ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ВОВЕД ВО SQL:

Data Manipulation Language (DML)

БАЗИ НА ПОДАТОЦИ - предавања

Проф. д-р Слободан Калајџиски

Прашалници за пребарување во SQL

- SQL има една основна наредба за извлекување на податоци од базата – тоа е наредбата **SELECT**
- Битна разлика помеѓу SQL и формалниот релациски модел:
  - SQL дозволува некоја табела (релација) да има два или повеќе реда (торки) кои што се идентични за сите вредности на атрибутите
  - Оттука, SQL табела (релација) е **повеќекратно множество** (**multi-set**), наречен и **вреќа** (**bag**) од торки; НЕ е множество од торки

SQL релациите може да бидат ограничени да станат множества со одредување на PRIMARY KEY или UNIQUE за некои атрибути, или со употреба на опцијата DISTINCT во прашалниците

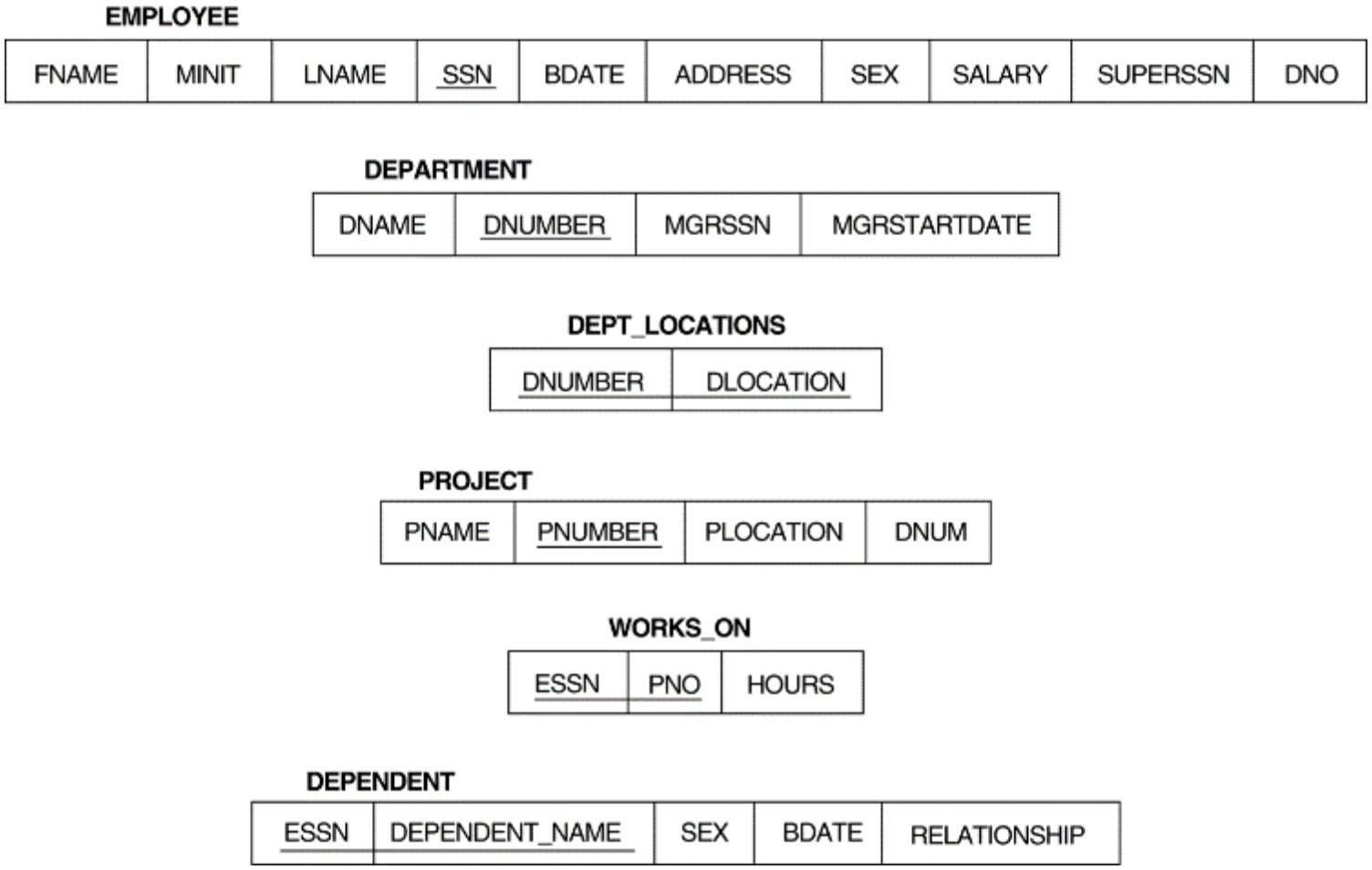
Прашалници за пребарување во SQL (2)

- Основниот облик на SQL-SELECT наредбата е:

```
SELECT   <attribute list>
FROM     <table list>
WHERE    <condition>
```

- <attribute list> е список на имиња на атрибути чиишто вредности треба да бидат извлечени со прашалникот од базата
- <table list> е список на имиња на релациите од кои ќе се влечат податоците потребни за обработка на прашалникот
- <condition> е некој логички израз кој треба да го задоволат торките што треба да бидат извлечени со прашалникот (филтрирање)
  - =, <, <=, >, >=,<>
  - AND, OR, NOT

Пример на шема на БП Копманија



Состојба на базата на податоци

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

Едноставни SQL прашалници

- Основните SQL прашалници соодветствуваат на примената на следниве операции во релационата алгебра:
  - SELECT
  - PROJECT
  - JOIN
- Сите следни примери ја користат БП COMPANY

Едноставни SQL прашалници

- **Прашалник 0.** Да се врати (добие) датумот на раѓање и адреса на вработениот (или вработените) чие име е 'John B. Smith'

```
Q0: SELECT BDATE, ADDRESS
    FROM EMPLOYEE
    WHERE FNAME='John' AND MINIT='B'
          AND LNAME='Smith'
```

  - Слично на парот SELECT-PROJECT кај операциите во релациона алгебра:
    - SELECT-делот ги одредува атрибутите што се проектираат
      WHERE-делот ги одредува условите за избор (селекција)
  - Сепак, резултатот од прашалникот **може да содржи дупликат торки**

Едноставни SQL прашалници

- Операторот **LIKE** се употребува за да се споредат делумни низи од знаци
- Два резервирани знаци кои што се користат за тоа се:
  - '%' (или '*' во некои варијанти) заменува низа со произволен број на знаци, и
  - '_' заменува точно еден произволен знак

Едноставни SQL прашалници

- **Прашалник 25.** Најди ги сите вработени чиишто адреси се наоѓаат во Houston, Texas.

```
Q25: SELECT FNAME, LNAME
     FROM EMPLOYEE
     WHERE ADDRESS LIKE '%Houston,TX%'
```

  - Овде, вредноста на атрибутот ADDRESS мора да ја содржи подниза та 'Houston,TX‘

- **Прашалник 26.** Најди ги сите вработени родени во текот на пеесетите години.

```
Q26: SELECT FNAME, LNAME
     FROM EMPLOYEE
     WHERE BDATE LIKE '_________5_'
```

  - Овде, '5' мора да е 9-иот знак на низата (формат на датум dd-mm-yyyy), така што вредноста што BDATE треба да ја задоволи е '_____5_’.

Едноставни SQL прашалници

- Стандардните аритметички оператори ‘**+**’, ‘**-**’, ‘*****’, и ‘**/**’ (за собирање, одземање, множење и делење) може да се применуваат врз бројчани вредности во резултатите од SQL прашалниците

- **Прашалник 27.** Прикажи го ефектот од зголемување на платата за 10% на вработениот ‘John B. Smith’.

```
Q27: SELECT FNAME, LNAME, 1.1*SALARY
     FROM EMPLOYEE
     WHERE FNAME='John' AND MINIT='B'
           AND LNAME='Smith'
```

Едноставни SQL прашалници

- **Прашалник 1.** Вратете ги имињата и адресите на сите вработени кои работат во одделот ’Research’.

```
Q1: SELECT FNAME, LNAME, ADDRESS
    FROM EMPLOYEE, DEPARTMENT
    WHERE DNAME='Research' AND DNUMBER=DNO
```

  - Слично на низата SELECT-PROJECT-JOIN кај операциите во релационата алгебра
  - (DNAME='Research') е услов за избор (соодветствува на SELECT операцијата во релациона алгебра)
  - (DNUMBER=DNO) е услов за спојување (соодветствува на JOIN операцијата во релациона алгебра)

Едноставни SQL прашалници

- **Прашалник 2.** За секој проект лоциран во 'Stafford', излистајте го бројот на проектот, бројот на управувачкиот оддел и презимето, адресата и датумот на раѓање на раководителот на одделот.

```
Q2: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
    FROM PROJECT, DEPARTMENT, EMPLOYEE
    WHERE DNUM=DNUMBER AND MGRSSN=SSN
        AND PLOCATION='Stafford'
```

  - Во Q2, има два услови за спојување
  - Условот за спојување DNUM=DNUMBER го поврзува проектот со неговиот управувачки оддел
  - Условот за спојување MGRSSN=SSN го спојува управувачкиот оддел со вработениот кој е раководител на тој оддел

Прекари – преименување во SQL

- Во SQL, може да се употребува истото име за два (или повеќе) атрибути сè дури атрибутите се во *различни релации*
- Прашалник што користи два или повеќе атрибути со исто име, мора пред името на атрибутот да го наведе името на релацијата, за да се избегне **двозначноста**
  - На пример:
    EMPLOYEE.LNAME, **DEPARTMENT.**DNAME
- Во некои прашалници треба да се повикаат истите релации повеќе од еднаш
  - Во овој случај, прекари (aliases) се даваат на името на релацијата
  - На пример:
    EMPLOYEE E, **EMPLOYEE** AS EMP

Прекари – преименување во SQL

- **Прашалник 8.** За секој вработен, врати го името и презимето на вработениот, како и името и презимето на неговиот шеф.

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM EMPLOYEE E S
    WHERE E.SUPERSSN = S.SSN
```

- Во Q8, алтернативните имиња на релациите E и S се наречени **прекари** (**aliases**) или **променливи на торка** (**tuple variables**) за релацијата EMPLOYEE
- Може да ги сметаме E и S како две различни *копии* на EMPLOYEE; E ги претставува вработените во улога на *надгледувани* (*supervisees*), а S ги претставува вработените во улога на *надзорници* (*supervisors*)

Прекари – преименување во SQL

- Може да се употреби и клучното зборче AS за да се одредат прекари

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM EMPLOYEE AS E, EMPLOYEE AS S
    WHERE E.SUPERSSN=S.SSN
```

Неодреден WHERE-дел

- Испуштен *WHERE-дел* покажува дека не постои услов; оттука, сите торки во релациите од FROM-делот се избрани
  - Еквивалентно е со условот WHERE TRUE

- **Прашалник 9.** Дај ги SSN вредностите за сите вработени.

```
Q9: SELECT SSN
    FROM EMPLOYEE
```

Неодреден WHERE-дел

- Ако повеќе од една реалција е наведена во FROM-делот и ако не постои услов за спојување, тогаш се добива Декартов производ (CARTESIAN PRODUCT) на торките

```
Q10: SELECT SSN, DNAME
     FROM EMPLOYEE, DEPARTMENT
```

Исклучително е важно да не се заборави да се наведат условите за избор и за спојување во WHERE-делот;
инаку, како резултат може да се добијат **неточни** и **многу големи релации!**

Употреба на *

- За да се извлечат сите вредности на атрибутите на избраните торки, се става *, која што значи **сите атрибути**

- Примери:

```
Q1C: SELECT *
     FROM EMPLOYEE
     WHERE DNO=5

Q1D: SELECT *
     FROM EMPLOYEE, DEPARTMENT
     WHERE DNAME='Research' AND DNO=DNUMBER
```

Употреба на DISTINCT

- SQL не ги третира релациите како множества
  - може да се појават дупликат-торки
- За да се отстранат дупликатите на торките во резултатите од прашалниците, се употребува клучниот збор **DISTINCT**
- Пример, Врати ги платите на сите вработени

```
Q11:   SELECT SALARY
       FROM EMPLOYEE

Q11A:  SELECT DISTINCT SALARY
       FROM EMPLOYEE
```

Q11:

Salary
30000
40000
25000
43000
38000
25000
25000
55000

Q11A:

Salary
30000
40000
25000
43000
38000
55000

Операции со множества

- SQL има директно вклучено некои од операциите за работа со множества:
  - UNION
  - MINUS
  - INTERSECT
- Разлика и пресек се дефинирани само во некои верзии на SQL
  - Резултантните релации од овие операции се множества
    - торките-дупликати се отстрануваат од резултатот
- Примена само врз релации кои се компатибилни по тип

Операции со множества

- **Прашалник 4.** Излистајте ги сите називи на проекти, за проектите кои вклучуваат вработен со презиме 'Smith' како вработен или како раководител на одделот кој управува со проектот.

```
Q4: (SELECT PNAME
     FROM PROJECT, DEPARTMENT, EMPLOYEE
     WHERE DNUM=DNUMBER AND MGRSSN=SSN
           AND LNAME='Smith')
    UNION
    (SELECT PNAME
     FROM PROJECT, WORKS_ON, EMPLOYEE
     WHERE PNUMBER=PNO AND ESSN=SSN
           AND LNAME='Smith')
```

Експлицитни множества

- Можно е да се користат и **експлицитни (набројиви) множества од вредности** во WHERE-делот
- Споредбата на вредностите се прави со операторот **IN**
- **Прашалник 13**. Врати ги матичните броеви на сите вработени кои работат на проектите со кодни броеви 1, 2 и 3.

```
Q13: SELECT DISTINCT ESSN
     FROM WORKS_ON
     WHERE PNO IN (1, 2, 3)
```

Вгнездување на прашалници

- Целосен SELECT прашалник, наречен **вгнезден** (*nested*) прашалник, може да биде вметнат во рамките на WHERE-делот од другиот прашалник, наречен надворешен прашалник
  - Некои од претходните прашалници може да бидат препишани во друг облик со употреба на вгнездување

Прашалник 1. Вратете ги имињата и адресите на сите вработени кои работат во одделот 'Research'.

```
Q1: SELECT FNAME, LNAME, ADDRESS
    FROM EMPLOYEE
    WHERE DNO IN ( SELECT DNUMBER
                   FROM DEPARTMENT
                   WHERE DNAME='Research' )
```

Вгнездување на прашалници

- Во општ случај, може да имаме неколку нивоа на вгнездени прашалници
- Употреба на името на атрибут без означување на табелата, за атрибут којшто се сретнува во неколку нивоа на прашалникот, се однесува на табелата декларирана во **најдлабоко вгнездениот прашалник**
- Прашалник со вгнездени SELECT... FROM... WHERE... блокови кои го употребуваат = или IN операторот за споредување можат секогаш да бидат изразени како прашалник од еден блок (со спојување)

Поврзани вгнездени прашалници

- Кога услов од WHERE делот на вгнезден прашалник пристапува до атрибут од релацијата наведена во надворешниот прашалник, тогаш за двата прашалници се вели дека се **меѓуповрзани (корелирани)**
- Резултатот на поврзаниот вгнезден прашалник е различен за секоја торка (или комбинација на торки) за релацијата (релациите) на надворешниот прашалник

Вгнездени прашалници и спојување

- **Прашалник 12.** Дај ги имињата на сите вработени кои имаат издржувани лица со исто име како и вработениот (меѓуповрзани прашалници).

```
Q12:  SELECT   E.FNAME, E.LNAME
      FROM EMPLOYEE AS E
      WHERE E.SSN IN (SELECT ESSN
                      FROM DEPENDENT
                      WHERE ESSN=E.SSN AND
                            E.FNAME=DEPENDENT_NAME)
```

- Q12 може да биде препишан и како Q12A

```
Q12A:  SELECT E.FNAME, E.LNAME
       FROM EMPLOYEE E, DEPENDENT D
       WHERE E.SSN=D.ESSN AND
             E.FNAME=D.DEPENDENT_NAME
```

Функцијата EXISTS

- EXISTS се употребува за да се провери дали резултатот на поврзаниот вгнезден прашалник е празен (дали не содржи ниту една торка) или не
  - Ова се проверува за секоја торка од надворешниот SELECT прашалник и доколку има торка во внатрешниот прашалник, тогаш надворешната торка се враќа во резултатот

```
Q12B: SELECT FNAME, LNAME
      FROM EMPLOYEE
      WHERE EXISTS (SELECT *
                    FROM DEPENDENT
                    WHERE SSN=ESSN AND
                          FNAME=DEPENDENT_NAME)
```

Функцијата EXISTS

- **Прашалник 6.** Прикажи ги имињата на вработените кои немаат издржувани лица.

```
Q6: SELECT FNAME, LNAME
    FROM EMPLOYEE
    WHERE NOT EXISTS (SELECT *
                      FROM DEPENDENT
                      WHERE SSN=ESSN)
```

- Поврзаниот вгнезден прашалник ги извлекува сите торки од DEPENDENT што се поврзани со тековната торка од EMPLOYEE
- Ако **не постои ниту една** торка во вгнездениот прашалник, само тогаш се избира торката од EMPLOYEE (во надворешниот прашалник) што ќе се врати во резултатот

NULL вредности во SQL прашалници

- SQL дозволува прашалници кои што проверуваат дали некоја вредност е **NULL** (испуштена или недефинирана или неприменлива)
- Споредбата со еднаквост не е соодветна
  - се смета дека секоја NULL вредност е различна од другите NULL вредности
- SQL ги користи **IS NULL** или **IS NOT NULL** за споредба на NULL вредностите

NULL вредности во SQL прашалници

- **Прашалник 14.** Врати ги имињата на сите вработени кои што немаат надзорници.

```
Q14: SELECT FNAME, LNAME
     FROM EMPLOYEE
     WHERE SUPERSSN IS NULL
```

Ако се специфицира услов за спојување, торките со NULL вредности на атрибутите според кои се прави спојувањето не се вклучуваат во резултатот

Споени табели во SQL

- Во FROM-делот може да се специфицира “споена релација”
  - Изгледа како било која друга релација, но е резултат од примената на операцијата спојување
  - Овозможува да одредиме различни видови на спојувања
    - регуларен "theta" JOIN,
    - NATURAL JOIN,
    - LEFT OUTER JOIN,
    - RIGHT OUTER JOIN,
    - CROSS JOIN итн.

Споени табели во SQL

- Пример: Врати ги имињата и презимињата на **сите** вработени заедно со имињата и презимињата на нивните шефови.

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM EMPLOYEE E S
    WHERE E.SUPERSSN=S.SSN
```

- може да биде пренапишано како:

```
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
    FROM (EMPLOYEE E LEFT OUTER JOIN EMPLOYEE S
                     ON E.SUPERSSN=S.SSN)
```

Споени табели во SQL

```
Q1: SELECT FNAME, LNAME, ADDRESS
    FROM EMPLOYEE, DEPARTMENT
    WHERE DNAME='Research' AND DNUMBER=DNO
```

- може да биде напишан како:

```
Q1*: SELECT FNAME, LNAME, ADDRESS
     FROM (EMPLOYEE
           JOIN DEPARTMENT ON DNUMBER=DNO)
     WHERE DNAME='Research'
```

- или како:

```
Q1**: SELECT FNAME, LNAME, ADDRESS
      FROM (EMPLOYEE
            NATURAL JOIN
            (DEPARTMENT AS
             DEPT(DNAME, DNO, MSSN, MSDATE)))
      WHERE DNAME='Research'
```

Битно е DNUMBER да се преименува во DNO за да може NATURAL JOIN

Споени табели во SQL

- **Прашалник 2.** За секој проект лоциран во 'Stafford', излистајте го бројот на проектот, бројот на управувачкиот оддел и презимето, адресата и датумот на раѓање на раководителот на одделот.

```
Q2: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
    FROM  PROJECT, DEPARTMENT, EMPLOYEE
    WHERE DNUM=DNUMBER AND MGRSSN=SSN
          AND PLOCATION='Stafford'
```

- Може да се презапише како:

```
Q2: SELECT PNUMBER,DNUM,LNAME,ADDRESS,BDATE
    FROM ((PROJECT JOIN
           DEPARTMENT ON DNUM=DNUMBER)
           JOIN EMPLOYEE ON MGRSSN=SSN)
    WHERE PLOCATION='Stafford'
```

Агрегатни функции

- Се користат да пресметаат некоја агрегатна функција врз множество од записи
  - Најчесто користени агрегатни функции:
    - **COUNT**
    - **SUM**
    - **MAX**
    - **MIN**
    - **AVG**

Во општ случај, NULL вредностите се **занемаруваат** кога се применуваат агрегатни функции врз одредена колона (атрибут)

Резултатот што се враќа е секогаш една вредност, а не множество од повеќе записи

Агрегатни функции

- **Прашалник 15.** Најди ги најголемата, најмалата и просечната плата меѓу сите вработени.

```
Q15:  SELECT MAX(SALARY),
             MIN(SALARY),
             AVG(SALARY)
      FROM EMPLOYEE
```

- **Прашалник 16.** Најди ја најголемата, најмалата и просечната плата кај вработените кои работат во одделот 'Research'.

```
Q16:  SELECT MAX(SALARY), MIN(SALARY),
             AVG(SALARY)
      FROM EMPLOYEE, DEPARTMENT
      WHERE DNO=DNUMBER AND DNAME='Research'
```

Агрегатни функции

- **Прашалник 17.** Врати го вкупниот број на вработени во компанијата.

```
Q17: SELECT COUNT(*)
     FROM EMPLOYEE
```

- **Прашалник 18.** Врати го вкупниот бројот на вработени во одделот 'Research'.

```
Q18: SELECT COUNT(*)
     FROM EMPLOYEE, DEPARTMENT
     WHERE DNO=DNUMBER AND DNAME='Research'
```

Агрегатни функции (COUNT)

- Во претходните примери ѕвездичката (*) ги означува редовите (торките), така што COUNT(*) во резултатот од прашалникот го дава бројот на редови
- Можеме да ја користиме функцијата COUNT за броење на вредности од некоја колона наместо да броиме торки
  - Ако станува збор за не-клучен атрибут, дупликатите не се отстрануваат, а NULL вредностите се изоставуваат
- **Прашалник 23.** Дајте го вкупниот број на различни вредности на плати во базата.

```
Q23: SELECT COUNT(DISTINCT Salary)
     FROM Employee;
```

Групирање

- Во многу случаи сакаме да ги примениме агрегатните функции на **подгрупи од торките** во резултатот
- Секоја подгрупа од торки се состои од множество на торки што ја имаат истата вредност на групирачкиот атрибут (или групирачките атрибути)
- Агрегатната функцијата се применува за секоја подгрупа посебно
- SQL има **GROUP BY**-дел за да се одредат групирачките атрибути, кои што исто така мора да постојат и во SELECT-делот

Групирање

- **Прашалник 20.** За секој оддел, дај ги шифрата на одделот, бројот на вработени во тој оддел и нивната просечна плата.

```
Q20: SELECT DNO, COUNT (*), AVG (SALARY)
     FROM EMPLOYEE
     GROUP BY DNO
```

- Во Q20, торките EMPLOYEE се поделени во групи
  - Секоја група има иста вредност за групирачкиот атрибут DNO
  - Функциите COUNT и AVG се применети на секоја група посебно
- SELECT-делот ги вклучува само групирачките атрибути и бараните агрегатни функции

Агрегатни функции и групирање

Пример

Figure 8.6
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

Групирање

- При групирањето може да се користи и услов за спојување
- **Прашалник 21.** За секој проект, дај ги шифрата на проектот, името на проектот и бројот на вработени кои работат на тој проект.

```
Q21: SELECT PNUMBER, PNAME, COUNT(*)
     FROM PROJECT, WORKS_ON
     WHERE PNUMBER=PNO
     GROUP BY PNUMBER, PNAME
```

Во ваквите случаи, групирањето и агрегатните функции се применуваат после спојувањето на релациите

HAVING-делот

- Понекогаш сакаме да ги добиеме вредностите на агрегатните функции само за оние **групи што исполнуваат одредени услови**
- **HAVING**-делот се користи за да се одреди условот за избор на групите (а не за поединечните торки)

HAVING-делот

- **Прашалник 22**. За секој проект *на кој што работат повеќе од двајца вработени*, вратете ги шифрата на проектот, името на проектот, како и бројот на вработени што работат на тој проект.

```
Q22: SELECT PNUMBER, PNAME, COUNT(*)
     FROM PROJECT, WORKS_ON
     WHERE PNUMBER=PNO
     GROUP BY PNUMBER, PNAME
     HAVING COUNT(*) > 2
```

ORDER BY

- **ORDER BY** делот се користи за да се подредат торките во резултатот на прашалникот врз основа на еден или неколку атрибути
  - Предодредениот редослед е “во растечки редослед”
  - Може да го специфицираме клучниот збор **DESC** ако сакаме опаѓачки редослед
  - Клучниот збор **ASC** е за растечки редослед, но тој не мора да се пишува бидејќи се подразбира

ORDER BY

- **Прашалник 28.** Прикажи го списокот на вработените, заедно со проектот на којшто работи секој од нив, подреден според одделот на вработениот, па според презимето на вработениот

```
Q28: SELECT DNAME, LNAME, FNAME, PNAME
     FROM DEPARTMENT, EMPLOYEE, WORKS_ON,
          PROJECT
     WHERE DNUMBER=DNO AND SSN=ESSN
           AND PNO=PNUMBER
     ORDER BY DNAME, LNAME
```

Преглед на SQL SELECT прашалници

- Прашалник во SQL може да има најмногу 6 делови, но само првите два, SELECT и FROM, се задолжителни.
- Деловите од SQL прашалникот се пишуваат во следниов редослед:

```
SELECT      <attribute list>
FROM        <table list>
[WHERE      <condition>]
[GROUP BY   <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY   <attribute list>]
```

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

АЖУРИРАЊЕ НА БАЗИ НА ПОДАТОЦИ

БАЗИ НА ПОДАТОЦИ - предавања

Проф. д-р Слободан Калајџиски

Наредби за ажурирање кај SQL

- Постојат три SQL наредби за промени на содржината на базата на податоци:
  - **INSERT**
  - **DELETE** и
  - **UPDATE**

INSERT

INSERT INTO <table name> **VALUES** (<atributes>)

- Се користи за да додаде **една** или **повеќе** торки во табелата
  - Може да се додаваат целосни торки (со сите вредности на атрибутите), или
  - Делумни торки (со одредени вредности на атрибутите)

INSERT

- Вредностите на атрибутите **мора** да се излистани по истиот редослед како и атрибутите во дефиницијата на табелата (пример, при специфицирањето во **CREATE TABLE** наредбата)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

```
U1: INSERT INTO EMPLOYEE VALUES
    ('Richard','K','Marini',
     '653298653', '30-DEC-52',
     '98 Oak Forest,Katy,TX',
     'M', 37000,'987654321', 4)
```

INSERT

- Исто така може експлицитно да се специфицираат имињата на атрибутите за кои се наведени вредностите во делот VALUES
  - Така, атрибутите со NULL вредности може и да се изостават

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

```
U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
     VALUES ('Richard', 'Marini', '653298653')
```

Останатите атрибути ќе се пополнат или со предефинираните вредности (доколку ги има) или со NULL вредности!

INSERT

- Друга варијанта на INSERT дозволува додавање на повеќе торки одеднаш, кои се резултат на некој прашалник
- **Пример**: Сакаме да создадеме привремена табела која ќе ги содржи називите на одделите заедно со бројот на вработени и вкупните плати за секој оддел.

```
U3A: CREATE TABLE DEPTS_INFO
     (DEPT_NAME VARCHAR(10),
      NO_OF_EMPS INTEGER,
      TOTAL_SAL INTEGER);

U3B: INSERT INTO
     DEPTS_INFO (DEPT_NAME, NO_OF_EMPS, TOTAL_SAL)
     SELECT DNAME, COUNT(*), SUM(SALARY)
     FROM DEPARTMENT, EMPLOYEE
     WHERE DNUMBER=DNO
     GROUP BY DNAME ;
```

INSERT

- **Забелешки за претходниот пример**:
  - Табелата DEPTS_INFO е создадена од прашалникот U3A, и е наполнета со сумарната информација извлечена од базата на податоци со прашалникот U3B
  - Табелата DEPTS_INFO може **да не содржи актуелни податоци**
    - ако по извршувањето на прашалникот U3B се променат некои од записите било во табелата DEPARTMENT било во табелата EMPLOYEE, тогаш DEPTS_INFO ќе содржи невалидни податоци

Овој проблем се разрешува доколку се создаде т.н. **поглед** (**view**)
преку кој ќе се одржува ажурноста на ваквите сумарни прегледни табели

DELETE

DELETE FROM <table> **WHERE** <conditions>

- Ги брише записите од табелата
  - Вклучува WHERE-дел за да се изберат торките што ќе бидат избришани
    - Ако не се наведе WHERE-делот, тогаш сите торки во табелата ќе бидат избришани – табелата ќе стане празна табела
    - Бројот на избришани торки зависи од бројот на торки во релацијата кои го задоволуваат условот во WHERE-делот
  - Мора да се применува референцијалниот интегритет
    - Торки се бришат само од една табела во даден момент (освен ако не е одредена и CASCADE опцијата)

DELETE

- Избриши ги сите вработени со презиме Brown.

```
U4A:  DELETE  FROM  EMPLOYEE
      WHERE  LNAME  =  'Brown'
```

- Избриши ги сите вработени со матичен број 123456789.

```
U4B:  DELETE  FROM  EMPLOYEE
      WHERE  SSN  =  '123456789'
```

DELETE

- Избриши ги сите вработени што работат во одделот Research.

```
U4C: DELETE FROM EMPLOYEE
     WHERE DNO IN
         ( SELECT DNUMBER
           FROM DEPARTMENT
           WHERE DNAME='Research')
```

- Избриши ги сите вработени.

```
U4D: DELETE FROM EMPLOYEE
```

UPDATE

UPDATE <table> **SET** <attributes>
WHERE <conditions>

- Се употребува за да ги промени вредностите на атрибутите на еден или повеќе избрани записи
  - Дополнителен SET-дел одредува кои атрибути ќе бидат променети со новите вредности
  - WHERE-делот одредува кои торки ќе бидат променети
  - Секоја наредба променува торка во истата релација
  - Референцијалниот интегритет се проверува и зачувува

UPDATE

- **Пример**: За проектот со реден број 10, промени ги местоположбата и управувачкиот оддел на 'Bellaire' и 5, соодветно.

```
U5: UPDATE PROJECT
    SET PLOCATION = 'Bellaire', DNUM = 5
    WHERE PNUMBER = 10
```

UPDATE

- **Пример**: Дај им на сите вработени во одделот 'Research' повишување на платата од 10%.

```
U6: UPDATE EMPLOYEE
    SET SALARY = SALARY * 1.1
    WHERE DNO IN (SELECT DNUMBER
                  FROM DEPARTMENT
                  WHERE DNAME='Research')
```

- Во ова барање, променетата вредност за SALARY зависи од оригиналната вредност на SALARY за секоја торка
  - Повикувањето на атрибутот SALARY на **десно** од знакот = се однесува на **старата** вредност на SALARY пред промената
  - Повикувањето на атрибутот SALARY на **лево** од знакот = се однесува на **новата** вредност на SALARY што ќе ја добие по промената

ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

НАПРЕДЕН SQL и ТЕХНИКИ НА ПРОГРАМИРАЊЕ НА БП

БАЗИ НА ПОДАТОЦИ - предавања

Проф. д-р Слободан Калајџиски

Напредна примена на SQL

- Општи ограничувања и тврдења (assertions)
- Активатори (triggers)
- Погледи во SQL (views)
- Програмирање кај БП и вградлив SQL (embedded SQL) *
- Снимени процедури (stored procedures) *

Тврдења (assertions)

CREATE ASSERTION <ime>
CHECK (uslov)

- Покрај ограничувањата одредени со DDL наредбите и кои автоматски се применуваат од страна на СУБП при ажурирања на содржината на базата на податоци, може да се дефинираат и дополнителни ограничувања, наречени **тврдења** (**assertions**)

Тврдења

- **Пример**. Вработените во некој оддел не може да имаат плата поголема од платата на шефот на тој оддел.

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
       (SELECT *
        FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
        WHERE E.SALARY > M.SALARY AND
              E.DNO = D.NUMBER AND D.MGRSSN = M.SSN))
```

- Спецификација на прашалник кој не задоволува некој услов; вклучување во рамките на `NOT EXISTS` изразот
- Резултатот од прашалникот треба да биде празно множество
  - ако резултатот од прашалникот не е празно множество, тогаш тврдењето било повредено (не исполнето)

Активатори (triggers)

CREATE TRIGGER <ime na aktivator>
(**AFTER** | **BEFORE**) <nastani> **ON** <tabela>
[**FOR EACH ROW**][**WHEN** <uslov>]
<akcii>;

- Примена: **мониторирање** на базата на податоци и превземање иницирачки акции кога ќе се случат одредени состојби (услови)
- Синтаксата на активаторите ги вклучува следните делови:
  - **Настан** <nastani>::=<nastan> {**OR** <nastan>}
  - **Услов** <nastan>::=**INSERT**|**DELETE**|**UPDATE** [**OF** <kolona> {, <kolona>}]
  - **Акција** <akcii>::=<PL/SQL block>

Активатори

- **Пример**: Ако сакаме да водиме log за секое бришење на вработен од табелата EMPLOYEE, тогаш треба да се напише следниот активатор:

```
CREATE TRIGGER Emp_Delete
AFTER DELETE ON EMPLOYEE
FOR EACH ROW
INSERT INTO Emps_Deleted_Log
        VALUES (OLD.Fname, OLD.Lname);
```

Активатори

- **Пример**. Да се напише активатор кој ќе ја споредува платата на вработениот со платата на неговиот шеф во текот на операциите внесување (Insert) и промена (Update)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR
       UPDATE OF SALARY, SUPER_SSN ON EMPLOYEE
FOR EACH ROW
WHEN
(NEW.SALARY > (SELECT SALARY
               FROM EMPLOYEE
               WHERE SSN=NEW.SUPER_SSN)
)
INFORM_SUPERVISOR (NEW.SUPER_SSN, NEW.SSN);
```

Ова е некоја снимена процедура (ќе ги учиме подоцна)

Активатори

- **ЗА ДОМА**:
  Додадете атрибут во релацијата DEPARTMENT со име totalSalary во кој ќе се чува вкупната плата од сите вработени во тој оддел.

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date	totalSalary

- Напишете ги активаторите за следните настани:
  - Вметнување на нови торки за вработени
  - Промена на платата на постоечки вработени
  - Прераспределба на постоечки вработени од еден во друг оддел
  - Бришење на торки за вработени

Погледи (views)

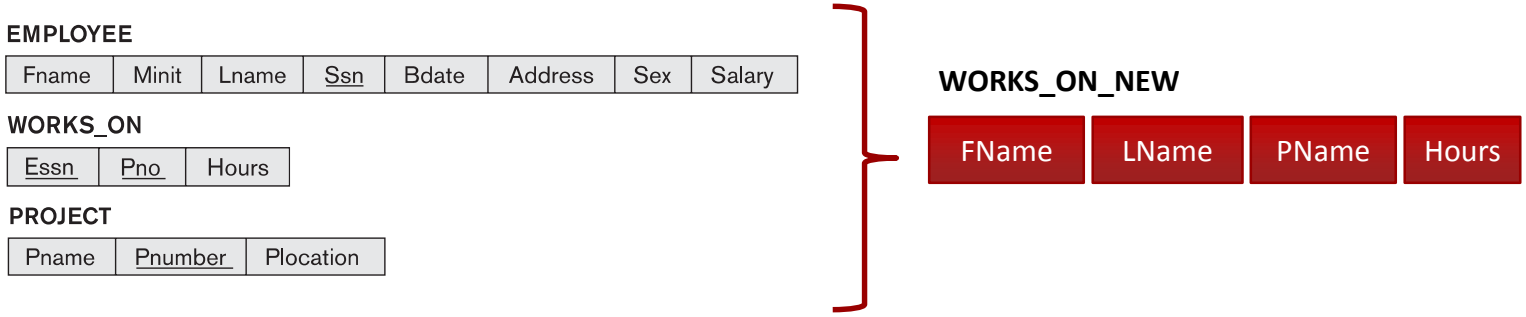
CREATE VIEW <ime> [<koloni>] **AS**
<**SELECT** izraz>

- **Погледот** е “**виртуелна**” табела која се изведува од други табели.
- Секој поглед треба да содржи:
  - име на табелата (погледот)
  - можна листа на имиња на атрибути
    кога имаме аритметички операции или кога сакаме да ги промениме имињата на атрибутите од базните релации од кои ќе се влече погледот
  - прашалник преку кој ќе се специфицира содржината на табелата (погледот)

Погледи

- **Пример**. Специфицирај нова табела WORKS_ON_NEW во која ќе се чуваат податоците за името и презимето на учесникот во проектот, како и името на проектот и реализираните часови на истиот

```
V1: CREATE VIEW WORKS_ON_NEW AS
    SELECT FNAME, LNAME, PNAME, HOURS
    FROM EMPLOYEE, PROJECT, WORKS_ON
    WHERE SSN=ESSN AND PNO=PNUMBER;
```



Погледи

- Веќе креираните погледи со CREATE VIEW наредбата може рамноправно да се употребуваат во SQL прашалниците (како веќе постоечки табели)

```
SELECT FNAME, LNAME
FROM WORKS_ON_NEW
WHERE PNAME='ProductX';
```

- Кога виртуелната табела веќе не ни е потребна, може да ја отстраниме

```
DROP WORKS_ON_NEW;
```

СУБП треба да води сметка за конзистентноста на податоците што се содржани во погледот

Имплементација на погледи

- **Промена на прашалници**:
  - Трансформација на прашалникот од погледот преку прашалник врз основните табели

```
SELECT FNAME, LNAME
FROM WORKS_ON_NEW
WHERE PNAME='ProductX';
```

```
SELECT FNAME, LNAME
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER
      AND PNAME = 'ProductX'
```

Неефикасно за погледи добиени од комплексни прашалници

посебно кога дополнителни прашалници треба да се применат над погледот во релативно краток временски период

РЕШЕНИЕ: Материјализација на поглед

Имплементација на погледи (2)

- **Материјализација на погледи**:
  - Предизвикува физичко креирање и чување на привремени табели (од содржината на погледот)
  - **Претпоставки**:
    - Би следеле и други прашалници врз погледот
  - **Ризици**:
    - Одржување на усогласеноста помеѓу базната табела и погледот во случаи кога базната табела се ажурирала
  - **Стратегија**:
    - Инкрементална промена (ажурирање)

Ажурирање на погледи

- Ажурирање на единечен поглед **без користење агрегатни операции**:
  - Ажурирањето на погледот може да биде проследено со ажурирање на базните табели од кои произлегува погледот
- Ажурирање на погледи кои **вклучуваат соединувања**:
  - Ажурирањето на погледот *може* да биде проследено со ажурирање на базните табели врз кои се заснова погледот
    - **Ова не е секогаш возможно!**
    - **`WITH CHECK OPTION`**: мора да се додаде во дефиницијата на погледот доколку сакаме погледот да може да се ажурира

Погледите добиени преку групирање и агрегатни функции **не може** да се ажурираат!

Ажурирање на погледи

- **Пример**. Промени го насловот на проектот ProductX во ProductY.

```
UV1: UPDATE WORKS_ON_NEW
     SET Pname = 'ProductY'
     WHERE Lname='Smith' AND Fname='John'
           AND Pname='ProductX';
```

```
UPDATE WORKS_ON
SET Pno = ( SELECT Pnumber FROM PROJECT
            WHERE Pname='ProductY')
WHERE Essn IN ( SELECT Ssn FROM EMPLOYEE
                WHERE Lname='Smith' AND Fname='John')
      AND Pno = ( SELECT Pnumber FROM PROJECT
                  WHERE Pname='ProductX')
```

```
UPDATE PROJECT SET Pname = 'ProductY'
WHERE Pname = 'ProductX';
```

Ажурирање на погледи

- **Пример**. Креирај поглед во кој за секој оддел ќе се чуваат името, вкупниот број вработени во одделот и нивната сумарна плата.

```
CREATE VIEW DEPT_INFO(DName, NoEmps, TotalSal) AS
 SELECT Dname, COUNT (*), SUM (Salary)
 FROM DEPARTMENT, EMPLOYEE
 WHERE Dnumber=Dno
 GROUP BY Dname;
```

- Промени ја сумарната плата на вредност 100.000 во одделот ‘Research’.

```
UPDATE DEPT_INFO
SET TotalSal=100000
WHERE Dname='Research';
```

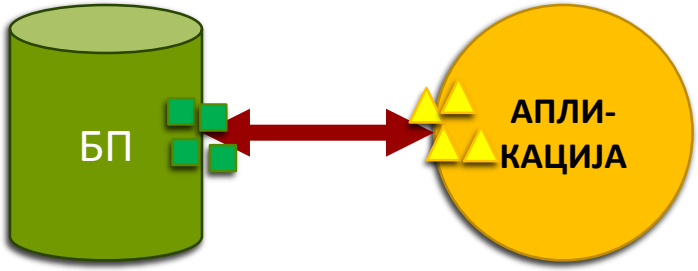
Во овој случај не е возможно да се изврши промена во базните табели од кои произлегол погледот!

Програмирање кај БП

- **Цел:**
  - Пристап до базата на податоци преку апликациска програма (наспроти интерактивните интерфејси)
- **Зошто?**
  - Интерактивниот интерфејс е удобен начин на користење, меѓутоа не е доволен
    - поголемиот дел од операциите во БП се прават преку апликациски програми
    - во последно време операциите се извршуваат преку веб и мобилни апликации

Несогласување во импедансата

- Некомпатибилност помеѓу програмскиот јазик и податочниот модел на базата на податоци, на пример,
  - несогласувања во типовите и некомпатибилност (type mismatch); бара ново поврзување за секој јазик
  - обработка на цело множество **наспроти** обработка на еден по еден запис
    - бара специјални итератори за изминување преку резултатите од прашалникот и манипулација со индивидуалните променливи

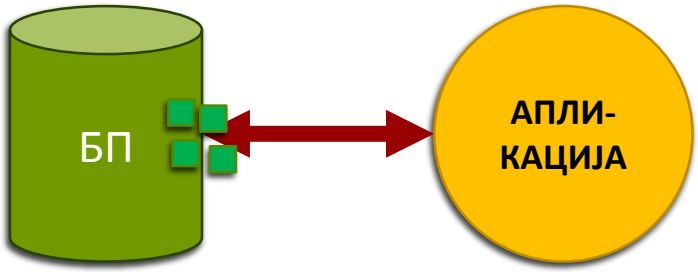


Пристапи за програмирање на БП

- Вгнездени команди:
  - Командите од БП се вгнездуваат во програмскиот јазик кој се користи
- Библиотека од БП функции:
  - Достапна до програмскиот јазик за пристап до базата на податоци; позната како *API* (Application Program Interface)
- Сосем нов целосно-поддржлив јазик:
  - го минимизира несогласувањето на импедансата

Чекори во програмирањето кај БП

1. Клиентската програма **отвара конекција** кон податочниот сервер
2. Клиентската програма **испраќа прашалници и/или барања за модификација** на базата на податоци
3. Кога пристапот до базата на податоци не е повеќе потребен, клиентската **програма ја затвара (терминира) конекцијата** кон БП



SQL наредби за конектирање до БП

- Поврзување со БП (повеќе конекции се можни, меѓутоа само една е активна)

```
CONNECT TO server-name AS connection-name
AUTHORIZATION user-account-info;
```

- Промена од активна конекција во друга конекција

```
SET CONNECTION connection-name;
```

- Раскинување на врската со БП:

```
DISCONNECT connection-name;
```

Вгнезден SQL

- Повеќето SQL изрази може да се вгнездат во најчесто користените програмски јазици како што се COBOL, C, Java...
- Вгнезденият SQL израз се разликува од наредбите на програмскиот јазик преку вгнездување помеѓу наредбите
  - **`EXEC SQL`** или **`EXEC SQL BEGIN`** и
  - **`END-EXEC`** или **`EXEC SQL END`** (или точка-запирка)
- Синтаксата може да варира од јазик до јазик
- **Делените променливи** (кои се користат и во двата јазици) најчесто добиваат префикс две точки (:) во SQL

Вгнезден SQL

- **Итераторот - cursor** е потребен за обработка на повеќе торки
- **`FETCH`** командите го поместуваат курсорот на *следната* торка
- **`CLOSE CURSOR`** покажува дека обработката на резултатите од прашалникот е завршена

ПРИМЕР

- Декларација на променлива во програмскиот јазик C:
  - Променливите внатре во **DECLARE** се заеднички (се делат) и може да се појават во SQL изразите
  - **SQLCODE** се користи за комуникација (errors/exceptions) помеѓу базата на податоци и програмата

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
   varchar dname[16], fname[16], …;
   char ssn[10], bdate[11], …;
   int dno, dnumber, SQLCODE, …;
EXEC SQL END DECLARE SECTION;
```

```
loop = 1;
while (loop)
{  prompt ("Enter SSN: ", ssn);
   EXEC SQL
      select FNAME, LNAME, ADDRESS, SALARY
      into :fname, :lname, :address, :salary
      from EMPLOYEE where SSN == :ssn;
      if (SQLCODE == 0) printf(fname, …);
      else printf("SSN does not exist: ", ssn);
      prompt("More SSN? (1=yes, 0=no): ", loop);
   END-EXEC
}
```

Вгнезден SQL

- **SQLJ**: стандард за вгнездување на SQL во Java
  - SQLJ преведувачот ги конвертира SQL изразите во Java
- **JDBC** (Java Database Connectivity) интерфејс
  - Ги извршува конвертираните SQL изрази во Јава
  - Било која Јава програма со JDBC функции може да пристапи до било кој релациски СУБП кој има JDBC двигател
  - JDBC овозможува програмата да се конектира до повеќе бази на податоци (познати и како *data sources*)
- Одредени класи треба да се импортираат
  - на пример, **`java.sql`**

Вгнезден SQL

- Чекори во JDBC пристапот до БП:
  1. Import на JDBC библиотека **`java.sql.*`**
  2. Вчитај го JDBC двигателот: **`Class.forname("oracle.jdbc.driver.OracleDriver")`**
  3. Дефинирај ги соодветните променливи
  4. Создај конекциски објект (преку **`getConnection`**)
  5. Создај објект за наредби од класата **`Statement`**:
     - PreparedStatment
     - CallableStatement
  6. Идентификувај ги параметрите на наредбата (назначени во дупли наводници)
  7. Поврзи ги параметрите во програмските променливи
  8. Изврши го SQL изразот (референциран од објектот) преку JDBC **`executeQuery`**
  9. Обработи ги резултатите од прашалникот (вратени како објект од типот ResultSet)
     - **`ResultSet`** е 2-димензионална табела

ПРИМЕР

- Вгнезден SQL во Java:

```
ssn = readEntry("Enter a SSN: ");
try {
        #sql{select FNAME, LNAME, ADDRESS, SALARY
        into :fname, :lname, :address, :salary
        from EMPLOYEE where SSN = :ssn};
}
catch (SQLException se) {
        System.out.println("SSN does not exist: ",+ssn);
        return;
}
System.out.println(fname + " " + lname + " " + address + " " + salary);
```

Динамички SQL

- **Цел:**
  - Создавање и извршување на нови (не претходно компајлирани) SQL изрази во време на извршување на програмите (run-time)
    - програмата ги прифаќа SQL изразите преку тастатурата во времето на извршување
    - point-and-click операција ги преведува истите во соодветни SQL прашалници
  - Динамичката промена е релативно едноставна; динамичкиот прашалник може да е многу комплексен
    - бидејќи типот и бројот на извлечените атрибути е непознат за време на компајлирањето

ПРИМЕР

- Динамички SQL во програмскиот јазик C:

```
EXEC SQL BEGIN DECLARE SECTION;
varchar sqlupdatestring[256];
EXEC SQL END DECLARE SECTION;
…
prompt ("Enter update command:", sqlupdatestring);
EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;
EXEC SQL EXECUTE sqlcommand;
```

Снимени процедури

- Постојани процедури/функции (модули) кои локално се чуваат и извршуваат од страна на податочниот сервер
- **Предности:**
  - Ако процедурата е користена од повеќе апликации, тогаш може да се повика од било која од апликациите
    - се намалува редундантноста на кодови
  - Извршувањето на серверската страна ги намалува комуникациските трошоци
  - Подобрување на моќта на моделирање на погледите
- **Недостатоци:**
  - Секој СУБП има сопствена синтакса со што системот ќе биде тешко портабилен

Конструкти за снимени процедури

- Снимена процедура

```
CREATE PROCEDURE procedure-name (params)
    local-declarations
    procedure-body;
```

- Снимена функција

```
CREATE FUNCTION fun-name (params) RETRUNS return-type
    local-declarations
    function-body;
```

- Повикување на процедура или функција

```
CALL procedure-name/fun-name (arguments);
```

ПРИМЕР

- Декларација на снимена функција:

```
CREATE FUNCTION DSize(IN deptno INTEGER) RETURNS VARCHAR [7]

DECLARE No_of_emps INTEGER;

SELECT COUNT(*) INTO No_of_emps
FROM EMPLOYEE WHERE Dno = deptno;

IF No_of_emps > 100 THEN
   RETURN "HUGE”
ELSEIF No_of_emps > 25 THEN
   RETURN "LARGE”
ELSEIF No_of_emps > 10 THEN
   RETURN "MEDIUM”
ELSE RETURN "SMALL”
ENDIF;
```

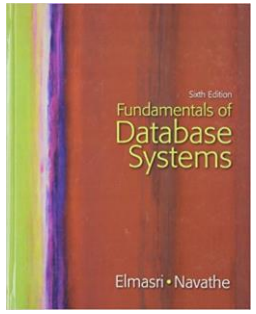
Мал преглед на темата

- Тврдењата обезбедуваат начин за специфицирање дополнителни ограничувања
- Активаторите се тврдења кои дефинираат акции кои автоматски се превземаат во ситуации кога ќе се случи некој настан
- Погледите креираат помошни, привремени (вирутелни) табели
- Базата на податоци може да се пристапи во интерактивен мод, но најчесто податоците во БП се манипулираат преку апликациските програми
- Различни начини на програмирање на БП:
  - Вгнезден SQL
  - Динамички SQL
  - Снимени процедури и функции

Користена литература

- **Глава 8** (241 - 296)
- **Глава 9** (299 - 331)

- **Глава 3** (59 - 85)
- **Глава 4** (87 - 114)
- **Глава 5** (115 - 143)



- **од Глава 6** (287 - ...)
- **до Глава 9** (... - 422)