OQL alapfogalmak

- Object Query Language (OQL) objektum-orientált adatbázisok lekérdezésére szolgál.
- az OQL-t használhatjuk egy objektum-orientált befogadó nyelv, például C++, Java, stb. kiterjesztéseként.
- a befogadó nyelvi utasítások és az OQL parancsok keveredhetnek anélkül, hogy a lekérdezés eredményei és befogadó nyelvi változók között az értékeket explicit át kellene adni.

példa: Legyen egy zenés CD-ket tartalmazó objektum-orientált adatbázis.
Feltételezzük:

- egy zenés CD típusa audio, vagy mp3-as, vagy videoklippeket tartalmaz és ezek nem keverednek.
- egy előadónak több albuma is van,
- egy albumon több zeneszám jelenik meg, viszont egy zeneszám csak egy albumon jelenik meg,
- egy albumon azonos stílusú zeneszámok jelennek meg.
- egy album teljes egészében egy CD-n található
- egy zeneszám, csak egyszer jelenik meg a CD-ken.

```
interface Előadó
   (extent Előadók
       key eNév)
{
   attribute string eNév;
   attribute string nemzetiség;
   relationship set<Album> albumai
      inverse Album::előadója;
};
```

```
interface Album
   (extent Albumok
       key (aCíme, megjelenEve))
{
    attribute string aCíme;
   attribute string stílus;
   attribute integer megjelenEve;
   relationship Előadó előadója
      inverse Előadó::albumai;
   relationship CD CDje
      inverse CD::CDalbumai;
   relationship set<Zeneszám> zeneszámai
      inverse Zeneszám::albuma;
 };
```

```
interface Zeneszám
   (extent Zeneszámok
        key (szCíme, időtartam))
{
   attribute string szCíme;
   attribute float időtartam;
   relationship Album albuma
        inverse Album::zeneszámai;
   string stílus() raises (nincsStílus);
   ElőadóNeve(out string);
};
```

```
interface CD
   (extent CDk
        key azonosító)
{
   attribute string azonosító;
   attribute enum CDTípus {audio, mp3,
videoklipp} típus;
   relationship set<Album> CDalbumai
        inverse Album::CDje;
 };
```

Típusok az OQL-ben

Az OQL-ben használt változókat a befogadó nyelvben deklaráljuk. Konstansokat a következő típusokból kiindulva építhetünk:

- alaptípusok
  - atomi típusok: egész számok, valós számok, karaktersorok és logikai értékek. A karakterláncokat dupla idézőjel közé helyezzük.
  - felsorolások: egy felsorolásban szereplő értékeket ODL-ben adjuk meg, ezek közül bármelyik használható konstans értékeként.

- összetett típusok
  - kollekciótípusok
    - Set()
    - Bag()
    - List()
    - Array()
  - Rekortípus
    - Struct().

példa:

```
struct (tanárgy: "matematika",
                jegyek:bag (9,8,9,7,8,10,9))
```

Útkifejezések

- összetett változók komponenseit a pont segítségével választjuk el.

Ha *o* egy *C* osztályba tartozó objektum és *k* az osztály valamely összetevője: attribútuma, kapcsolata vagy metódusa, akkor az *o*.*k* kifejezés az *o* objektum *k* komponensére hivatkozik.

Az összetevő típusa szerint a következő esetek állnak fenn:

- Ha *k* egy attribútum, akkor *o*.*k* az *o* objektum *k* attribútumának értéke.
- Ha *k* egy kapcsolat, akkor *o*.*k* az *o* objektumnak a *k* kapcsolaton keresztül kapcsolatban álló objektum vagy objektumok kollekciója.
- Ha *k* egy metódus, akkor az *o*.*k* a *k* *o*-ra való alkalmazásának az eredménye.

példa: Legyen a `kedvencZeneszám` egy Zeneszám típusú befogadó nyelvi változó, akkor:

- `A kedvencZeneszám.hossz` értéke, az adott zeneszám hossza, vagyis a hossz attribútum értéke abban a Zeneszám objektumban, mely a kedvencZeneszám változóban található.
- A kedvencZeneszám.stílus() értéke egy karaktersor, mely a zeneszám stílusát adja meg. Mivel a stílus nem a zeneszámnál, hanem a szám albumánál van megadva, írtunk egy metódust, mely megadja a zeneszám stílusát.
- A kedvencZeneszám.albuma megadja azt az Album objektumot, melyhez a kedvencZeneszám tartozik.
- A kedvencZeneszám.ElőadóNeve(ElőadóNév) szintén metódus meghívás, mely az ElőadóNév paraméter értékeként karaktersor formájában magadja a kedvencZeneszám előadójának a nevét .

A pontot többször is alkalmazhatjuk egy kifejezésben. A fenti példát folytatva a kedvencZeneszám.albuma.előadója.eNév kifejezés megadja a kedvencZeneszám változóban található Zeneszám objektum előadójának a nevét, a következőképpen:

- Amint már láttuk fennebb, a kedvencZeneszám.albuma azt az Album objektumot adja meg, melyhez a kedvencZeneszám tartozik. Tovább lépve,
- A kedvencZeneszám.albuma.előadója a fenti Album objektum Előadó objektumához vezet.
- A kedvencZeneszám.albuma.előadója.eNév megadja az keresett előadó nevét.

Select-from-where kifejezések OQL-ben

```
SELECT <kifejezések_listája>
FROM <változó_deklaráció>
WHERE <logikai_kifejezés>
```

példa: Keressük a „Gyöngyhajú lány” című zeneszám időtartamát.

```
SELECT z.időtartam
FROM Zeneszámok z
WHERE z.szCíme = "Gyöngyhajú lány"
```

A FROM kulcsszó után változókat adunk meg a következőképpen:

– egy kifejezés, amelynek értéke valamilyen kollekciótípusú (halmaz vagy multihalmaz).
  - leginkább: egy osztálynak az objektumkészlete, példa: Albumok objektumkészlet. (objektumkészlet ≈ relációs tábla)
  - kifejezés, mely kollekciót ad eredményül, példa: egy másik select-from-where kifejezés. SQL2-ben ez nem volt lehetséges, de kereskedelmi rendszerek (pl. Oracle) már megengedik az alkérdések használatát a FROM záradékban;
– opcionálisan az AS kulcsszó
– a változó neve (kötelező).

- A SELECT kulcsszó után a FROM záradékban deklarált változókat tartalmazó kifejezéseket adhatunk meg.

- A WHERE kulcsszó utáni logikai kifejezésben:
  - operandusok: a FROM záradékban deklarált változókat és konstansokat tartalmaznak.
  - összehasonlító operátorok: <, >, <=, >=, =, !=
  - logikai operátorok pedig: AND, OR, NOT.
  - karaktersor esetén dupla idézőjelet használtunk.

– A FROM záradékban több változó is szerepelhet.

példa: Keressük az Omega 7 album zeneszámait.

```
SELECT z.szCíme, z.időtartam
FROM Albumok a, a.zeneszámai z
WHERE a.aCíme = "Omega 7"
```

Algoritmikusan a kifejezés kiértékelése:

minden a-ra az Albumok-ból
minden z-re az a.zeneszámai-ból
ha a.aCíme = "Omega 7"
a z.szCíme, z.időtartam az eredménybe kerül

- select-from-where kifejezések eredménye multihalmaz
- ismétlődő sorokat megszüntethetjük a DISTINCT kulcsszóval

példa: Keressük azon előadókat, akik rock stílusú albumokat adtak ki. (egy előadónak több rock albuma is lehet)

```
SELECT DISTINCT e.eNév
FROM Előadók e, e.albumai a
WHERE a.stílus LIKE "%rock%"
```

Kvantort használó kifejezések

Az OQL lehetőséget ad logikai kvantorok: minden és létezik, használatára.

Legyen A egy halmaz, x egy változó és $F(x)$ egy feltétel.

`FOR ALL` *x* `IN` *A* **:** *F*(*x*)

– ellenőrzi, hogy az A halmaz minden eleme kielégíti az $F(x)$ feltételt.
– a kifejezés értéke TRUE, ha **minden** x az A-ból kielégíti az $F(x)$ feltételt, különben FALSE (hamis).

példa: Keressük azon előadókat, akik csak rock stílusú albumot adtak ki.

```
SELECT DISTINCT e.eNév
FROM Előadók e
WHERE FOR ALL a IN e.albumai :
a.stílus LIKE "%rock%"
```

– *e* változó végigjárja az Előadó osztály objektumkészletét
– *A* halmaz = e.albumai, melyet az *a*-val járunk végig
– az előadó bekerül az eredménybe, ha minden *a* albumra igaz a feltétel, hogy a stílusa rock.

Legyen A egy halmaz, x egy változó és $F(x)$ egy feltétel.

```
EXISTS x IN A : F(x)
```

- ha létezik legalább egy x az A-ból, mely kielégíti az $F(x)$ feltételt, a kifejezés értéke TRUE (igaz), különben FALSE (hamis).

példa: Keressük azon előadókat, akiknek van olyan zeneszáma, melyek címében szerepel a love szó.

```
SELECT DISTINCT a.előadója.eNév
FROM Albumok a
WHERE EXISTS z IN a.zeneszámai:
z.szCíme LIKE "%love%" OR
     z.szCíme LIKE "%loving%"
```

Alkérdések

- minden olyan helyen, ahol egy kollekció megjelenhet, használhatunk select-from-where kifejezést.

példa: Keressük Celine Dion zeneszámait!

```
SELECT z.szCíme, z.időtartam,
           a.aCíme,a.megjelenEve
    FROM (SELECT a FROM Albumok a
        WHERE a.előadója.eNév = "Celine Dion") c,
            c.zeneszámai z
```

OQL-ben is használhatjuk a halmazműveleteket

- UNION
- EXCEPT
- INTERSECT.

Az eredmény rendezése

példa: Adjuk meg a 2004-ben megjelent rap számok címét és előadójának a nevét előadók szerint rendezve, egy előadón belül pedig albumon belül rendezve.

```
SELECT DISTINCT e.eNév, a.aCíme, z.szCíme
FROM Előadók e, e.albumai a, a.zeneszámai.z
WHERE a.megjelenEve=2004
      AND a.stílus LIKE "%rap%"
ORDER BY e.eNév, a.aCíme, z.szCíme
```

Az alapértelmezett a növekvő sorrend (ASC),
a csökkenő sorrendet a DESC kulcsszóval kérhetjük.

Összesítő kifejezések

- OQL-ben is használhatjuk az AVG, COUNT, SUM, MIN, MAX összesítő operátorokat.
- SQL-ben tábla oszlopára vagy kifejezésekre, melyek táblák oszlopaira vonatkoznak,
- OQL-ben olyan kollekciókra alkalmazhatjuk, amelyek elemei megfelelő típusúak.

- A COUNT bármilyen típusú kollekcióra alkalmazható.
- A SUM és AVG aritmetikai típusú (egész számok, valós számok) kollekcióra.
- A MIN és MAX olyan típusú kollekciókra alkalmazhatóak, amelyeken értelmezett az összehasonlítás, például az aritmetikai típus és karaktersorok.

példa: Adjuk meg a CD-ken található zeneszámok összidejét!

```
SUM (SELECT z.időtartam
 FROM Zeneszámok z)
```

példa: Adjuk meg hány különböző gótikus metal album található a CD-ken.

```
COUNT (SELECT a.aCíme
    FROM Albumok a
    WHERE a.stílus = "gótikus metál")
```

Csoportosító kifejezések (GROUP BY)

példa: Adjuk meg zenénk összidejét stílusokra, évekre leosztva a következőképpen: StílusNév, Ev, Időtartam.

```
SELECT stl, év,
     összidő: SUM (SELECT p.z.időtartam
                        FROM partition p)
FROM Zeneszámok z
GROUP BY stl:z.albuma.stílus,
         év:z.albuma.megjelenEve
```

- a GROUP BY záradékban is megadunk változókat: *stl* és *év*
- minden egyes *stl* stílus esetén, minden *év* esetén, azon zeneszámok, melyek stílusa *stl* és megjelenési éve *év* bekerülnek a partition nevű multihalmazba, melyre összesítő függvényt alkalmazhatunk, a mi esetünkben összegezést.
- a partition olyan típusú objektumokat tartalmaz, amilyen a FROM után szerepel,
- a *p* változó végigfutja a partition multihalmazt.

- OQL-ben is használhatjuk a HAVING záradékot, melynek segítségével a GROUP BY által létrehozott csoportok közül csak azokat választjuk ki, melyek a HAVING feltételét kielégítik,
- a HAVING feltételében szintén a partition multihalmazra hivatkozhatunk.

példa: Egészítsük ki utolsó példánkat azzal, hogy csak akkor érdekel zenénk összideje stílusokra, évekre leosztva, ha az adott stílusból, az adott évben van legalább 10 perc zene.

```
SELECT stl, év,
       összidő: SUM (SELECT p.z.időtartam
                     FROM partition p)
FROM Zeneszámok z
GROUP BY stl: z.albuma.stílus,
         év: z.albuma.megjelenEve
HAVING SUM (SELECT p.z.időtartam
                     FROM partition p) >= 10
```

Objektumok létrehozása és kollekció végigjárása

- a C++-t fogjuk használni befogadó nyelvnek, melyben az OQL select-from-where kifejezéseit természetes módon lehet használni, nincs szükség adatátvitelre, mint SQL és befogadó nyelv között.
- a select-from-where kifejezések eredményként objektumokat állítanak elő, melyet megfelelő típusú változónak értékül adhatunk.

példa: Legyen SarahSzamai egy set<Zeneszám> típusú változó. OQL-el kiegészített C++-ban írhatjuk a következő értékadást:

```
SarahSzamai = SELECT DISTINCT z
        FROM Albumok a, a.zeneszámai z
        WHERE a.előadója.eNév = "Sarah Connor";
```

- egy select-from-where kifejezés eredményéhez soronként is hozzáférhetünk, ha listává alakítjuk ORDER BY kulcsszóval
- egy ilyen lista elemeihez a tömböknél megszokott módon férhetünk hozzá.

példa: `RasmusSzamok` egy `list<Zeneszám>` típusú változó;

`szam` egy `Zeneszám` típusú változó;

```
RasmusSzamok = SELECT DISTINCT z
               FROM Albumok a, a.zeneszámai z
               WHERE a.előadója.eNév = "The Rasmus"
               ORDER BY z.szCíme;
   NrSzamok = COUNT(SELECT DISTINCT z
              FROM Albumok a, a.zeneszámai z
               WHERE a.előadója.eNév = "The Rasmus");
   for (i=0; i<NrSzamok; i++) {
       szam = RasmusSzamok[i];
       cout << szam.szcíme << " "
            << szam.idotartam << "\n";
   }
```

`RasmusSzamok[i]` - egy zeneszám