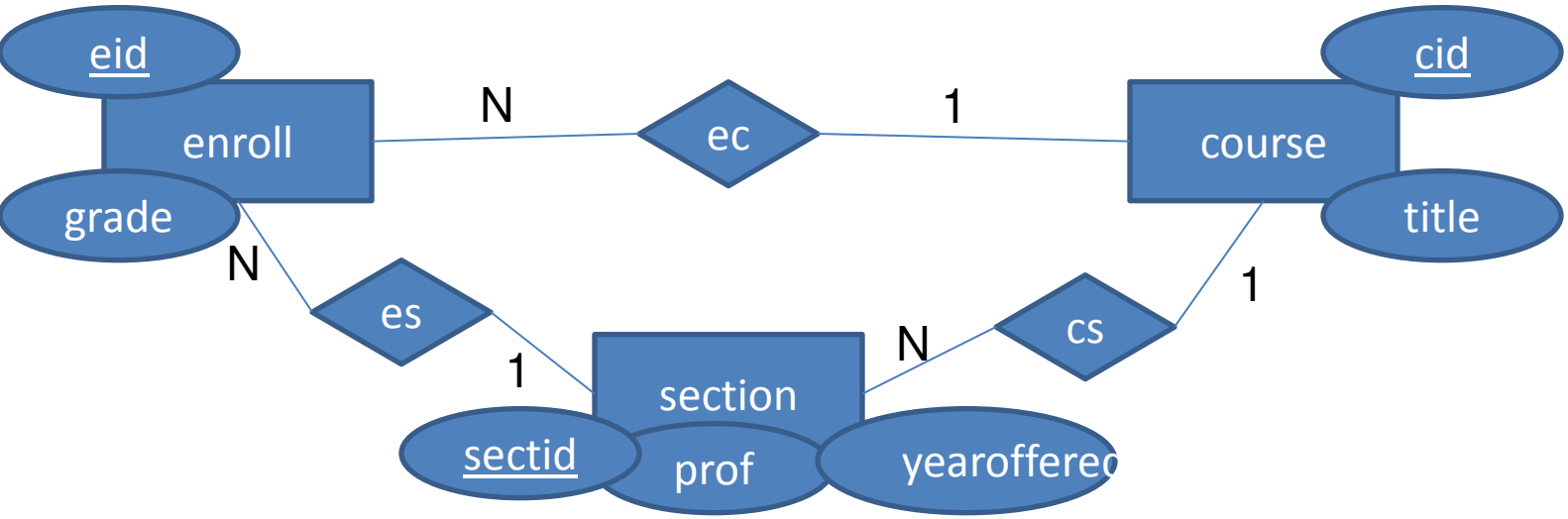
VT Sistem Gerçeklemesi Ders Notları- #12

- Somut görüntünün amacı ve kullanımı
  - Kontrollü Bilgi tekrarı
  - Bakım
  - Denormalizasyon yerine kullanımı
  - Görüntü seçimindeki kriterler
- Sorgu işlemede somutlaştırma
  - Somutlaştırma maliyeti
  - Somutlaştırma gerçeklemesi

Büyük VT *(Large databases)*

- Verimli,hızlı erişim için bazı yardımcı yapılara ihtiyacımız var:
  - İndeksleme
    - Arama anahtarları ile ana dosyaya daha hızlı erişim
  - **Somut Görüntüler (Materialized view)**
    - **Sorgu sonuçlarının geçici tablolarda saklanması**
- Maaliyet:
  - Yer kaybı
  - Bakım

Bilgi Tekrarı *(Redundancy)*



~~Enroll (eid, *sectionid*, grade)~~

Enroll (eid, *sectionid*, ***courseid*** grade)

Section (sectid, *courseid,* prof, yearoffered)

Course (cid, title)

- Bilgi Tekrarı, sorgu işleme hızı noktasından iyi…
- Bilgi takrarı, tasarım kalitesi oktasından kötü *(yenileme anormallikleri)*

```
select c.Title
from COURSE c, SECTION k, ENROLL e
where c.CId=k.CourseId and k.SectId=e.SectionId
and e.Grade='F'
```

```
select c.Title
from COURSE c, ENROLL e
where c.CId=e.CourseId and e.Grade='F'
```

«Kontrollü» bilgi tekrarı:

- Somut Görüntü (*Materialized views*)
- Indeksleme (*Indexes*)

- Sorgu işlemede görüntüler:
  - ***Normal görüntü***: Görüntü tanımı, sorgunun bir parçası olarak çalıştırılır.
    - Son Kullanıcı odaklı
    - Tipik kullanımı: Dış şema *(external schema)* tanımları
    - Bakım: yok
  - ***Somut Görüntü***: Görüntü doğrudan çalıştırılır, elde edilen çıktı tablosu (*geçici tablo*) sonraki sorgularda ihtiyaca göre **tekrar kullanılır. (reusibility)**
    - Sistem odakli
    - Bakım: VTYS tarafından otomatik
  - faydalı somut görüntü:
    - «Getirdiği sorgu işleme kazancı > bakım maaliyeti»
      - Görüntünün kullanım sıklığı
      - Bilgi yenileme sıklığı
  - VT tasarımcısı somut görüntülerin tanımlanması ve gerektiğinde çıkarılmasından sorumlu..
  - Önceki sayfada tanımlanabilecek somut görüntü:

```
create materialized view ENROLL_PLUS_CID as
   select e.*, k.CourseId
   from ENROLL e, SECTION k
   where e.SectionId=k.SectId
```

Bakım:

- Somut görüntüyü oluşturan tablolardaki olası değişikliklerin görüntüye yansıması:
  - Mevcut görüntüyü sil, yeniden oluştur.
  - Artımlı yenileme *(Incremental update)*

```
define materialized view STUDENT_STATS as
    select MajorId, count(SId) as HowMany,
           min(GradYear) as MinGradYear
    from STUDENT
    group by MajorId
```

- Updates on STUDENT:
  - insert into STUDENT <r>
  - delete from STUDENT <r>
  - modify <r> in STUDENT

```
create materialized view ENROLL_PLUS_CID as
   select e.*, k.CourseId
   from ENROLL e, SECTION k
   where e.SectionId=k.SectId
```

- Updates on ENROLL:
  - insert into ENROLL <r>
  - delete from ENROLL <r>
  - modify <r> in ENROLL

Somut görüntü / denormalizasyon

- Denormalizason, normalize edilmiş tabloların bazılarının, verimlilik amacıyla, birleştirilip (*join*) oluşturulmasıdır.

```
STUDENT(SId, SName, GradYear, MajorId)
DEPT(DId, DName)
ENROLL(EId, StudentId, SectionId, Grade)
SECTION(SectId, CourseId, Prof, YearOffered)
COURSE(CId, Title, DeptId)
```

(a) A normalized schema

```
STUD_ENR(EId, SId, SName, GradYear, MajorId,
         SectionId, Grade)
SECT_CRS(SectId, Prof, YearOffered, CId, Title, DeptId)
DEPT(DId, DName)
```

(b) A denormalized version of the schema

normalize şema + **Somut görüntüler** = hızlı sorgulama

```
create materialized view STUD_ENR as
   select s.*, e.EId, e.SectionId, e.Grade
   from STUDENT s, ENROLL e
   where s.SId=e.StudentId;
```

```
create materialized view SECT_CRS as
   select c.*, k.SectId, k.Prof, k.YearOffered
   from SECTION k, COURSE c
   where k.CourseId=c.CId
```

```
select c.Title
from STUDENT s, ENROLL e, SECTION k, COURSE c
where s.SId=e.StudentId and e.SectionId=k.SectId
and k.CourseId=c.CId and s.SName='joe'
```

(a) Using the normalized tables

Ne?

Sorgu eniyileyici

Nasıl?

```
select kc.Title
from STUD_ENR se, SECT_CRS kc
where se.SectionId=kc.SectId and se.SName='joe'
```

(b) Using the materialized views

En verimli görüntünün belirlenmesi:

- Karmaşık ve çok tablo içeren somut görüntü
  - **Az sayıda** karmaşık sorgular için **çok verimli** çalışma planları
  - Fakat küçük sorgular için kullanışsız..
  - Kullanım ihtimali düşük
- Az tablo içeren basit somut görüntüler
  - **Çok sayıda** sorgu için kullanışlı fakat
  - **Sorgu verimine katkısı az**..
  - Kullanım ihtimali yüksek
- Seçim kriteri:
  - Hedef sorguların kullanım sıklığı
  - Somut görüntünün içerdiği tabloların değişim sıklığı

Sorgu işlemede somutlaştırma:

- Bir sorguya ait altsorgu çıktısının geçici bir tabloda saklanmasıdır.
- VT'da SIRALAMA (*SORT*) işleminde kullanılır. (ORDER, GROUP BY, JOIN)

Sid	Sname	GradYear	MajorId
1	Joe	2004	10
2	Amy	2004	20
3	Max	2005	10
4	Sue	2005	20
5	Bob	2003	30
6	Kim	2001	20
7	Art	2004	30
8	Pat	2001	20
9	lee	2004	10

Sid	Sname	GradYear	MajorId
6	Kim	2001	20
8	Pat	2001	20
5	Bob	2003	30
1	Joe	2004	10
2	Amy	2004	20
7	Art	2004	30
9	lee	2004	10
3	Max	2005	10
4	Sue	2005	20

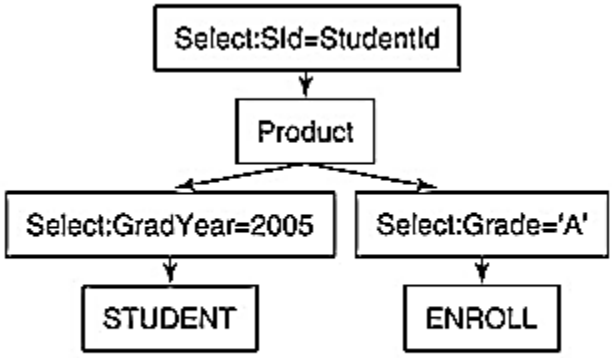
GradYear	CountOfSId
2001	2
2003	1
2004	4
2005	2

ÖNİŞLEME (somut tablo)

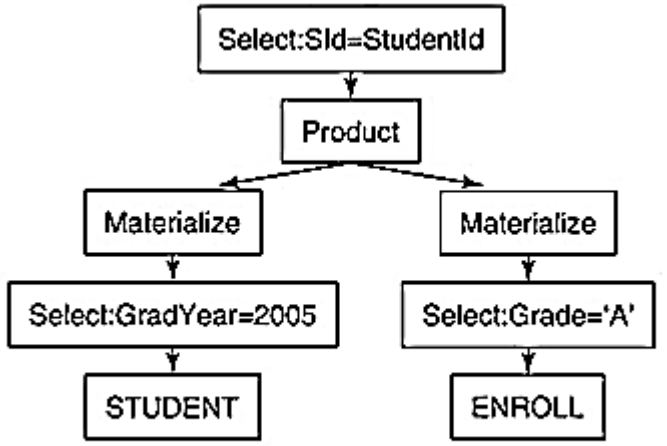
- Örnek: Group by:
  - Sıralama ➔ her grup üzerinde aggr. fonksiyonun çalıştırılması
- **Bakım/Maliyet**:
  - Geçici tablonun (somut tablo) oluşturulması için ekstra disk erişimi
  - Geçici tablonun kullanımında erişim için ekstra disk erişimi
  - Önişlemede somut tablo oluşturmayı gereksiz kılacak sorgular olabilir. *(Kullanıcının sadece 1-2 kayda erişmek istemesi gibi)*

Sorgu işlemede somutlaştırma:

- Boru hattı sorgu işlemede, bir operasyon sonucu elde edilen kayıtlar saklanmıyordu. Önceki bir kayda erişmek için ilgili altsorgu tekrar çalıştırılmalı idi.



(a) The original query



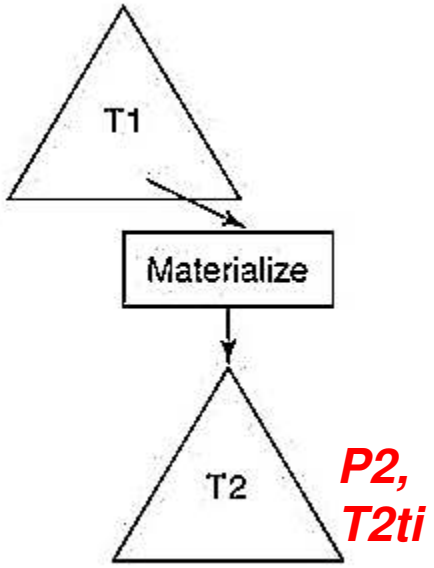
(b) Materializing the left and right sides of the product

- ***(KatalogYonetimi/Sunu:8'deki değerler):***
  - 2005'de, 900 STUDENT kaydı
  - 50.000 ENROLL blok
  - 4500 STUDENT blok
  - ENROLL tablosunda 14 farklı GRADE

- Maliyet
- 4500 + 900*50.000 = **45.004.500** blok erişimi

- Maliyet
- Somut Tablo: 3572 blok
  - Somut tablo oluşturma maliyeti: 53.572
- 53.572 + (4500 + 900*3572) = **3.272.872** blok erişimi

Somutlaştırma maliyeti



- Somut tablo maliyeti:
  - Önişleme maliyeti
    - «T2 sorgu ağacının maliyeti» + «geçici tabloya yazma maliyeti»
  - Somut tablo taraması=«geçici tablodaki blok sayısı»
- Örnek: Somut tablo **B** bloktan oluşursa:
  - Önişleme maliyeti=
    - «T2 sorgu ağacının maliyeti» + **B**
  - Somut tablo taraması =**B**

- MaterilizedPlan maliyet fonksiyonları:

blocksAccessed() = somut düğümdeki (temp_file) blok sayısı=B

(önişleme maliyeti ihmal edilebilir)

B= *ceiling* [p2.recordsOutput() /*floor* [BLOCK_SIZE/T2ti.recordsLength()]]

recordsOutput() = p2.recordsOutput();

distinctValues(F) = p2.distinctValues(F)

Geçici tablo

- Katalogda kaydı yoktur.
- İşi bitince sistem tarafından silinir.
- Veri kurtarma modülü tarafından takip edilmez.

```
public class TempTable {
   private static int nextTableNum = 0;
   private TableInfo ti;
   private Transaction tx;

   public TempTable(Schema sch, Transaction tx) {
      String tblname = nextTableName();
      ti = new TableInfo(tblname, sch);
      this.tx = tx;
   }

   public UpdateScan open() {
      return new TableScan(ti, tx);
   }

   public TableInfo getTableInfo() {
      return ti;
   }

   private static synchronized String nextTableName() {
      nextTableNum++;
      return "temp" + nextTableNum;
   }
}
```

Figure 22-1
The code for the SimpleDB class *TempTable*

Somutlaştırma operasyonunun (materialize(Q) =Q) gerçeklenmesi

```
public class MaterializePlan implements Plan {
   private Plan srcplan;
   private Transaction tx;

   public MaterializePlan(Plan srcplan, Transaction tx) {
      this.srcplan = srcplan;
      this.tx = tx;
   }

   public Scan open() {
      Schema sch = srcplan.schema();
      TempTable temp = new TempTable(sch, tx);
      Scan src = srcplan.open();
      UpdateScan dest = temp.open();
      // copy the input records to the temporary table
      while (src.next()) {
         dest.insert();
         for (String fldname : sch.fields())
            dest.setVal(fldname, src.getVal(fldname));
      }
      src.close();
      dest.beforeFirst();
      return dest;
   }

   public int blocksAccessed() {
      // we create a dummy TableInfo object
      // in order to calculate record length
      TableInfo ti = new TableInfo("dummy",
                                   srcplan.schema());
      double rpb = (double) (BLOCK_SIZE /
                             ti.recordLength());
      return (int) Math.ceil(srcplan.recordsOutput() / rpb);
   }

   public int recordsOutput() {
      return srcplan.recordsOutput();
   }

   public int distinctValues(String fldname) {
      return srcplan.distinctValues(fldname);
   }

   public Schema schema() {
      return srcplan.schema();
   }
}
```