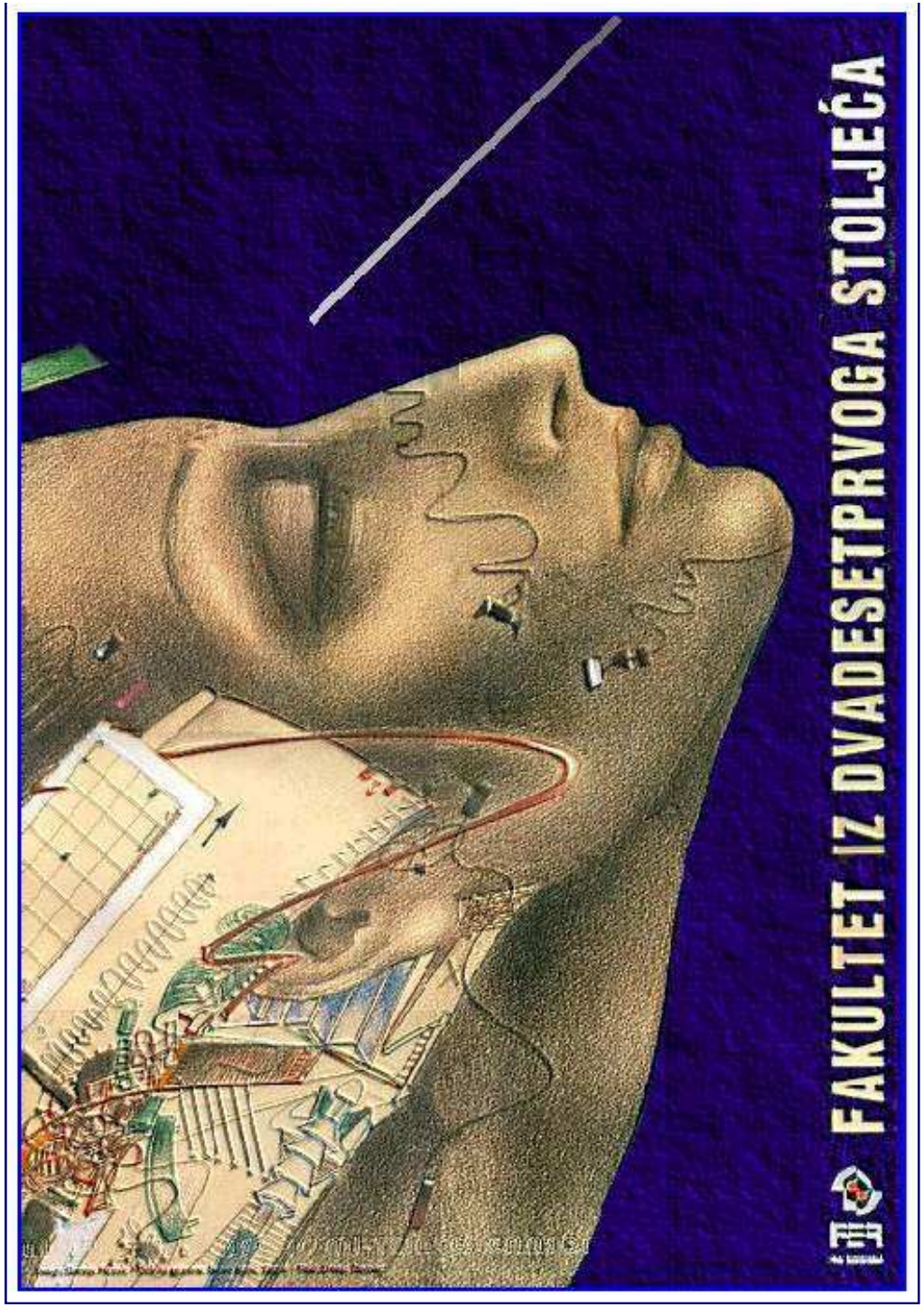
Napredni modeli i baze podataka

Predavanja

5.
Objektno-relacijske baze podataka (2/2)

Listopad 2008.



Pregled

- Objektno-relacijske mogućnosti prema SQL standardu - 2.dio
  - kolekcije, REF tip, DISTINCT tip, LOB tip
- Usporedba relacijskih, objektno-orijentiranih i objektno-relacijskih baza podataka
- Prednosti i nedostaci objektno relacijskih baza podataka
- Objektno-relacijska proširenja u IBM Informix sustavu - 2. dio
  - kolekcije, korisnički definirani tipovi
  - DataBlade moduli
- Literatura

SQL: tipovi podataka

- podjela tipova podataka
  - unaprijed definirani tipovi (*predefined types*)
    - integer, float, character, boolean, datetime, interval ...
  - izgrađeni tipovi (c*onstructed types*)
    - izgrađeni atomarni tipovi (*constructed atomic types*)
      - **referenca (*reference*)**
    - izgrađeni kompozitni tipovi (*constructed composite types*)
      - **kolekcije *(collection):* polje *(array),* multiset**
      - *row*
  - korisnički definirani tipovi (*user-defined types*)
    - ***distinct type***
    - strukturirani tip (*structured type*)

Korišteni primjer

- Primjer: informacijski sustav studentske službe
- Svaki je predmet opisan:
  - šifrom,
  - nazivom,
  - nositeljem,
  - skupom izvođača,
  - skupom studenata koji trenutno slušaju predmet,
  - zavodom kojem predmet pripada (podatak složen od kratice i naziva predmeta)

- kreirana je hijerarhija tablica *osoba*, *nastavnik* i *student*

Kolekcija

- izgrađeni kompozitni tip
- kolekcije vrijednosti homogenog podatkovnog tipa
- strukture podataka kao što je:
  - polje (ARRAY)
    - jednodimenzionalno polje s maksimalnim brojem elemenata
    - podržano SQL:1999 standardom
  - multiskup (MULTISET)
    - neuređena kolekcija koja dozvoljava duplikate
    - podržano SQL:2003 standardom
  - skup (SET)
    - neuređena kolekcija koja ne dozvoljava duplikate
  - lista (LIST)
    - uređena kolekcija koja dozvoljava duplikate

ARRAY (1)

- indeksirana kolekcija elemenata homogenog podatkovnog tipa
- elementima polja moguće je pristupiti pomoću indeksa
- indeks elemenata polja $\in$ [1, maksimalna kardinalnost]

```
CREATE TABLE predmet (
 sifPredmet    INTEGER
 nazPredmet    VARCHAR(250),
 nositelj      INTEGER REFERENCES nastavnik(sifOsoba),
 izvodjaci     INTEGER ARRAY[10] REFERENCES nastavnik(sifOsoba),
 polaznici     INTEGER ARRAY[500] REFERENCES student(sifOsoba),
 zavod         ROW(kratZavod CHAR(8), nazZavod VARCHAR(50)),
 PRIMARY KEY (sifPredmet)
)
```

  - korištenjem polja za pohranu izvođača poznat je i poredak izvođača (na temelju indeksa)

ARRAY (2)

- upisivanje n-torke s atributom tipa ARRAY:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka',
                           123,                    /*nositelj*/
                           ARRAY[123,111,345,24],  /*izvođači*/
                           ARRAY[],                /*polaznici*/
                           ROW('ZPR','Zavod za prim.računarstvo')
);
```

- postavljanje elemenata polja:

```
UPDATE predmet
   SET polaznici = ARRAY[13503, 14111, 9000]
WHERE sifPredmet = 1;
```

- dodjeljivanje vrijednosti elementa na pojedinoj poziciji u polju:

```
UPDATE predmet
   SET polaznici[4] = 14678
WHERE sifPredmet = 1;
```

ARRAY (3)

- funkcija CARDINALITY - vraća trenutni broj elemenata u polju

```
SELECT CARDINALITY(izvodjaci) AS brojIzvodjaca
  FROM predmet
```

brojIzvodjaca
4

```
UPDATE predmet
   SET izvodjaci[6] = 555
WHERE sifPredmet = 1;

SELECT CARDINALITY(izvodjaci) AS brojIzvodjaca
  FROM predmet
 WHERE sifPredmet = 1;

SELECT predmet.izvodjaci[5] AS petiElement
  FROM predmet
WHERE sifPredmet = 1;
```

brojIzvodjaca
6

petiElement
NULL

ARRAY (4)

- UNNEST - obavlja konverziju kolekcije u tablicu
  - kreira se jedna n-torka za svaki element kolekcije
- Primjeri:
  - UNNEST u FROM dijelu SELECT naredbe :

```
SELECT i.sifIzvodjac
  FROM UNNEST(izvodjaci) AS i(sifIzvodjac)
 WHERE sifPredmet = 1
```

sifIzvodjac
123
111
345
24

```
SELECT p.sifPredmet, i.sifIzvodjac
  FROM predmet AS p,
       UNNEST(p.izvodjaci) AS i(sifIzvodjac)
```

→ rezultat upita su svi parovi (*sifPredmet*, *sifIzvodjac*)

```
SELECT p.sifPredmet,
       i.sifIzvodjac, i.pozicija
  FROM predmet AS p,
       UNNEST(p.izvodjaci) WITH ORDINALITY
           AS i(sifIzvodjac, pozicija)
```

→ prethodni upit uz dohvat pozicije izvođača u polju

ARRAY (5)

- UNNEST na mjestu podupita

```
SELECT nazPredmet
  FROM predmet
 WHERE 123 IN (UNNEST(izvodjaci))
```

- dva polja usporedivih tipova smatraju se identičnim akko su jednake kardinalnosti i na istoj poziciji imaju elemente jednakih vrijednosti

MULTISET (1)

- neuređena i neograničena kolekcija elemenata homogenog podatkovnog tipa u kojoj se iste vrijednosti mogu ponavljati
- osigurani su operatori za konverziju multiseta u tablicu (UNNEST)

```
CREATE TABLE predmet (
 sifPredmet    INTEGER
 nazPredmet    VARCHAR(250),
 nositelj      INTEGER REFERENCES nastavnik(sifOsoba),
 izvodjaci     INTEGER MULTISET REFERENCES nastavnik(sifOsoba),
 polaznici     INTEGER ARRAY[1000] REFERENCES student(sifOsoba),
 zavod         ROW(kratZavod CHAR(8), nazZavod VARCHAR(50)),
 PRIMARY KEY (sifPredmet)
)
```

  - korištenjem multiseta za pohranu izvođača, nije poznat njihov poredak

MULTISET (2)

- upisivanje n-torke s atributom tipa MULTISET:

```
INSERT INTO predmet VALUES(1, 'Napredni modeli i baze podataka',
                           123,                    /*nositelj*/
                           MULTISET[123,111,345,24],  /*izvođači*/
                           ARRAY[13503, 14111, 9000], /*polaznici*/
                           ROW('ZPR','Zavod za prim.računarstvo')
);
```

- dva multiseta, *A* i *B*, usporedivih tipova elemenata, smatraju se identičnim akko imaju jednaku kardinalnost i za svaki je element *x* iz *A*, broj elemenata iz *A* koji su identični elementu *x*, uključujući sam *x*, jednak broju elemenata iz *B* koji su jednaki elementu *x*.

MULTISET (3)

- operacije nad multisetom:
  - SET funkcija - uklanja duplikate iz multiseta
  - CARDINALITY funkcija – vraća trenutni broj elemenata
  - ELEMENT funkcija – vraća element multiseta, ukoliko je to jedini element (ili NULL ako multiset nema elemenata)
  - MULTISET UNION – unija dva multiseta; uz ključnu riječ ALL zadržavaju se duplikati, DISTINCT izbacuje duplikate
  - MULTISET INTERSECT – presjek dva multiseta; DISTINCT izbacuje duplikate; ALL u rezultat smješta onoliko instanci svake vrijednosti koliki je minimalan broj instanci te vrijednosti u oba multiseta
  - MULTISET EXCEPT – razlika dva multiseta; DISTINCT izbacuje duplikate; ALL u rezultat smješta broj instanci vrijednosti jednak broju instanci vrijednosti u prvom multisetu umanjen za broj instanci u drugom multisetu

MULTISET (4)

- agregatne funkcije nad multisetovima:
  - COLLECT – kreira multiset iz vrijednosti svake n-torke iz grupe
  - FUSION – kreira multiset koji se sastoji od unije elemenata multiseta iz svih n-torki iz grupe
  - INTERSECTION – kreira multiset koji se sastoji od presjeka elemenata multiseta iz svih n-torki iz grupe

$\pi_{\text{sifPredmet, izvodjaci}}(\textit{predmet})$

sifPredmet	izvodjaci
2	MULTISET [145,245]
3	MULTISET [204,145,265]
4	MULTISET [284,145]
5	NULL

```
SELECT COLLECT(sifPredmet) AS predmeti,
       FUSION(izvodjaci) AS unija,
       INTERSECTION(izvodjaci) AS presjek
  FROM predmet
```

predmeti	unija	presjek
MULTISET [2,3,4,5]	MULTISET [145,145,145,245,204,265,284]	MULTISET [145]

MULTISET (5)

- dodatni predikati koje je moguće koristiti s multisetovima:
  - predikat usporedbe (samo jednakost i nejednakost)
  - MEMBER - ispituje je li navedena vrijednost član multiseta

    value1 [NOT] MEMBER [OF] *multiset_value2*
  - SUBMULTISET – ispituje je li jedan multiset podskup drugog

    multiset_value1 [NOT] SUBMULTISET [OF] *multiset_value2*
  - IS [NOT] A SET – provjerava postoje li u multisetu duplikati

    multiset_value IS [NOT] A SET

Deugnježđivanje i ugnježđivanje (1)

- Primjer: informacijski sustav knjižnice

knjiga

sifra	naslov	autori	kljucneRijeci
1	Compilers	{Smith, Jones}	{parsing, analysis}
2	Networks	{Jones, Frick}	{Internet, Web}

knjiga1NF

sifra	naslov	autor	kljucnaRijec
1	Compilers	Smith	parsing
1	Compilers	Jones	parsing
1	Compilers	Smith	analysis
1	Compilers	Jones	analysis
2	Networks	Jones	Internet
2	Networks	Frick	Internet
2	Networks	Jones	Web
2	Networks	Frick	Web

Deugnježđivanje i ugnježđivanje (2)

- normalizirani oblik:

knjigaN

sifra	naslov
1	Compilers
2	Networks

knjigaAutor

sifra	autor
1	Smith
1	Jones
2	Jones
2	Frick

knjigaKR

sifra	kljucnaRijec
1	parsing
1	analysis
2	Internet
3	Web

Deugnježđivanje i ugnježđivanje (3)

- deugnježđivanje (*unnesting*) - transformacija ugnježđene relacije u oblik s manje (ili bez) složenih atributa

- primjer: deugnježđivanje relacije *knjiga*

```
SELECT sifra, naslov,
       a.autor,
       r.kljucnaRijec
  FROM knjiga AS k,
       UNNEST(k.autori) AS a(autor),
       UNNEST(k.kljucneRijeci) AS r(kljucnaRijec)
```

Deugnježđivanje i ugnježđivanje (4)

- ugnježđivanje (*nesting*) - transformacija neugnježđene 1NF relacije u ugnježđenu relaciju
  - grupiranjem - korištenje funkcije COLLECT

```
SELECT sifra, naslov,
       COLLECT(autor) AS autori,
       COLLECT(kljucnaRijec) AS kljucneRijeci
  FROM knjiga1NF
GROUP BY sifra, naslov
```

  - podupitima u SELECT klauzuli

```
SELECT sifra, naslov,
       ARRAY(SELECT autor FROM knjigaAutor AS a
              WHERE a.sifra = k.sifra) AS autori,
       MULTISET(SELECT kljucnaRijec FROM knjigaKR AS r
                 WHERE k.sifra = r.sifra) AS kljucneRijeci
  FROM knjigaN AS k
```

REF tip (1)

- tip čija vrijednost pokazuje na lokaciju na kojoj je pohranjena vrijednost referenciranog tipa, tj.
  - ako je *T* tip, tada je REF *T* pokazivač na objekt tipa *T*
- može pokazivati samo na n-torke tipizirane tablice
- SQL sintaksa za definiranje REF tipa:

```
<reference type> ::= REF (<referenced type>) [ <scope clause> ]
           [ARRAY [<unsigned integer>]] /* [] je dio sintakse */
           [ <reference scope check> ]
<referenced type> ::= <user-defined type name>
<scope clause> ::= SCOPE <table name>
<reference scope check> ::= REFERENCES ARE [ NOT ] CHECKED
                            [ ON DELETE <action> ]
```

- koristi se za definiranje:
  - atributa u relaciji
  - atributa strukturiranog tipa
    - modelira povezanost objekata u tipiziranim tablicama, temeljenu na identitetu objekta, umjesto korištenja stranih ključeva
  - varijable ili parametra

REF tip (2)

```
CREATE TYPE predmetT (
   sifPredmet  INTEGER,
   nazPredmet  VARCHAR(250),
   nositelj    REF (nastavnikT),
   izvodjaci   REF (nastavnikT) MULTISET,
   polaznici   REF (studentT) ARRAY[1000],
   zavod       zavodT)
   INSTANTIABLE NOT FINAL
   REF IS SYSTEM GENERATED
```

```
CREATE TABLE predmet OF predmetT
(PRIMARY KEY (sifPredmet)
 REF IS predmetID SYSTEM GENERATED)
```

→ veza s objektima tipa *nastavnikT* i *studentT* ostvarena je korištenjem REF tipa

predmet

sifPredmet	nazPredmet	nositelj	...
1	Nap...		

prema *nastavnikT* objektu

REF tip (3)

- upisivanje n-torke s NULL vrijednosti atributa tipa REF:

```
INSERT INTO PREDMET (sifPredmet, nazPredmet, nositelj)
       VALUES (1, 'Napredni modeli i baze podataka', NULL);
```

- dodjeljivanje vrijednosti atributu definiranom kao REF tip:

```
UPDATE predmet
   SET nositelj = (SELECT osobaID
                     FROM osoba
                    WHERE sifOsoba = 123)
 WHERE nazPredmet = 'Napredni modeli i baze podataka'
```

  - pri dodjeljivanju vrijednosti atributa REF tipa koristi se ime dodatnog atributa (identifikatora objekta) tipizirane tablice u kojoj se nalazi objekt na kojeg REF tip pokazuje

REF tip (4)

- doseg reference:
  - određuje na koje se objekte može referencirati
  - definiran SCOPE klauzulom u specifikaciji REF tipa u CREATE TYPE ili CREATE TABLE naredbi
    - navedena SCOPE klauzula - dozvoljeno referenciranje samo na objekte navedene tipizirane tablice
    - nije navedena SCOPE klauzula - dozvoljeno referenciranje na objekte iz bilo koje tablice temeljene na navedenom tipu
- kontrola dosega reference:
  - klauzula REFERENCED ARE CHECKED u specifikaciji REF tipa - nisu dozvoljene neispravne vrijednosti reference
- akcije pri pokušaju narušavanja ograničenja dosega:
  - klauzula ON DELETE u specifikaciji REF tipa
    - NO ACTION, SET NULL, SET DEFAULT, CASCADE

REF tip (5)

- primjer specificiranja dosega reference, kontrole dosega reference i akcije pri pokušaju narušavanja ograničenja dosega
  - u CREATE TYPE naredbi:

```
CREATE TYPE predmetT (
 ...
 nositelj REF (nastavnikT) SCOPE nastavnik
                           REFERENCES ARE CHECKED
                           ON DELETE SET NULL
 ...)
```

  - u CREATE TABLE naredbi:

```
CREATE TABLE predmet OF predmetT
                  (nositelj WITH OPTIONS SCOPE nastavnik
                                REFERENCES ARE CHECKED
                                ON DELETE SET NULL)
```

REF tip (6)

- postavljanje upita nad tablicama s atributima REF tipa
  - korisnike zanimaju vrijednosti atributa referencirane n-torke
  - → (*dereference* operator) - omogućava "prijelaz" do referencirane n-torke
    - do referencirane n-torke dolazi se *implicitnim spajanjem*
    - upit postavljen na jednoj tablici može vratiti vrijednost atributa referencirane n-torke iz druge tablice

```
SELECT nositelj → prezime
  FROM predmet
 WHERE nazPredmet = 'Napredni modeli i baze podataka';
```

  - funkcija *deref* - vraća referencirane n-torke
    - tip dohvaćenih n-torki odgovara tipu na temelju kojeg je definirana tipizirana tablica u kojoj se nalaze

```
SELECT deref(nositelj)
  FROM predmet p
 WHERE p.zavod.kratZavod = 'ZPR';
```

→ rezultat upita je tablica s jednom kolonom tipa *nastavnikT*

DISTINCT tip (1)

- korisnički definirani tip
- temeljen na atomarnom tipu
- dodjeljuje posebno značenje postojećem atomarnom tipu
  - sprječava miješanje logički nekompatibilnih vrijednosti (npr. usporedbu godina s težinom)
- nad DISTINCT tipovima moguće je definirati metode
- nije moguća hijerarhija

```
CRETE TYPE godineT AS INTEGER FINAL;
CRETE TYPE tezinaT AS INTEGER FINAL;
CREATE TABLE osoba (
        sifOsoba    INTEGER
        prezime     VARCHAR(25),
        godineOsoba godineT
        tezinaOsoba tezinaT
        PRIMARY KEY sifOsoba);
```

- FINAL obavezno navesti prema trenutnom SQL standardu

DISTINCT tip (2)

- uspoređivanje vrijednosti istog DISTINCT tipa:

```
CRETE TYPE godineT ASELECT o1.sifOsoba
  FROM osoba o1, osoba o2
 WHERE o2.sifOsoba = 123 AND o1.godineOsoba < o2.godineOsoba;
```

- uspoređivanje vrijednosti DISTINCT tipa s atomarnim tipom na kojem je temeljen te različitih DISTINCT tipa temeljenih na istom atomarnom tipu – nije dozvoljeno:

```
SELECT sifOsoba FROM osoba
 WHERE (godineOsoba * 2) < tezinaOsoba;
```

→ ERROR

- usporedba DISTINCT tipa i atomarnog tipa na kojem je temeljen dozvoljena uz korištenje funkcije CAST:

```
SELECT sifOsoba FROM osoba
 WHERE CAST(godineOsoba AS INTEGER) * 2 < CAST(tezinaOsoba AS INTEGER);
```

LOB tip (*Large Object Type*)

- omogućava pohranu velikih vrijednosti (fotografije osoba, medicinske slike u visokoj rezoluciji, video zapisi)
- predefinirani tip podatka (*String*)
  - *Character Large Object* (CLOB)
    - sadrže "tekstualne" podatke (*printable characters*, *tabs*, *newlines*, *newpages*)
    - dozvoljene neke operacije nad znakovnim nizovima (npr. konkatenacija (||), funkcije SUBSTRING, UPPER, TRIM ...), usporedba (=, <>)
  - *Binary Large Object* (BLOB) - bilo kakav niz binarnih podataka
- ograničeno korištenje
  - ne može biti dio ORDER BY, GROUP BY klauzule, dio UNIQUE ograničenja, dio stranog ključa ...

Pohranjene procedure

- pohranjene procedure *(SQL-invoked routines*): procedure, funkcije i metode
- mogu biti pozvane iz SQL koda
- **SQL rutina** - napisana u SQL-u
- **eksterna rutina** (*external routines*)
  - napisana u eksternom programskom jeziku (npr. Java, C++)
  - poziva se na isti način kao SQL procedura ili funkcija
  - omogućava korištenje postojećeg koda
  - portabilnost između različitih sustava baza podataka
  - problem nepodudaranja tipova podataka (npr. time, blob, …)
  - primjer eksterne rutine:

```
CREATE FUNCTION malaSlika (IN ulaznaSlika SlikaT) RETURNS BOOLEAN
EXTERNAL NAME '/usr/bin/slike/malaSlika'
LANGUAGE C ...
```

Vrste dozvola potrebnih za rad s proširenjima

- potrebne dozvole prilikom kreiranja strukturiranog tipa:
  - USAGE dozvola nad svim korisnički definiranim tipovima koji se koriste u definiciji tipa
  - UNDER dozvola za nadređeni tip - kod kreiranja tipa koji sudjeluje u hijerarhiji tipova
- neke dozvole potrebne za rad s objektima koji koriste strukturirane tipove:
  - SELECT dozvola za kolone [virtualne] relacije
    - ako se koriste metode u kojima se pristupa vrijednosti kolone koja je strukturiranog tipa
  - SELECT dozvola za strukturirane tipove
    - u slučaju korištenja n-torke iz tipizirane tablice u kombinaciji s referencom na tu n-torku (REF tipom)
  - EXECUTE dozvola za metode koje se pozivaju

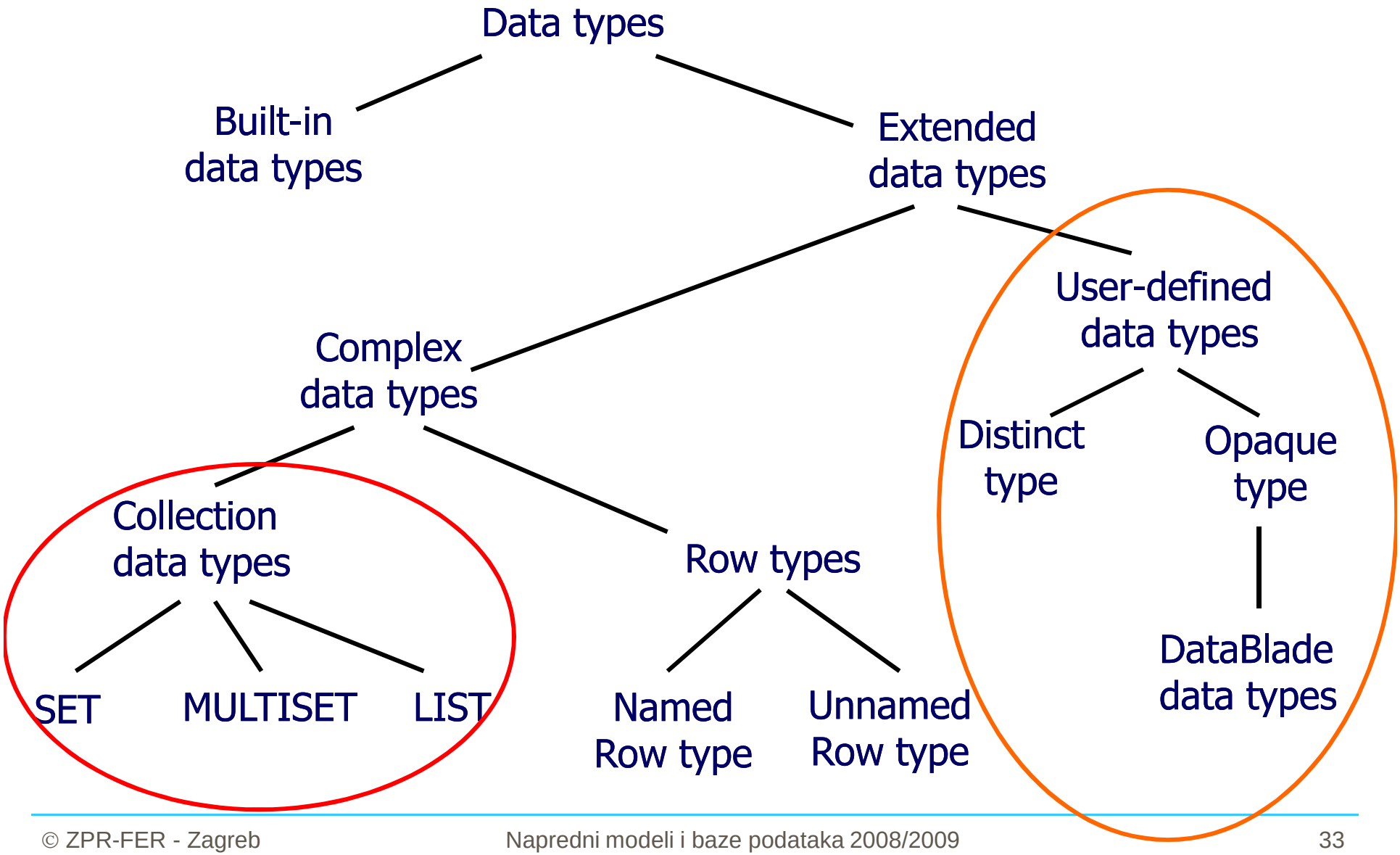
Usporedba relacijskih, objektnih i objektno-relacijskih baza podataka

- **Relacijske**
  - jednostavni tipovi podataka, moćan upitni jezik, visoki stupanj zaštite
- **Objektne**
  - složeni tipovi podataka, integriranost s programskim jezikom, efikasnost
- **Objektno-relacijske**
  - složeni tipovi podataka, moćan upitni jezik, visoki stupanj zaštite

ORDBMS prednosti i nedostaci

- prednosti
  - mogućnost ponovnog korištenja i dijeljenja funkcionalnosti
    - proširenjem poslužitelja SUBP-a funkcionalnost dostupna svima
    - zadržane sve mogućnosti relacijskih baza podataka
      - proširenim relacijskim pristupom očuvana znanja i iskustva uložena u razvoj aplikacija temeljenih na relacijskom modelu
- nedostaci
  - složenost
  - nezadovoljstvo pobornika relacijskog modela
    - izgubljena osnovna jednostavnost i čistoća relacijskog modela
    - performance proširenja lošije u odnosu na trenutnu relacijsku tehnologiju
  - nezadovoljstvo pobornika objektno-orijentiranog modela
    - nezadovoljstvo korištenom terminologijom i pristupom objektnim konceptima

IBM Informix implementacija – 2. dio



Kolekcije: *SET*

- grupa elemenata (jednakog tipa) u kojoj nije dopušteno ponavljanje istih vrijednosti. Redoslijed elemenata nije bitan.

```
CREATE TABLE prisutni (
   datum     DATE
 , studenti  SET(ROW(ime CHAR(20)
                   , prez CHAR(20)) NOT NULL) NOT NULL
);
```

```
INSERT INTO prisutni VALUES (
   '09.10.2008'
 , SET{ ROW('Ivan', 'Novak')
      , ROW('Ana', 'Kolar')
      , ROW('Zrinka', 'Horvat')
      , ROW('Ana', 'Kolar')
      }
);
```

NOT NULL uz element skupa je obavezan - element skupa ne smije biti NULL

Ubačena su 4 elementa, ali je jedan duplikat.

U konačnom rezultatu u skupu ima samo 3 elementa

```
SELECT *
FROM prisutni
```

datum	studenti
09.10.2008	SET{ROW('Ivan','Novak'), ... ROW('Zrinka','Horvat')}

Kolekcije: *MULTISET*

- grupa elemenata (jednakog tipa) u kojoj se iste vrijednosti mogu ponavljati. Redoslijed elemenata nije bitan.

```
CREATE TABLE ishodPokusaBaciOdjednom5Novcica (
   datum     DATE
 , rezultat  MULTISET(CHAR(5) NOT NULL) NOT NULL
);
```

  - NOT NULL uz definiciju tipa elementa multi-skupa je obavezan - element multiseta ne smije biti NULL
  - drugi NOT NULL ima uobičajeno značenje: sustav neće dopustiti unos n-torke za koju atribut *rezultat* ima nepoznatu vrijednost

```
INSERT INTO ishodPokusaBaciOdjednom5Novcica VALUES (
   '09.10.2008'
  , MULTISET{ 'glava', 'glava', 'pismo', 'pismo', 'pismo'}
);
```

```
SELECT * FROM ishodPokusaBaciOdjednom5Novcica
```

datum	rezultat
09.10.2008	MULTISET { 'glava', 'glava','pismo', 'pismo', 'pismo'}

Kolekcije: *LIST*

- grupa elemenata (jednakog tipa) u kojoj se iste vrijednosti mogu ponavljati. Redoslijed elemenata je bitan.

```
CREATE TABLE ishodPokusaBaciNovcic5Puta (
   datum     DATE
 , rezultat  LIST(CHAR(5) NOT NULL)NOT NULL
);
```

  - NOT NULL uz definiciju tipa elementa liste je obavezan - element liste ne smije biti NULL.

```
INSERT INTO ishodPokusaBaciNovcic5Puta VALUES (
   '09.10.2008'
 , LIST{ 'pismo', 'glava', 'pismo', 'pismo', 'glava'}
);
```

```
SELECT * FROM ishodPokusaBaciNovcic5Puta
```

datum	rezultat
09.10.2008	LIST {'pismo', 'glava', 'pismo', 'pismo', 'glava'}

IN operator i kolekcije

- IN operator ispituje nalazi li se neki izraz u objektu tipa kolekcije
- izraz po tipu mora odgovarati elementu kolekcije

```
CREATE TABLE turist1(jmbg CHAR(13)
                   , posjetioDrzave MULTISET(CHAR(20) NOT NULL));
```

```
SELECT * FROM turist1
   WHERE 'Argentina' IN posjetioDrzave
```

```
CREATE TABLE turist2(
     jmbg CHAR(13)
   , posjetioDrzave MULTISET (ROW(oznDrzava CHAR(2),
                                  nazDrzava CHAR(20)) NOT NULL));
```

```
SELECT turist2.jmbg FROM turist2
   WHERE ROW('AR', 'Argentina') IN turist2.posjetioDrzave
```

CARDINALITY funkcija

- broj elemenata u kolekcijama (bez obzira na tip elementa)

```
SELECT jmbg, CARDINALITY(posjetioDrzave) FROM turist1
```

jmbg	(expression)
123	2

Collection Subquery

- pretvara rezultat upita ("virtualnu" relaciju) u MULTISET. Ovdje se pod pojmom virtualne relacije ne misli na pogled - view, nego rezultat neke SELECT naredbe
- postoje dva oblika *collection subquery*
  1. rezultat je MULTISET čiji su elementi tipa *expr*

```
MULTISET(SELECT ITEM expr FROM ...)
```

Primijetite: samo jedan izraz (*expr*)

  2. rezultat je MULTISET čiji su elementi tipa *neimenovani* ROW, čiji su tipovi atributa određeni s tipovima *expr1*, *expr2*, ...

```
MULTISET(SELECT expr1, expr2, ... FROM ...)
```

Primijetite: jedan ili više izraza (*expr1*, *expr2*, ...)

Collection Subquery: 1. oblik

```
CREATE TABLE drzava1 (nazDrzava CHAR(20));
CREATE TABLE turist1 (jmbg           CHAR(13)
                    , posjetioDrzave MULTISET(CHAR(20) NOT NULL));
```

```
INSERT INTO drzava1 VALUES ('Argentina');
INSERT INTO drzava1 VALUES ('Austrija');
INSERT INTO drzava1 VALUES ('Brazil');
```

```
INSERT INTO turist1 VALUES ('123'
  , MULTISET(SELECT ITEM nazDrzava
               FROM drzava1
              WHERE nazDrzava MATCHES 'A*'));
```

```
SELECT * FROM turist1
```

- u relaciji *turist1*: atribut *posjetioDrzave* je **kolekcija nizova znakova**!

jmbg	123
(posjetioDrzave)	MULTISET {Argentina', 'Austrija'}

Collection Subquery: 2. oblik

```
CREATE TABLE drzava2 (oznDrzava CHAR(2), nazDrzava CHAR(20));
```

```
CREATE TABLE turist2(jmbg CHAR(13)
                   , posjetioDrzave
                          MULTISET (ROW(oznDrzava CHAR(2),
                                        nazDrzava CHAR(20)
                                   ) NOT NULL));
```

```
INSERT INTO drzava2 VALUES ('AR' ,'Argentina');
INSERT INTO drzava2 VALUES ('A', 'Austrija');
INSERT INTO drzava2 VALUES ('BR', 'Brazil');
```

```
INSERT INTO turist2 VALUES (
    '123'
  , MULTISET(SELECT oznDrzava, nazDrzava
               FROM drzava2
               WHERE nazDrzava MATCHES 'A*'));
```

- u relaciji *turist2*: atribut *posjetioDrzave* je kolekcija **neimenovanih row tipova**!

```
SELECT * FROM turist2
```

jmbg	123
(posjetioDrzave)	MULTISET{ROW('AR','Argentina'),ROW('A','Austrija')}

Collection-derived Table (1)

- pretvara objekt tipa *kolekcija* u "virtualnu" relaciju (ne misli se na pogled)

```
TABLE(collection_expression)
    [AS aliasName]
    [(columnName, ...)]
```

```
MULTISET(CHAR(20) NOT NULL)
```

```
SELECT * FROM TABLE((SELECT posjetioDrzave FROM turist1
                     WHERE jmbg = '123'));
```

unnamed_col_1
Argentina
Austrija

sustav ne može odrediti ime atributa u "virtualnoj" relaciji - ime atributa se "generira"

```
SELECT drz.naz FROM TABLE((SELECT posjetioDrzave FROM turist1
                           WHERE jmbg = '123')) AS drz(naz);
```

naz
Argentina
Austrija

ime virtualne relacije i imena atributa u virtualnoj relaciji određena su s AS drz(naz)

Collection-derived Table (2)

```
MULTISET (ROW(oznDrzava CHAR(2),
              nazDrzava CHAR(20)
          ) NOT NULL)
```

```
SELECT * FROM TABLE((SELECT posjetioDrzave FROM turist2
                     WHERE jmbg = '123'));
```

oznDrzava	nazDrzava
AR	Argentina
A	Austrija

imena atributa "virtualne" relacije određuju se na temelju imena atributa neimenovanog ROW tipa kojim je definiran tip elemenata multiseta

```
SELECT drz.naz, drz.ozn
    FROM TABLE((SELECT posjetioDrzave FROM turist2
                WHERE jmbg = '123')) AS drz(ozn, naz);
```

naz	ozn
Argentina	AR
Austrija	A

ime "virtualne" relacije i imena atributa u virtualnoj relaciji određena su s AS drz(ozn, naz)

```
SELECT drz1.nazDrzava, drz2.nazDrzava
  FROM TABLE((SELECT posjetioDrzave FROM turist2)) AS drz1
     , TABLE((SELECT posjetioDrzave FROM turist2)) AS drz2
    WHERE drz1.nazDrzava <> drz2.nazDrzava
```

DISTINCT TYPE

- definira se na temelju postojećeg ugrađenog (*built-in*), OPAQUE tipa, imenovanog ROW tipa ili nekog drugog DISTINCT tipa
- moguće je definirati funkcije, agregatne funkcije, operatore i CAST operatore koji će biti primjenjivi na tom tipu podatka
- ima jednaku fizičku reprezentaciju kao i tip na temelju kojeg se definira, ali je "različit" od tog tipa (od tuda dolazi i ime: DISTINCT)
- definira se pomoću naredbe:

```
CREATE DISTINCT TYPE distinctTypeName
                     AS sourceTypeName
```

- unatoč tome što je fizička reprezentacija DISTINCT tipa i tipa iz kojeg je izveden jednaka, usporedba njihovih vrijednosti nije moguća bez navođenja CAST operatora
- sustav prilikom definicije novog DISTINCT tipa automatski generira CAST operatore za "pretvorbu" novog DISTINCT tipa u tip iz kojeg je izveden i obratno

DISTINCT TYPE - primjer

- naredbe za kreiranja dva DISTINCT tipa (*tCelsTemp* i *tFahrTemp*):

```
CREATE DISTINCT TYPE tCelsTemp AS DECIMAL(3,1);
CREATE DISTINCT TYPE tFahrTemp AS DECIMAL(3,1);
```

  - automatski su kreirani i CAST operatori za pretvaranje *tCelsTemp* u DECIMAL(3,1) i obratno, te *tFahrTemp* u DECIMAL(3,1) i obratno.
  - nema mogućnosti za usporedbu temperatura izraženih u stupnjevima celzijusa s temperaturama izraženim u stupnjevima farenhajta

```
CREATE TABLE usTemperature (
  datum   DATE
, mjesto  CHAR(20)
, temp    tFahrTemp);
```

```
CREATE TABLE euroTemperature (
  datum   DATE
, mjesto  CHAR(20)
, temp    tCelsTemp);
```

```
INSERT INTO usTemperature VALUES ('3.1.2003', 'Anchorage'
                                  , -6.0::tFahrTemp);
INSERT INTO euroTemperature VALUES ('3.1.2003', 'Rovaniemi'
                                  , -20.0::tCelsTemp);
SELECT AVG(temp::DECIMAL(3,1)) FROM usTemperature
```

  - CAST operator kojeg je kreirao sustav omogućio je da se DECIMAL tip podatka "pretvori" u *tFahrTemp* i obratno (slično i za *tCelsTemp*)

Korisnički definiran CAST

```
SELECT * FROM usTemperature WHERE temp < ALL (
   SELECT temp FROM euroTemperature)
```

```
SQL ERROR(-674): Routine (equal) cannot be resolved
```

- da bi se omogućila usporedba vrijednosti različitih DISTINCT tipova, potrebno je definirati CAST operator kojim će se vrijednost jednog tipa "pretvoriti" u vrijednost drugog tipa

```
CREATE FUNCTION cTof (cTemp tCelsTemp) RETURNING tFahrTemp;
   RETURN (9/5*cTemp::DECIMAL(3,1)+32)::tFahrTemp;
END FUNCTION;
CREATE EXPLICIT CAST (tCelsTemp AS tFahrTemp WITH cTof);
```

```
SELECT * FROM usTemperature WHERE temp < ALL (
   SELECT temp::tFahrTemp FROM euroTemperature);
SELECT temp::tFahrTemp tempIzrazenaUfahr FROM euroTemperature;
```

- na sličan način može se kreirati i CAST operator za "pretvaranje" vrijednosti tipa *tFahrTemp* u *tCelsTemp*. Time bi omogućili upite oblika:

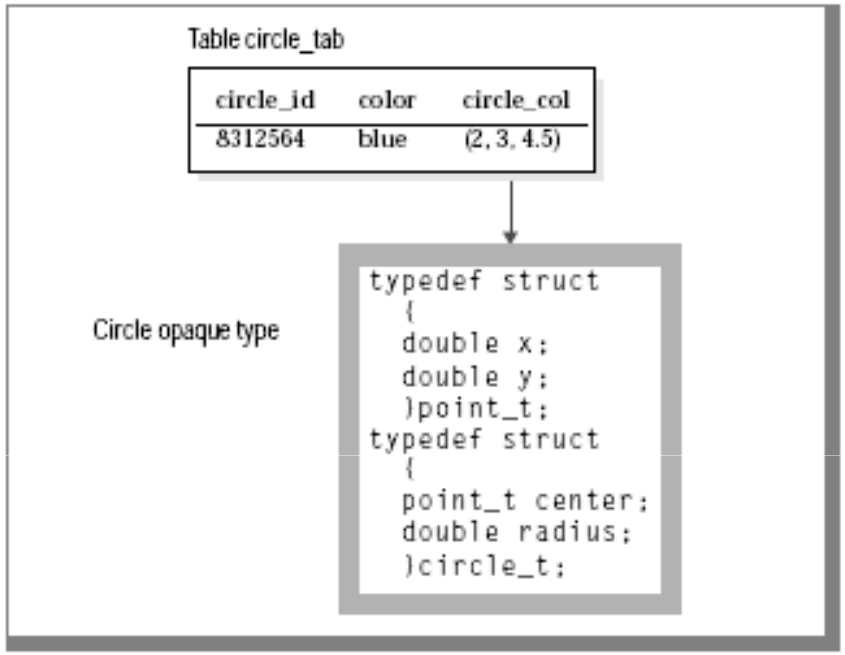
```
SELECT * FROM usTemperature WHERE temp::tCelsTemp < ALL (
   SELECT temp FROM euroTemperature)
```

OPAQUE TYPE (1)

- korisnički definirani, učahureni tip podatka
- potrebno je, osim funkcija, agregatnih funkcija, operatora i CAST operatora koji će biti na tom tipu primjenjivi, **definirati fizički način pohrane tog tipa**
  - primjer: ukoliko bi trebalo kreirati tip podatka "matrica" - ne postoji ugrađeni tip podatka čija bi se struktura mogla elegantno upotrijebiti za fizičku reprezentaciju
- Opaque tipovi se u IBM Informix sustavu mogu definirati pomoću jezika C ili Java
- mogućnost definiranja ovakvih tipova podataka se intenzivno koristi u ***Datablade* modulima**

OPAQUE TYPE (2)

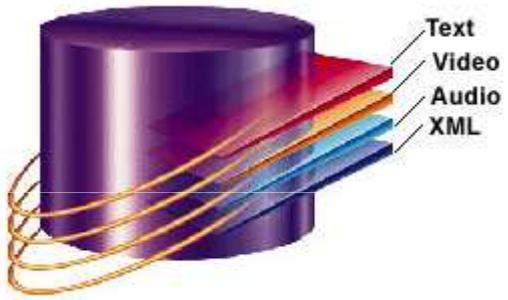
- primjer OPAQUE tipa:



  - atribut *circle_col* relacije *circle_tab* je OPAQUE tipa *circle*

DataBlade moduli (1)

- programski paketi koji proširuju funkcionalnost IBM Informix SUBP-a; rješenje za probleme čija domena nije dobro podržana standardnim SQL-om
- značajke:
  - mogu ih kreirati i korisnici
  - integriraju se u sustav
  - dinamički se dodaju i uklanjaju
  - neograničen broj uključenih modula
  - jednostavnije aplikacije



- potrebne su:
  - strukture koje podržavaju nove tipove podatke
  - rutine za unos i prikaz sadržaja
  - funkcije za obradu novih tipova podataka

DataBlade moduli (2)

- postojeći DataBlade moduli:
  - Geodetic
  - Large Object Locator
  - TimeSeries
  - Video Foundation
  - Web
  - Excalibur Text Search
  - Excalibur Image
- uključivanje vlastitog DataBlade modula u bazu podataka
  - kreiranje vlastitog DataBlade modula (alat BladeSmith)
    - postavljanje projekta za DataBlade modul
    - import objekata iz drugog DataBlade modula
    - definiranje novih objekata za DataBlade modul
  - pisanje potrebnih rutina u (npr. u C programskom jeziku); prevođenje
  - instaliranje DataBlade modula
  - registriranje DataBlade modula u bazi podataka (alat BladeManager)

Literatura

- S.W. Dietrich, S.D. Urban: **An Advanced Course in Database Systems : Beyond Relational Databases**, Prentice Hall, 2005
- T. Connolly, C. Begg: **Database Systems: A Practical Approach to Design, Implementation, and Management**, 4th Edition, Pearson Education , 2005
- A. Silberschatz, H.F. Korth, S. Sudarshan: **Database Systems Concepts**, 5th Edition, McGraw-Hill, 2005.
- M. Stonebraker, D. Moore: **Object-Relational DBMSs: The Next Great Wave**, Morgan Kaufmann Publichers, 1996
- R. Ramakrishnan, J. Gehrke: **Database Management Systems,**, McGraw-Hill, 2003
- **Oracle® Database, Application Developer's Guide - Object-Relational Features**, 10*g* Release 2 (10.2), 2005
- **IBM Informix Database Design and Implementation Guide**, Version 10.0, 2004
- **DataBlade Module Development Overview,** INFORMIX Press, 1998