati SQL upit koji za račun ispisuje koliko je bilo uplata na račun i koliko je iznosila njihova suma.

```
SELECT  R.IdRac, COUNT(*) as "Broj uplata",
    SUM(S.Iznos) AS "Suma uplata"
FROM Racun R, Stavka S, Uplata U
WHERE S.IdRac=R.IdRac AND S.idSta=U.idSta
GROUP BY R.idRac
```

Primeri (GROUP BY)

32. Napisati SQL upit koji za svakog komitenta ispisuje ukupno stanje na njihovim računima.

```
SELECT K.Naziv, SUM(Stanje) AS "Suma na racunima"
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom
GROUP BY K.idKom, K.Naziv
```

Primeri (GROUP BY)

33. Napisati SQL upit koji za svakog komitenta koji ima bar jedan aktivan račun ispisuje broj aktivnih računa koje on ima.

```
SELECT K.Naziv, COUNT(*) AS "Broj racuna"
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom AND R.Status='A'
GROUP BY K.idKom, K.Naziv
```

Primeri (GROUP BY)

34. Napisati SQL upit koji ispisuje broj računa na kojima je stanje pozitivno, samo za korisnike koji imaju bar jedan pozitivan račun.

```
SELECT K.Naziv, COUNT(*) AS "Suma na racunima"
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom AND Stanje >= 0
GROUP BY K.idKom, K.Naziv
```

Primeri (GROUP BY, HAVING)

35. Napisati SQL upit koji za komitente koji imaju ukupno pozitivno stanje na računima ispisuje naziv komitenta i sumu na tim računima.

```
SELECT K.Naziv, SUM(Stanje) AS "Suma na racunima"
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom
GROUP BY K.idKom, K.Naziv
HAVING SUM(Stanje)>=0
```

Primeri (GROUP BY, HAVING)

36. Napisati SQL upit koji ispisuje sve komitente koji imaju tačno dva računa.

```
SELECT K.Naziv
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom
GROUP BY K.idKom, K.Naziv
HAVING COUNT(*)=2
```

Primeri (GROUP BY, HAVING)

37. Napisati SQL upit koji ispisuje sve komitente koji imaju tačno dva aktivna računa.

```
SELECT K.Naziv
FROM Komitent K, Racun R
WHERE K.idKom = R.idKom AND R.Status = 'A'
GROUP BY K.idKom, K.Naziv
HAVING COUNT(*) = 2
```

Primeri (GROUP BY, HAVING)

38. Napisati SQL upit koji ispisuje sve račune koji su imali transakcije u dve ili više različitih filijala.

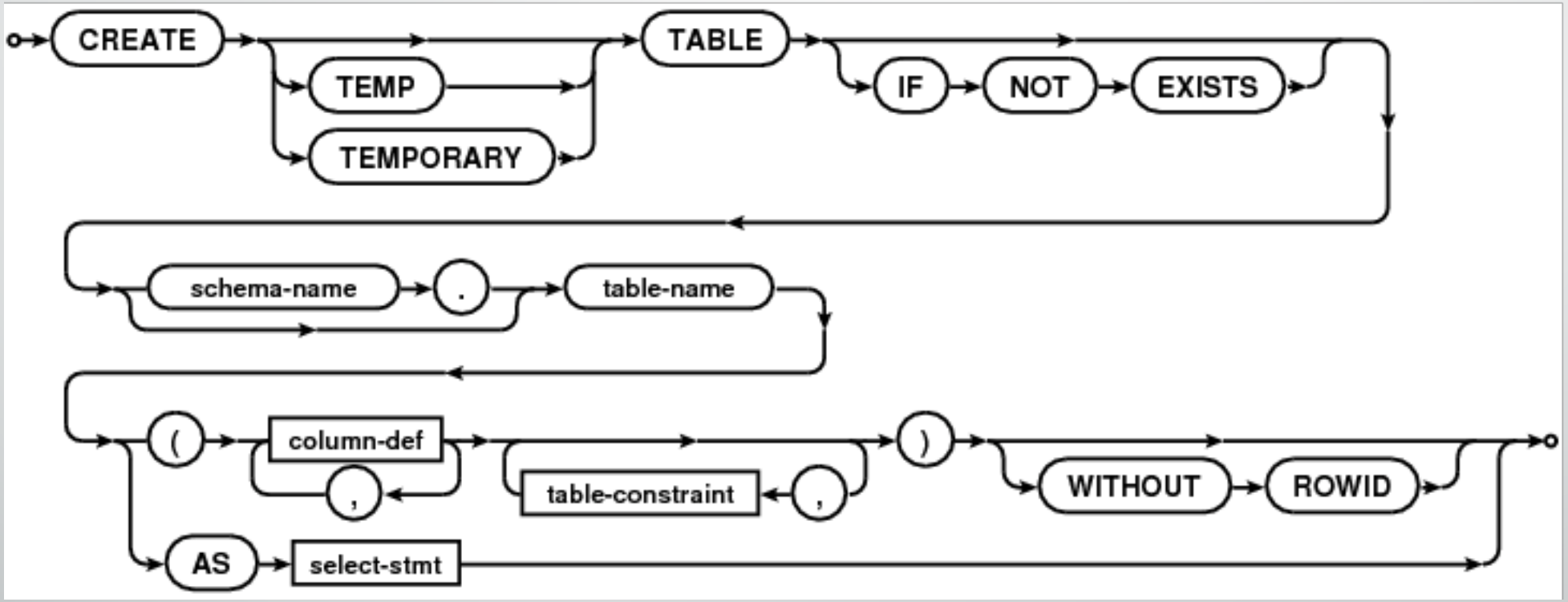
```
SELECT R.IdRac
FROM Racun R, Stavka S
WHERE R.IdRac = S.IdRac
GROUP BY R.IdRac
HAVING COUNT(DISTINCT(S.IdFil)) >= 2
```

Primeri (GROUP BY, HAVING)

39. Napisati SQL upit koji ispisuje sve račune koji su imali transakcije u dve ili više različitih mesta.

```
SELECT R.IdRac
FROM Racun R, Stavka S, Filijala F
WHERE R.IdRac = S.IdRac AND S.IdFil = F.IdFil
GROUP BY R.IdRac
HAVING COUNT(DISTINCT(F.IdMes)) >= 2
```

CREATE naredba



Preuzeto: https://www.sqlite.org

Primeri (CREATE)

40. Napisati SQL skriptu za kreiranje tabele **Komitent**. *IdKom* je celobrojna veličina koja identifikuje komitenta, *Naziv* predstavlja niz od najviše 50 karaktera i obavezan je, *Adresa* predstavlja niz od najviše 50 karaktera.

```
CREATE TABLE Komitent (
    IdKom INTEGER PRIMARY KEY,
    Naziv VARCHAR(50) NOT NULL,
    Adresa VARCHAR(50))
```

ili

```
CREATE TABLE Komitent (
    IdKom INTEGER,
    Naziv VARCHAR(50) NOT NULL,
    Adresa VARCHAR(50),
    PRIMARY KEY(IdKom))
```

Primeri (CREATE)

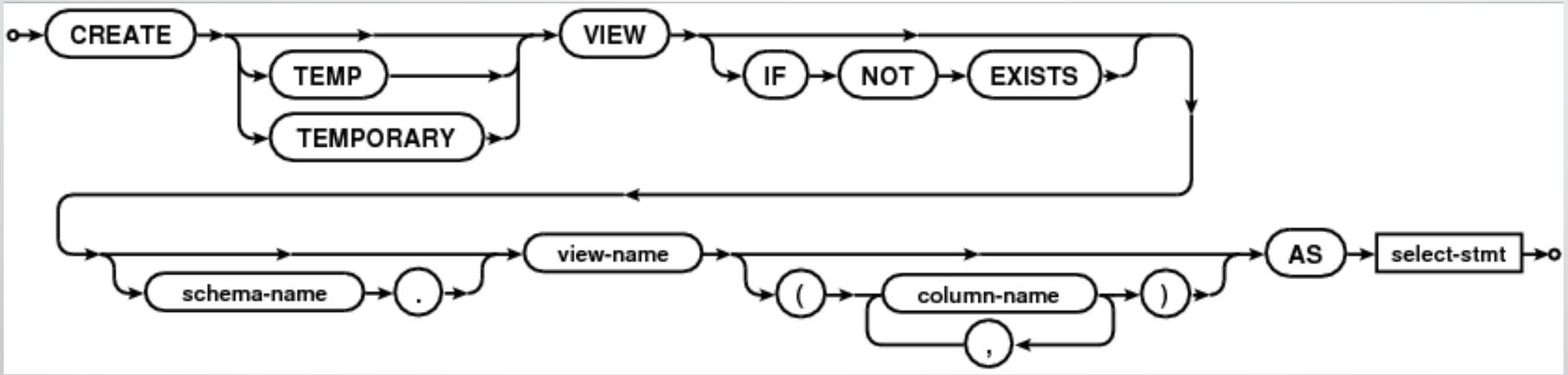
41. Napisati SQL skriptu za kreiranje tabele **Mesto**. *IdMes* je celobrojna veličina koja identifikuje mesto, *PostBr* predstavlja niz od 6 karaktera koji je jedinstven i obavezan, *Naziv* predstavlja niz od najviše 50 karaktera i obavezan je.

```
CREATE TABLE Mesto (
    IdMes INTEGER PRIMARY KEY,
    PostBr CHAR(6) NOT NULL UNIQUE,
    Naziv VARCHAR(50) NOT NULL)
```

ili

```
CREATE TABLE Mesto (
    IdMes INTEGER,
    PostBr CHAR(6) NOT NULL UNIQUE,
    Naziv VARCHAR(50) NOT NULL
    PRIMARY KEY(IdMes))
```

VIEW naredba



Preuzeto: https://www.sqlite.org

Primeri (VIEW)

42. Napisati SQL skriptu za kreiranje pogleda BlokiraniKomitent koji kao prikaz daje sve Komitente koji imaju bar jedan blokiran račun. Prikaz rezultata treba da bude u formatu: IdKom, Naziv, BrojBlokiranihRacuna.

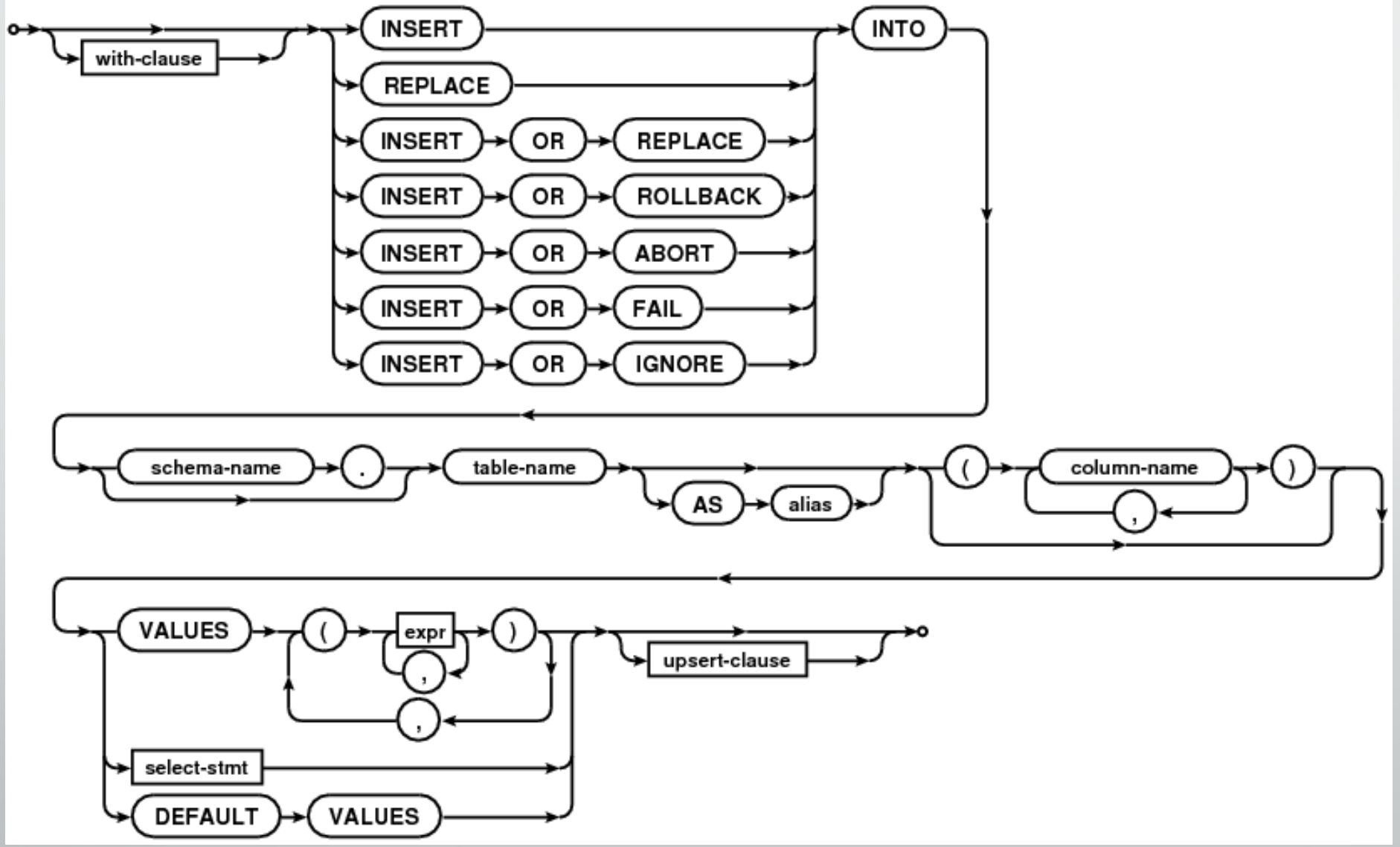
```
CREATE VIEW BlokiraniKomitent
    ( IdKom, Naziv, BrojBlokiranihRacuna )
AS SELECT K.IdKom, K.Naziv, COUNT(*)
    FROM Komitent K, Racun R
    WHERE K.IdKom =R.IdKom AND R.Status = 'B'
    GROUP BY K.IdKom, K.Naziv
```

Primeri (VIEW)

43. Napisati SQL skriptu za kreiranje pogleda FilijaleUMestu koji kao prikaz daje sve Gradove u kojima ima Filijala i njihov broj. Prikaz rezultata treba da bude u formatu: IdMes, Naziv, BrojFilijala. Iskoristiti pogled za prikaz samo mesta koje imaju bar 2 filijale.

```
CREATE VIEW FilijaleUMestu
    ( IdMes, Naziv, BrojFilijala )
AS SELECT M.IdMes, M.Naziv, COUNT(*)
    FROM Mesto M, Filijala F
    WHERE M.IdMes =F.IdMes
    GROUP BY M.IdMes, M.Naziv


SELECT IdMes, Naziv
FROM FilijaleUMestu
WHERE BrojFilijala >= 2
```

INSERT naredba



Preuzeto: https://www.sqlite.org

Primeri (INSERT)

44. Napisati SQL skriptu za dodavanje novog Komitenta sa idKom 10, koji se zove Nikola i živi u Bulevaru Kralja Milana 17.

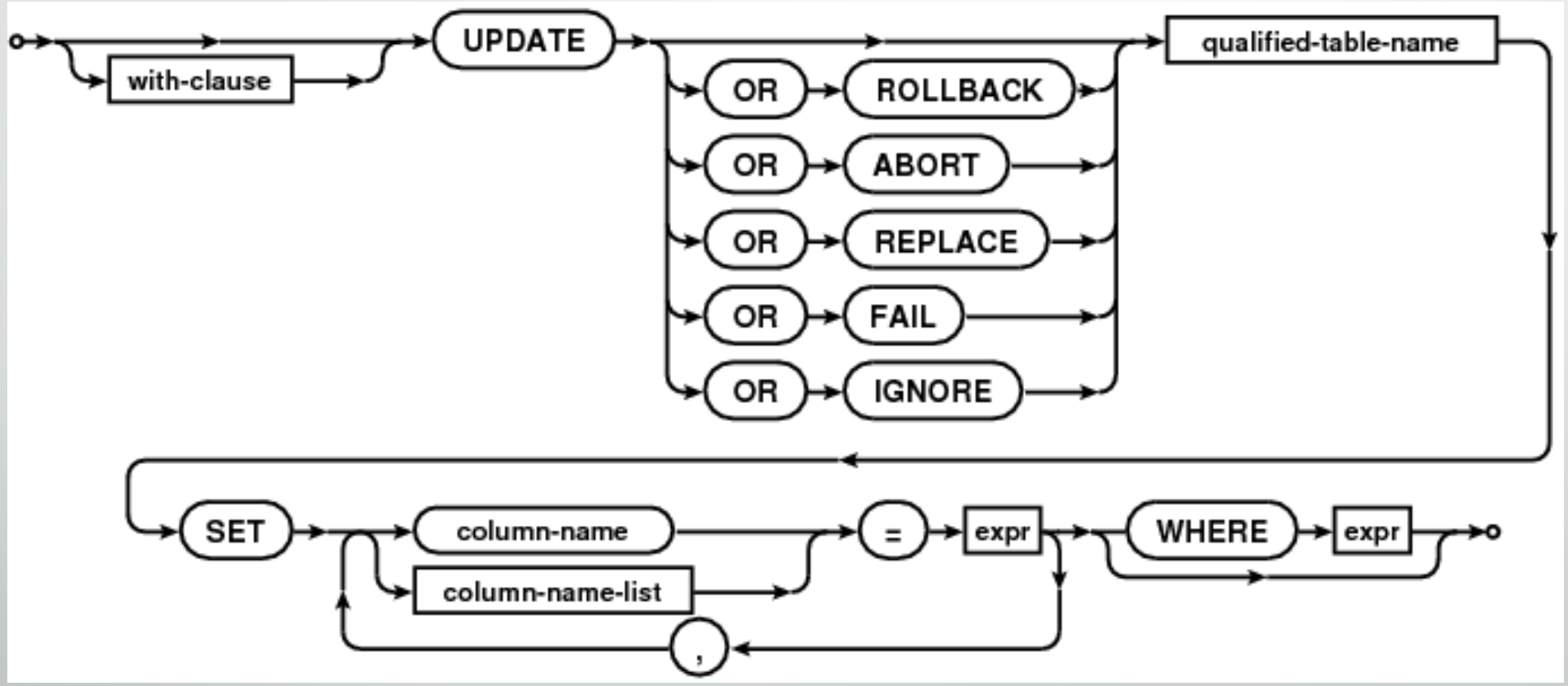
```
INSERT INTO Komitent
    (IdKom, Naziv, Adresa)
VALUES (10, 'Nikola', 'Bulevar kralja Milana 17')
```

Primeri (INSERT)

45. Nikoli su potrebna dva računa. Napisati SQL skriptu za kreiranje dva računa. Prvi je potrebno da ima id 8, dozvoljeni minus od 100000 dinara, da na stanju ima 0 dinara i da bude aktivan. Drugi je potrebno da ima id 9,da nema dozvoljeni minus, da na stanju ima 0 dinara i da bude aktivan. Oba su otvorena u Filijali sa idFil 2.

```
INSERT INTO Racun
    (IdRac, IdKom, dozvMinus, Stanje, Status, BrojStavki, IdFil)
VALUES
    (8, 10, 100000, 0, 'A', 0, 2),
    (9, 10, 0, 0, 'A', 0, 2)
```

UPDATE naredba



Preuzeto: https://www.sqlite.org

Primeri (UPDATE)

46. Nikola se preselio. Napisati SQL skriptu za Komitenta sa idKom 10 menja adresu stanovanja. Nova adresa je Pozeska 23.

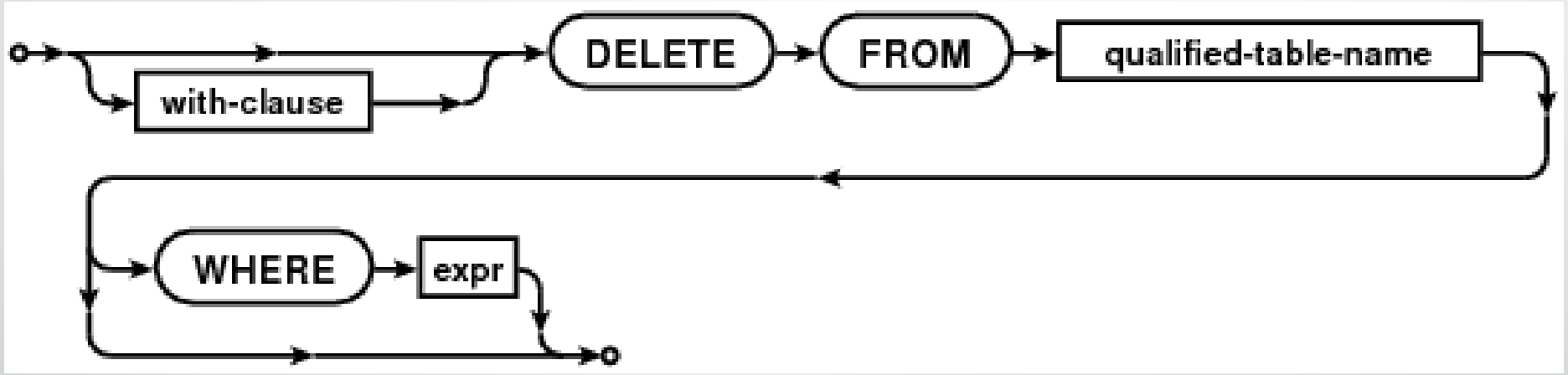
```
UPDATE  Komitent
SET Adresa = 'Pozeska 23'
WHERE IdKom = 10
```

Primeri (UPDATE)

47. Nikola je rešio da ugasi sve svoje račune. Napisati SQL skriptu za izmenu svih računa gde je IdKom jednako 10. Prilikom gašenja računa potrebno je dozvoljeni minus i stanje staviti na NULL i Status staviti na ugašen

```
UPDATE Racun
SET Stanje = NULL,  dozvMinus = NULL, Status = 'U'
WHERE IdKom = 10
```

DELETE naredba



Preuzeto: https://www.sqlite.org

Primeri (DELETE)

48. Napisati SQL skriptu za brisanje svih računa komitenta sa IdKom 10, a zatim i samog komitenta.

```
DELETE FROM Racun
WHERE IdKom = 10;


DELETE FROM Komitent
WHERE IdKom = 10;
```

Primeri (LIKE operator)

49. Prikazati sve filijale čija adresa ne sadrži reč „trg“ (vršiti case insensitive pretragu)

```
SELECT *
FROM Filijala F
WHERE F.Adresa NOT LIKE '%trg%'
```

LIKE operator je u SQLite case insensitive.

Ovo podrazumevano ponašanje menja se sa:
PRAGMA case_sensitive_like=ON;
PRAGMA case_sensitive_like=OFF;

Značenje specijalnih karaktera:
% - 0 ili više pojavljivanja nekih karaktera
_ - tačno jedno pojavljivanje nekog karaktera

Primeri (LIKE operator)

50. Prikazati koliko ima komitenata čije se ime sastoji od najmanje tri karaktera, a prvi karakter imena je slovo „m“.

```
SELECT COUNT(*)
FROM Komitent K
WHERE K.Naziv LIKE 'm__%'
```

Spojene dve donje crte

Primeri (LIKE operator)

51. Prikazati one filijale koje u svom nazivu ili u svojoj adresi imaju karakter "_"

```
SELECT *
FROM Filijala F
WHERE Naziv LIKE '%!_%' ESCAPE '!'
    OR Adresa LIKE '%#_%' ESCAPE '#'
```

Potrebno je koristiti escape karakter jer tražimo rezervisani znak („_“, „%“, „'“, ...) u tekstu.

LIMIT i OFFSET

Ako nam je potrebno da izvučemo samo određen broj redova iz rezultata nekog upita, a ne sve redove, onda možemo da koristimo klauzule **LIMIT** i **OFFSET**. Na primer prikazati prvih 10 redova rezultata ili na primer prikazati 5 redova od počev od stotog reda.

LIMIT i **OFFSET** se često koriste kod aplikacija gde je potrebno straničenje podataka.

```
SELECT ....
...
[LIMIT expLimit [OFFSET expOffset]]
```

LIMIT – definiše koliko maksimalno redova treba prikazati od rezultata upita

- Ako rezultat nema traženi broj redova, onda prikazuje koliko ih ima

OFFSET – definiše od kog reda u rezultatu treba primeniti **LIMIT**

- Mora da je naveden **LIMIT**
- Izostavljanjem klauzule se podrazumeva kao da je **OFFSET** postavljen na 0

LIMIT i OFFSET

- *expLimit* i *expOffset* mogu da budu:
  - celobrojna konstanta
  - izraz koji vraća (konvertovanjem) celobrojnu vrednost
    - **Ako izraz vrati NULL ili neku necelobrojnu vrednost → runtime greška**

- *expLimit* ako je negativan onda kao da je postavljen na onoliko redova koliko ima redova u rezultatu
- *expOffset* ako je negativan onda se ignoriše

- Primeri konvertovanja:
  - 3.0 → 3
  - 3.1 → **greška**
  - '3.0' → 3
  - '3' → 3
  - '3.1' → **greška**
  - Logički izraz (ako je TRUE → 1; ako je FALSE → 0)

LIMIT i OFFSET nisu standardne klauzule, pa se u različitim bazama ove funkcionalnosti definišu na drugačiji način

LIMIT i OFFSET (ostale baze)

- SELECT * FROM T LIMIT 10 OFFSET 20 -- Netezza, MySQL, PostgreSQL (also supports the standard, since version 8.4), SQLite, HSQLDB, H2
- SELECT * from T WHERE ROWNUM <= 10 -- Oracle (also supports the standard, since Oracle8i)
- SELECT FIRST 10 * from T -- Ingres
- SELECT FIRST 10 * FROM T order by a -- Informix
- SELECT SKIP 20 FIRST 10 * FROM T order by c, d -- Informix (row numbers are filtered after order by is evaluated. SKIP clause was introduced in a v10.00.xC4 fixpack)
- SELECT TOP 10 * FROM T -- MS SQL Server, Sybase ASE, MS Access
- SELECT TOP 10 START AT 20 * FROM T -- Sybase SQL Anywhere (also supports the standard, since version 9.0.1)
- SELECT FIRST 10 SKIP 20 * FROM T -- Interbase, Firebird
- SELECT * FROM T ROWS 20 TO 30 -- Firebird (since version 2.1)
- SELECT * FROM T WHERE ID_T > 10 FETCH FIRST 10 ROWS ONLY -- DB2

Standard ISO SQL2008:
SELECT * FROM T **ORDER BY acolumn DESC OFFSET 0 ROWS FETCH FIRST 10 ROWS ONLY**

Primeri (LIMIT i OFFSET)

52. Napisati upit koji vraća prvih 5 stavki koje su se desile nakon 10:00 sati.

```
SELECT *
FROM Stavka
WHERE Vreme>'10:00'
ORDER BY Vreme
LIMIT 5
```

ILI

```
SELECT *
FROM Stavka
WHERE Vreme>'10:00'
ORDER BY Vreme
LIMIT 5 OFFSET 0
```

Podrazumevano: OFFSET 0

53. Napisati upit koji vraća drugu i treću stavku po visini iznosa.

```
SELECT *
FROM Stavka
ORDER BY Iznos DESC
LIMIT 2 OFFSET 1
```

Primeri (LIMIT i OFFSET)

54. Napisati upit koji vraća prvu četvrtinu svih stavki.

```
SELECT *
FROM Stavka
LIMIT (SELECT COUNT(*) FROM Stavka)/4
```

55. Napisati upit koji vraća treću četvrtinu svih stavki.

```
SELECT *
FROM Stavka
LIMIT (SELECT COUNT(*) FROM Stavka)/4 OFFSET 2*(SELECT COUNT(*) FROM Stavka)/4
```

Krećem od reda koji je na polovini

Primeri (LIMIT i OFFSET)

56. Napisati upit koji vraća sve redove stavke nakon trećeg reda.

```
SELECT *
FROM Stavka
LIMIT -1 OFFSET 3
```

Može i bilo koji drugi negativan broj

Ugneždeni upiti

Ugneždeni upiti mogu biti:

- ***Nekorelisani*** – ugneždeni upit ne zavisi od spoljašnjih upita. Rezultat njegovog izvršavanja je isti bez obzira na spoljašnji upit. Može biti izvršen jednom.
- ***Korelisani*** – ugneždeni upit zavisi od bar jednog spoljašnjeg upita, tj. poseduje promenljivi deo čiju vrednost diktira spoljašnji upit. Rezultat njegovog izvršavanja je promenljiv. Mora biti izvršen za svaki red rezultata spoljašnjeg upita.

Primeri (Nekorelisani podupiti)

57. Prikazati sve uplate koje su veće od najveće uplate za komitenta čiji identifikator ima vrednost 1.

Nekorelisani podupiti

```
SELECT *
FROM Stavka
WHERE IdSta IN (SELECT IdSta FROM Uplata)
AND Iznos >
    (SELECT MAX(Iznos)
    FROM Stavka
    WHERE IdSta IN (SELECT IdSta FROM Uplata)
        AND IdRac IN (SELECT IdRac FROM Racun WHERE IdKom = 1))
```

Primeri (Korelisani podupiti)

58. Prikazati IdKom i Naziv onih komitenata koji imaju isti broj otvorenih računa kao komitent sa brojem 2.

```
SELECT K.IdKom, k.Naziv
FROM Komitent k
WHERE (
        SELECT COUNT(*)
        FROM Racun R
        WHERE R.IdKom = K.IdKom
    ) = (
        SELECT COUNT(*)
        FROM Racun
        WHERE IdKom = 2
    ) AND K.IdKom != 2
```

Korelisani podupit

Nekorelisani podupit

Operatori IN, ANY, ALL

Izraz **[NOT] IN** (Konstanta,...)

- Vrednost izraza jeste/nije u skupu.

Izraz { < | > | = | <= | >= | != | <> } **ANY** (Konstanta , ...)

- Vrednost izraza je [relacioni operator] od bar jedne vrednsoti iz skupa.
- Ukoliko ANY radi nad praznim skupom vraća se FALSE.

Izraz { < | > | = | <= | >= | != | <> } **ALL** (Konstanta , ...)

- Vrednost izraza je [relacioni operator] od svih vrednsoti iz skupa.
- Ukoliko ALL radi nad praznim skupom vraća se TRUE.

Umesto konstanti mogu se pisati i upiti.

*SQLite nije implemetirao operacije ANY I ALL.

Operator EXISTS

[NOT] EXISTS (RedniUpit)

- Ukoliko RedniUpit vrati bar jedan red EXISTS će vratiti TRUE, u suprotnom FALSE.

Primeri (Ugneždeni upiti, EXISTS, ALL)

59. Prikazati one komitente čije su sve uplate na račune bile iznad 20000. Prikazati i komitente koji nisu imali uplate na račune.

```
SELECT *
FROM Komitent K
WHERE NOT EXISTS(
    SELECT S.Iznos
    FROM Racun R, Stavka S, Uplata U
    WHERE R.IdKom = K.IdKom AND R.IdRac = S.IdRac
        AND S.IdSta = U.IdSta AND S.Iznos <= 20000
)
```

Primeri (Ugneždeni upiti, EXISTS, ALL)

Drugo rešenje:

```
SELECT *
FROM Komitent K
WHERE (
        SELECT MIN(S.Iznos)
        FROM Racun R, Stavka S, Uplata U
        WHERE R.IdKom = K.IdKom AND R.IdRac = S.IdRac  AND S.IdSta = U.IdSta
    ) > 20000
    OR (
        SELECT MIN(S.Iznos)
        FROM Racun R, Stavka S, Uplata U
        WHERE R.IdKom = K.IdKom AND R.IdRac = S.IdRac  AND S.IdSta = U.IdSta
    ) IS NULL
```

Podupiti su isti. SQL funkcija („min" u ovom slučaju) će vratiti NULL ukoliko nema vrednosti nad kojima radi. NULL prilikom poređenja sa bilo kojom vrednošću vraća FALSE, pa mora da postoji i drugi uslov koji će vratiti komitente bez uplata.

Primeri (Ugneždeni upiti, EXISTS, ALL)

Rešenje primenom operatora ALL.

```
SELECT *
FROM Komitent K
WHERE 20000 < ALL (
    SELECT s.Iznos
    FROM Racun R, Stavka S, Uplata U
    WHERE R.IdKom = K.IdKom AND R.IdRac = S.IdRac AND S.IdSta = U.IdSta
)
```

Napomena: Ovo rešenje neće raditi jer SQLite ne implementira operator ALL.

Primeri (IN, ANY)

60. Vratiti sve račune koji su blokirani ili ugašeni.

```
SELECT *
FROM Racun
WHERE Status IN ('B', 'U')
```

ili

```
SELECT *
FROM Racun
WHERE Status = ANY ('B', 'U')
```

Napomena:
Ovo rešenje neće raditi jer SQLite ne implementira operator ANY.

Primeri (Ugneždeni upiti, DELETE)

61. Obrisati komitente koji nemaju nijedan račun u banci.

```
DELETE FROM Komitent
WHERE NOT EXISTS (
    SELECT *
    FROM Racun
    WHERE IdKom = Komitent.IdKom
)
```

Ugneždeni SELECT iskaz može se koristiti u WHERE rečenici iskaza SELECT, DELETE, UPDATE.

UNION, UNION ALL, INTERSECT, EXCEPT

Nad rezultatima upita-tabelama definisane su skupovne operacije unije, preseka i razlike iako rezultati upita-tabele **ne predstavljaju skupove**. Skupovne operacije vrše spajanje rezultata upita-tabela po redovima.

U opštem slučaju kod rezultata upita-tabela može da postoji više jednakih redova.

UNION – unija
UNION ALL – unija sa ponavljanjem
INTERSECT – presek
EXCEPT – razlika

SELECT …
…
UNION/UNION ALL/INTERSECT/EXCEPT
SELECT …
…
UNION/UNION ALL/INTERSECT/EXCEPT
SELECT …
…

U slučaju redno povezanih upita sa skupovnim operacijama, operacije se izvršavaju redom.

UNION, UNION ALL, INTERSECT, EXCEPT

Za izvršavanje skupovnih operacija potrebno je da rezultati upita-tabele koje se spajaju zadovoljavaju unijsku kompatibilnost (*union-compatible*), a to znači da tabele koji se spajaju moraju da imaju:

1) Isti broj kolona
2) Kolone koje se spajaju moraju da imaju isti domen (tip, ograničenja)

Napomena:

SQLite ne poštuje pravilo 2), pa je moguće da u jednoj koloni ima više podataka koji međusobno nemaju isti domen.

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

62. Prikazati sve adrese komitenata i adrese filijala banke.

```
SELECT Adresa
FROM Komitent
UNION
SELECT Adresa
FROM Filijala
```

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

63. Prikazati sve adrese komitenata i adrese filijala banke. U slučaju da postoji više istih adresa prikazati ih sve (sa ponavljanjem).

```
SELECT Adresa
FROM Komitent

UNION ALL

SELECT Adresa
FROM Filijala
```

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

64. Prikazati sve adrese komitenata i adrese filijala banke. U slučaju da postoji više istih adresa prikazati ih jednom.

```
SELECT Adresa
FROM Komitent

UNION

SELECT Adresa
FROM Filijala
```

ILI

```
SELECT DISTINCT Adresa
FROM (  SELECT Adresa
        FROM Komitent

        UNION ALL

        SELECT Adresa
        FROM Filijala
)
```

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

65. Prikazati sve filijale gde je otvoren račun i pritom je bila barem jedna uplata ili isplata.

```
SELECT *
FROM Filijala
WHERE IdFil IN (    SELECT IdFil FROM Racun
                    INTERSECT
                    SELECT IdFil FROM Stavka
                )
```

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

66. Prikazati sve filijale gde su samo otvarani računi i pritom nije bila ni jedna uplata ili isplata.

```
SELECT *
FROM Filijala
WHERE IdFil IN (    SELECT IdFil FROM Racun
                    EXCEPT
                    SELECT IdFil FROM Stavka
                )
```

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

67. Uz račun je definisan stepen rizika na sledeći način:
- ako je stanje na računu pozitivno, stepen rizika je *'nizak*'
- ako je stanje na računu negativno i ako je (apsolutno) stanje manje od 90% dozvoljenog minusa, stepen rizika je *'srednji*'
- ako je stanje na računu negativno i ako je (apsolutno) stanje veće od 90% dozvoljenog minusa, stepen rizika je '*visok*'

Prikazati sve račune koji nisu ugašeni uz prikaz njegovog stepena rizika.

Primeri (UNION, UNION ALL, INTERSECT, EXCEPT)

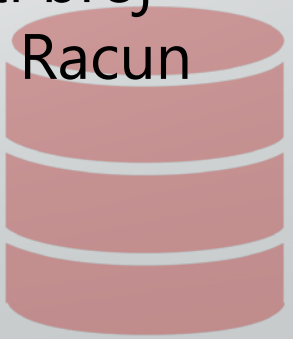
```
SELECT *, 'nizak' AS StepenRizika
FROM Racun
WHERE Stanje>=0

UNION

SELECT *, 'srednji' AS StepenRizika
FROM Racun
WHERE Stanje<0 AND DozvMinus*0.9 >= -Stanje

UNION

SELECT *, 'visok' AS StepenRizika
FROM Racun
WHERE Stanje<0 AND DozvMinus*0.9 < -Stanje
```

Primeri (CASE)

```
SELECT *, CASE
                WHEN Stanje>=0 THEN 'nizak'
                WHEN DozvMinus*0.9 >= -Stanje THEN 'srednji'
                ELSE 'visok'
          END AS StepenRizika
FROM Racun
WHERE Status != 'U'
```

Primeri (CASE)

68. Potrebno je proveriti da li se atribut BrojStavki iz tabele Racun odgovara broju stavki iz tabele Stavka. Rezultat upita treba da sadrži za svaki Racun izveštaj računa. Izveštaj može da ima tri vrednosti:
'Greška: višak stavki': Ako se u tabeli Stavka nalazi više stavki, nego što je navedeno u atributu BrojStavki iz tabele Racun
'Greška: manjak stavki': Ako se u tabeli Stavka nalazi manje stavki, nego što je navedeno u atributu BrojStavki iz tabele Racun
'OK' : Ako se u tabeli Stavka nalazi isti broj stavki kao što je navedeno u atributu BrojStavki iz tabele Racun



Primeri (CASE)

```
SELECT Racun.*,Br, CASE
                        WHEN Br>Racun.BrojStavki THEN 'Greška: višak stavki'
                        WHEN Br<Racun.BrojStavki THEN 'Greška: manjak stavki'
                        ELSE 'OK'
                   END AS Izvestaj
FROM (
        SELECT IdRac, COUNT(*) AS Br
        FROM Stavka
        GROUP BY IdRac) S, Racun
WHERE Racun.IdRac=S.IdRac
```

Primeri (CASE)

69. Potrebno je za svaki račun napisati pun naziv statusa računa (‘A’ - aktivan, ‘B’- blokiran, ‘U’ – ugašen).

```
SELECT IdRac, CASE Status
                    WHEN 'A' THEN 'Aktivan'
                    WHEN 'B' THEN 'Blokiran'
                    WHEN 'U' THEN 'Ugasen'
                    ELSE 'Greska'
              END AS Status, BrojStavki, DozvMinus, Stanje, IdFil, IdKom
FROM Racun
```

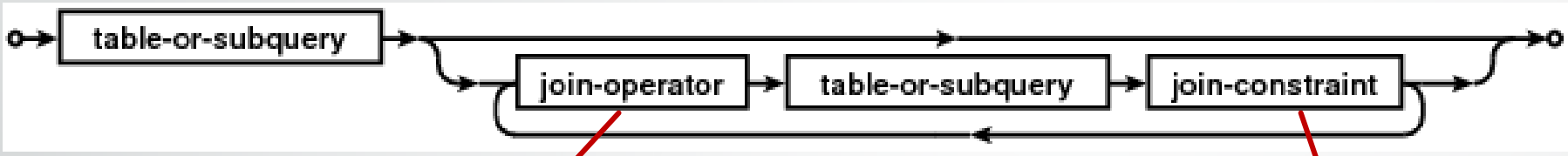
JOIN – CROSS, INNER, OUTER

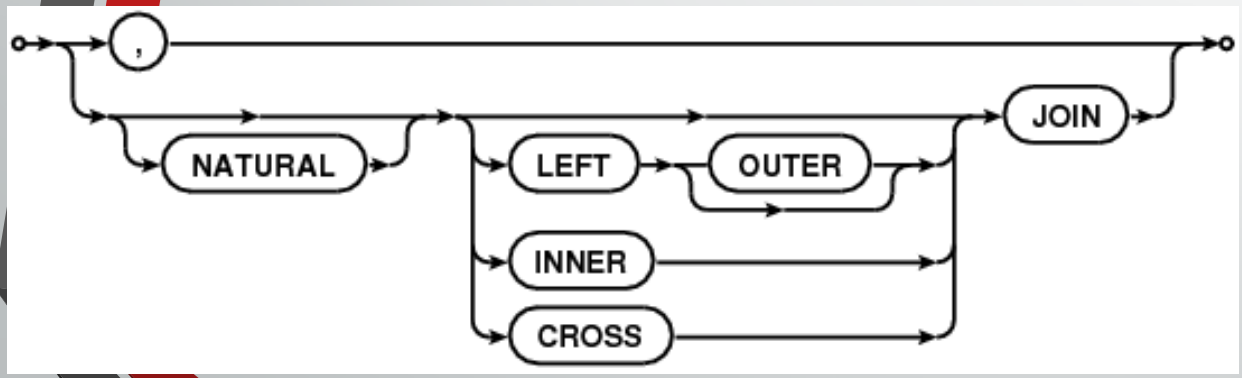
Kako bi odvojili kriterijum po kome spajamo tabele radi preglednosti, moguće je proširiti FROM klauzulu sa nekim od JOIN operatora.

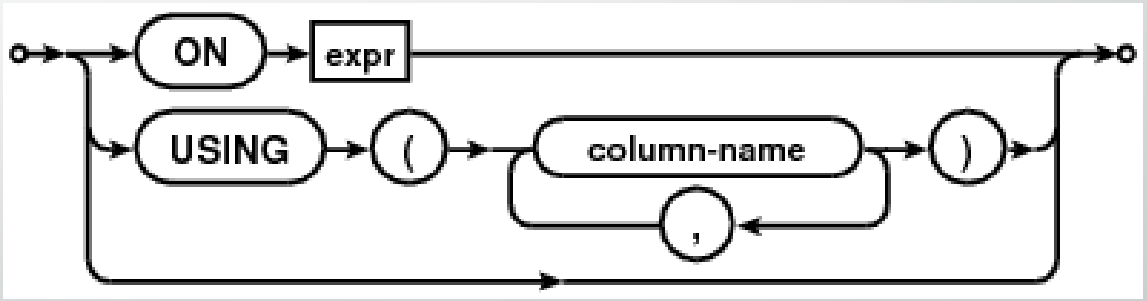
- CROSS JOIN – spajanje Dekartovim proizvodom
- INNER JOIN – unutrašnje spajanje
- OUTER JOIN – spoljašnje spajanje

INNER JOIN i OUTER JOIN su izvedene i mogu se realizovati preko standardnih operacija nad skupovima i uz zadovoljavanje kriterijuma u WHERE klauzuli.

JOIN – CROSS, INNER, OUTER







JOIN – CROSS

CROSS JOIN predstavlja spajanje Dekartovim proizvodom, gde se svaki red prve tabele uparuje sa svakim redom druge tabele. Ako prva tabela ima X redova, a druga tabela Y redova onda rezultat ima X*Y redova.

Implicitna notacija:

```
SELECT *
FROM Stavka, Uplata
```

Eksplicitna notacija:

```
SELECT *
FROM Stavka CROSS JOIN Uplata
```

JOIN – INNER

INNER JOIN predstavlja spajanje kod kog je potrebno zadovoljiti neki uslov. Uslov se definiše pomoću ključne reči ON.

Implicitna notacija:

```
SELECT *
FROM Stavka, Uplata
WHERE Stavka.IdSta = Uplata.IdSta
```

Napomena:
Reč INNER je podrazumevana, tako da je moguće umesto INNER JOIN napisati samo JOIN. Isti rezultat bi se dobio i kada bi se izostavile i obe ključne reči INNER JOIN.

Eksplicitna notacija:

```
SELECT *
FROM Stavka INNER JOIN Uplata ON (Stavka.IdSta = Uplata.IdSta)
```

JOIN – INNER

Kako se tabele najčešće spajaju po primarnim-stranim ključevima i tom prilikom se najčešće atributi zovu isto, onda je uveden ključna reč USING, čime se implicitno izjednačavaju navedene kolone iz obe tabele.

```
SELECT *
FROM Stavka INNER JOIN Uplata
WHERE Stavka.IdSta = Uplata.IdSta
```

```
SELECT *
FROM Stavka INNER JOIN Uplata USING (IdSta)
```

Obzirom da se izvršava izjednačavanje,
onda se kolone kojoj se izjednačavaju u rezultatu prikazuju samo jednom.

JOIN – INNER, NATURAL

Kako se izbeglo stalno navođenje istoimenih kolona pri korišćenju USING ključne reči, onda se uvelo novo spajanje NATURAL JOIN (prirodno spajanje). NATURAL JOIN izjednačava sve istoimene kolone iz tabela koje se spajaju.

```
SELECT *
FROM Stavka INNER JOIN Uplata USING (IdSta)
```

```
SELECT *
FROM Stavka NATURAL JOIN Uplata
```

Primeri (JOIN – INNER)

70. Prikazati sve uplate i filijale u kome je obavljena uplata.

```
SELECT *
FROM Stavka JOIN Uplata JOIN Filijala ON (Stavka.IdSta = Uplata.IdSta AND Filijala.IdFil=Stavka.IdFil)
```

```
SELECT *
FROM Stavka JOIN Uplata USING (IdSta) JOIN Filijala USING (IdFil)
```

U rezultatu se pojavljuju duple kolone (IdSta, IdFil)

```
SELECT *
FROM Stavka NATURAL JOIN Uplata NATURAL JOIN Filijala
```

Primeri (JOIN – INNER)

71. Prikazati sve stavke računa uz informacije i o trenutnom stanju na tom računu.

```
SELECT *
FROM Racun R JOIN Stavka S ON (R.IdRac = S.IdRac)
```

```
SELECT *
FROM Racun R JOIN Stavka S USING (IdRac)
```

```
SELECT *
FROM Racun NATURAL JOIN Stavka
```

→ Ovo nije ispravno, jer su se ispisale samo stavke računa koje su isplacene/uplaćene u istoj filijali u kojoj je i račun otvoren, zato što postoji idFill u obe tabele ali sa drugačijim značenjem.

JOIN – OUTER

U bazama podataka postoje tri vrste OUTER JOIN-a (spoljašnjih spajanja):

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

Napomena:
SQLite podržava samo LEFT OUTER JOIN, ostala spajanja su izvedena pa ih je moguće realizovati preko LEFT OUTER JOIN-a

Navedena spajanja vrše spajanje po zadatom kriterijumu (moguće kombinovati sa ON, USING, NATURAL). Redovi koji ispunjavaju uslov se spoje (kao INNER JOIN) i kao takvi ulaze u rezultat, a redovi koji ne ispunjavaju uslov se proširuju sa NULL vrednostima i onda ulaze u rezultat.

FULL OUTER JOIN služi za kreiranje denormalizovanih tabela.

JOIN – OUTER

72. Prikazati svaku stavku, a ako je stavka uplata prikazati i osnov.

```
SELECT Stavka.*, Uplata.Osnov
FROM Stavka LEFT OUTER JOIN Uplata ON Stavka.IdSta=Uplata.IdSta
```

```
SELECT Stavka.*, Uplata.Osnov
FROM Stavka LEFT OUTER JOIN Uplata USING (IdSta)
```

```
SELECT Stavka.*, Uplata.Osnov
FROM Stavka LEFT OUTER NATURAL JOIN Uplata
```

JOIN – OUTER

- Realizacija RIGHT OUTER JOIN-a korišćenjem LEFT OUTER JOIN-a:

```
SELECT *
FROM A RIGHT OUTER JOIN B
```



```
SELECT *
FROM B LEFT OUTER JOIN A
```

- Realizacija FULL OUTER JOIN-a korišćenjem LEFT OUTER JOIN-a:

```
SELECT *
FROM A FULL OUTER JOIN B
```



```
SELECT A.*,B.*
FROM A LEFT OUTER JOIN B

UNION

SELECT A.*,B.*
FROM B LEFT OUTER JOIN A
```

Primeri (JOIN – OUTER)

73. Prikazati za svaku filijalu koliko je računa otvoreno u njoj.

```
SELECT Filijala.IdFil, COUNT(IdRac)
FROM Filijala, Racun
WHERE Filijala.IdFil = Racun.IdFil
GROUP BY Filijala.IdFil
```

NE!

Da li će se u rezultat ući one filijale u kojima nije otvoren nijedan račun?

VS

DA

```
SELECT Filijala.IdFil, COUNT(IdRac)
FROM Filijala LEFT NATURAL OUTER JOIN Racun
GROUP BY Filijala.IdFil
```

Primeri (JOIN – OUTER)

Drugi način:

```
SELECT Filijala.IdFil, COUNT(IdRac)
FROM Filijala, Racun
WHERE Filijala.IdFil = Racun.IdFil
GROUP BY Filijala.IdFil

UNION

SELECT IdFil, 0
FROM Filijala
WHERE IdFil NOT IN ( SELECT idFil FROM Racun )
```

COALESCE

COALESCE(param1, param2,...) je funkcija koja vraća vrednost prvog parametra koji nije NULL. Ukoliko svi parametri imaju NULL vrednost, vraća se NULL.

COALESCE(param1, param2,...) je ekvivalentnt sa:

```
CASE   WHEN param1 IS NOT NULL THEN param1
       WHEN param2 IS NOT NULL THEN param2
       ...
       ELSE NULL
END
```

Primeri (JOIN – OUTER, COALESCE)

74. Prikazati za svaku filijalu kolika je suma na računima otvorenim u njoj.

```
SELECT Filijala.IdFil, COALESCE( SUM(Stanje), 0) AS Suma
FROM Filijala LEFT NATURAL OUTER JOIN Racun
GROUP BY Filijala.IdFil
```

Primeri (CREATE, FOREIGN KEY)

75. Napisati SQL skriptu za kreiranje tabele **Stavka**. *IdSta* je celobrojna veličina koja identifikuje stavku i dodeljuje se automatski. *RedniBroj* i *Iznos* su celobrojne veličine i obavezni su. *Datum* je tipa Date. *Vreme* je tipa Time. *IdFil* i *IdRac* su strani ključevi i obavezni su.

```
CREATE TABLE Stavka (
        IdSta      INTEGER   PRIMARY KEY AUTOINCREMENT,
        RedBroj    INTEGER NOT NULL,
        Datum      Date,
        Vreme      Time,
        Iznos      INTEGER NOT NULL,
        IdFil      INTEGER NOT NULL,
        IdRac      INTEGER NOT NULL,
        FOREIGN KEY(IdFil) REFERENCES Filijala (IdFil),
        FOREIGN KEY(IdRac) REFERENCES Racun (IdRac)
);
```

Primeri (CREATE, CHECK, DEFAULT)

76. Napisati SQL skriptu za kreiranje tabele **Racun**. *IdRac* je celobrojna veličina koja identifikuje stavku i dodeljuje se automatski. *Stanje i DozvMinus* i *BrojStavki* su celobrojne veličine i obavezni su i podrazumevano dobijaju vrednost 0. *BrojStavki* je nenegativna celobrojna veličina i obavezna je. *Status* je karakter koji može imati samo vrednosti 'A', 'B' i 'U' I prilikom kreiranja Racuna podrazumevano dobija vrednost 'A'. *IdFil* i *IdKom* su strani ključevi i obavezni su.

```
CREATE TABLE IF NOT EXISTS Racun (
    IdRac        INTEGER   PRIMARY KEY AUTOINCREMENT,
    Status       CHAR CHECK ( Status IN ('A', 'B', 'U') ) DEFAULT 'A',
    BrojStavki   INTEGER CHECK ( BrojStavki >= 0 ) DEFAULT 0 NOT NULL,
    DozvMinus    INTEGER DEFAULT 0 NOT NULL,
    Stanje       INTEGER DEFAULT 0 NOT NULL,
    IdFil        INTEGER NOT NULL,
    IdKom        INTEGER NOT NULL,
    FOREIGN KEY(IdFil) REFERENCES Filijala (IdFil),
    FOREIGN KEY(IdKom) REFERENCES Komitent (IdKom)
);
```

Korisna opcija kod pravljenja tabela jeste struktura IF NOT EXISTS, koja kreira tabelu jedino ako tabelu koju želimo da napravimo već ne postoji kao deo šeme.

Primeri (DROP)

77. Napisati SQL skriptu za uklanjanje tabele **Racun**.

```
DROP TABLE Racun;
```

```
DROP TABLE IF EXISTS Racun;
```

DROP TABLE uklanja tabelu iz šeme (suprotno od CREATE TABLE).

Ne mešati DROP TABLE Racun sa DELETE FROM Racun.
DELETE FROM briše podatke, dok tabela ostaje kao deo šeme.

Primeri (DROP)

78. Napisati SQL skriptu za uklanjanje pogleda **PozitivniRacuni**.

```
DROP VIEW PozitivniRacuni;
```

```
DROP VIEW IF EXISTS PozitivniRacuni;
```

NULLIF

NULLIF(param1,param2) je ekvivalentnt sa:

CASE WHEN param1 = param2 THEN NULL ELSE param1 END

79. Prebrojati sve neaktivne račune:

```
SELECT COUNT(NULLIF(Status, 'A'))
FROM Racun
```

WITH

Ako se u upitu koristi više puta isti podupit, može se koristit CTE (Common Table Expressions).

CTE predstavlja kao pomoćnu tabelu koja je deo samog upita.

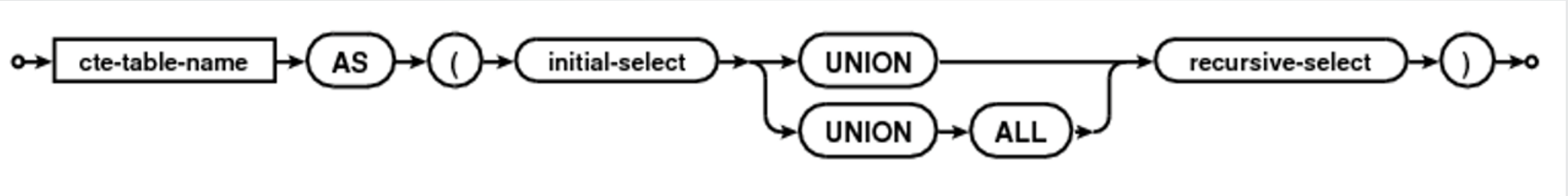
Za slične potrebe smo koristili VIEW, međutim **VIEW predstavlja deo šeme**.

Primeri (WITH)

80. Napisati SQL upit koji vraća u dva reda minimalno i maksimalno stanje onih računa koji su otvoreni u Beogradu. Prvi red treba da sadrži minimalno, a drugi red maksimalno stanje. U slučaju da nije otvoren ni jedan račun u Beogradu, vratiti 0 u oba reda.

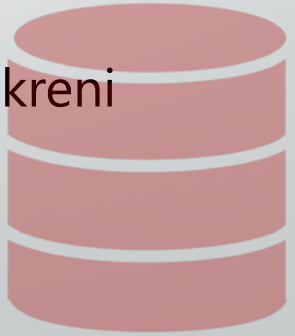
```
WITH BGStanjeRacuna (Stanje) AS
(
    SELECT Racun.Stanje
    FROM Racun NATURAL JOIN Filijala JOIN Mesto USING (IdMes)
    WHERE Mesto.Naziv='Beograd'
)
SELECT COALESCE(MAX(Stanje),0) AS Stanje FROM BGStanjeRacuna
UNION ALL
SELECT COALESCE(MIN(Stanje),0) FROM BGStanjeRacuna
```

Rekurzija – WITH RECURSIVE



Algoritam:

1. Izvrši initial-select i rezultat stavi u red (queue)
2. Sve dok red nije prazan:
   a) Uzmi jedanu torku iz reda
   b) Ubaci uzetu torku u rekurzivnu tabelu (cte-table-name)
   c) Pretvaraj se da je uzeta torka jedina torka u rekurzivnoj tabeli i pokreni recursive-select, pa dodaj rezultat u red



Primeri (Rekurzija)

81. Selektovati cele brojeve između 1 i 1000000

```
WITH RECURSIVE CNT (X) AS (
        VALUES(1)
        UNION ALL
        SELECT X+1 FROM CNT WHERE X<1000000
)
SELECT X FROM CNT;
```

Primeri (Rekurzija)

Drugo rešenje:

```
WITH RECURSIVE CNT (X) AS (
    SELECT(1)
    UNION ALL
    SELECT X+1 FROM CNT
    LIMIT 1000000
)
SELECT X FROM CNT;
```

Primeri (Rekurzija)

82. Dohvatiti prosečnu visinu radnika koji su podređeni radniku 3.

RADNIK(IdRad, Ime, IdRadNad, Visina)

Primeri (Rekurzija)

```
WITH RECURSIVE Podredjeni (IdRad) AS (
    SELECT IdRad
    FROM Radnik WHERE IdRad = 3
    UNION
    SELECT r.IdRad
    FROM Radnik r JOIN Podredjeni p ON (r.IdRadNad = p.IdRad)
)
SELECT AVG(Visina)
FROM Podredjeni p, Radnik r
WHERE r.IdRad != 3 AND r.IdRad = p.IdRad
```

Primeri (Rekurzija)

83. Dohvatiti sve radnike koji su podređeni radniku 3 pretragom stabla po širini i za svakog ispisati i nivo u hijerarhiji ispod radnika 3:

```
WITH RECURSIVE Podredjeni (IdRad, Nivo) AS (
    VALUES(3, 0)
    UNION
    SELECT r.IdRad, p.Nivo + 1
    FROM Radnik r JOIN Podredjeni p ON (r.IdRadNad = p.IdRad)
    ORDER BY 2 ASC
    LIMIT -1 OFFSET 1
)
SELECT r.*, p.Nivo
FROM Podredjeni p, Radnik r
WHERE r.IdRad = p.IdRad
```

ORDER BY u rekurzivnom SELECT-u (posle UNION ili UNION ALL) sortira red za čekanje, pa definiše koji će red iz queue-a biti sledeći uzet.
LIMIT definiše koliko redova maksimalno može biti u rekurzivnoj tabeli
OFFSET definiše koliko prvih redova ne staviti iz queue u rekurzivnu tabelu (ovi redovi će biti uzeti iz queue i obrađeni kao i ostali)

Primeri (Rekurzija)

84. Dohvatiti sve radnike koji su podređeni radniku 3 pretragom stabla po dubini i za svakog ispisati i nivo u hijerarhiji ispod radnika 3:

```
WITH RECURSIVE Podredjeni (IdRad, Nivo) AS (
    VALUES(3, 0)
    UNION
    SELECT r.IdRad, p.Nivo + 1
    FROM Radnik r JOIN Podredjeni p ON (r.IdRadNad = p.IdRad)
    ORDER BY 2 DESC
    LIMIT -1 OFFSET 1
)
SELECT r.*, p.Nivo
FROM Podredjeni p, Radnik r
WHERE r.IdRad = p.IdRad
```

Primeri (Rekurzija)

85. Dohvatiti imena svih živih predaka sortiranih po datumu rođenja za osobu sa brojem 12. DatumSmrti je NULL, kada je osoba živa.

PORODICA(Id, Ime, IdOtac, IdMajka, DatumRodjenja, DatumSmrti)

Primeri (Rekurzija)

```
WITH RECURSIVE
  Roditelji(Id, IdRoditelja) AS (
        SELECT Id, IdMajka FROM Porodica
        UNION
        SELECT Id, IdOtac FROM Porodica
),
  Preci(Id) AS (
        SELECT IdRoditelja FROM Roditelji WHERE Id = 12
        UNION ALL
        SELECT Roditelji.IdRoditelja FROM Roditelji JOIN Preci ON(preci.id = roditelji.id)
)
SELECT Porodica.Ime FROM Preci, Porodica
WHERE Preci.Id=Porodica.Id and Porodica.DatumSmrti is null
ORDER BY Porodica.DatumRodjenja;
```

Primeri (Rekurzija)

Drugo rešenje:

```
WITH RECURSIVE Preci(Id, IdOtac, IdMajka) AS(
      SELECT Id, IdOtac, IdMajka FROM Porodica WHERE Id = 12
      UNION ALL
      SELECT Porodica.Id, Porodica.IdOtac, Porodica.IdMajka
      FROM Porodica,  Preci
      WHERE Preci.IdOtac=Porodica.Id OR Preci.IdMajka=Porodica.Id
)
SELECT Porodica.Ime
FROM Preci, Porodica
WHERE Preci.Id=Porodica.Id AND Porodica.Id!=12
       AND Porodica.DatumSmrti IS NULL
ORDER BY Porodica.DatumRodjenja;
```

Primeri (Rekurzija)

86. Svaki commit (postavljanje nove verzije softvera) u VCS (Version Control System) i njihov redosled pamti se u bazi podataka. Iz jednog commit-a može nastati jedan ili više novih commit-ova (stvaraju se odvojene grane - branches), a grane se mogu spojiti (merge) kreiranjem commita od dva ili više postojećih. Smatrati da nije moguće napraviti dva commit-a u istom trenutku. Prikazati poslednjih 5 commit-ova koji su se dogodili pre commit-a 8.

COMMIT(Id, Vreme)

GRANA(Od, Do)

Primeri (Rekurzija)

```
WITH RECURSIVE Predak(Id, Vreme) AS (
      SELECT Id, Vreme FROM 'Commit' WHERE Id = 8
      UNION
      SELECT Grana.Od, 'Commit'.Vreme
      FROM Predak, Grana, 'Commit'
      WHERE Predak.Id=Grana.Do AND ’Commit'.Id=Grana.Od
      ORDER BY 'Commit'.Vreme DESC
      LIMIT 6
  )
SELECT * from Predak WHERE Id!=8
```