VT Sistem Gerçeklemesi Ders Notları- #7

Remote: Kullanıcıdan gelen JDBC isteklerini karşılar.

Planner: SQL ifadesi için işleme planı oluşturur ve karşılık gelen ilşkisel cebir ifadesini oluşturur.

Parse: SQL ifadesindeki tablo, nitelik ve ifadeleri ayrıştırır.

Query: Algebra ile ifade edilen sorguları gerçekler.

Metadata: Tablolara ait katalog bilgilerini organize eder.

Record: disk sayfalarına yazma/okumayı kayıt seviyesinde gerçekler.

Transaction&Recovery: Eşzamanlılık için gerekli olan disk sayfa erişimi kısıtlamalarını organize eder ve veri kurtarma için kayıt_defteri (log) dosyalarına bilgi girer.

Buffer: En sık/son erişilen disk sayfalarını ana hafıza tampon bölgede tutmak için gerekli işlemleri yapar.

Log: Kayıt_defterine bilgi yazılmasını ve taranması işlemlerini düzenler.

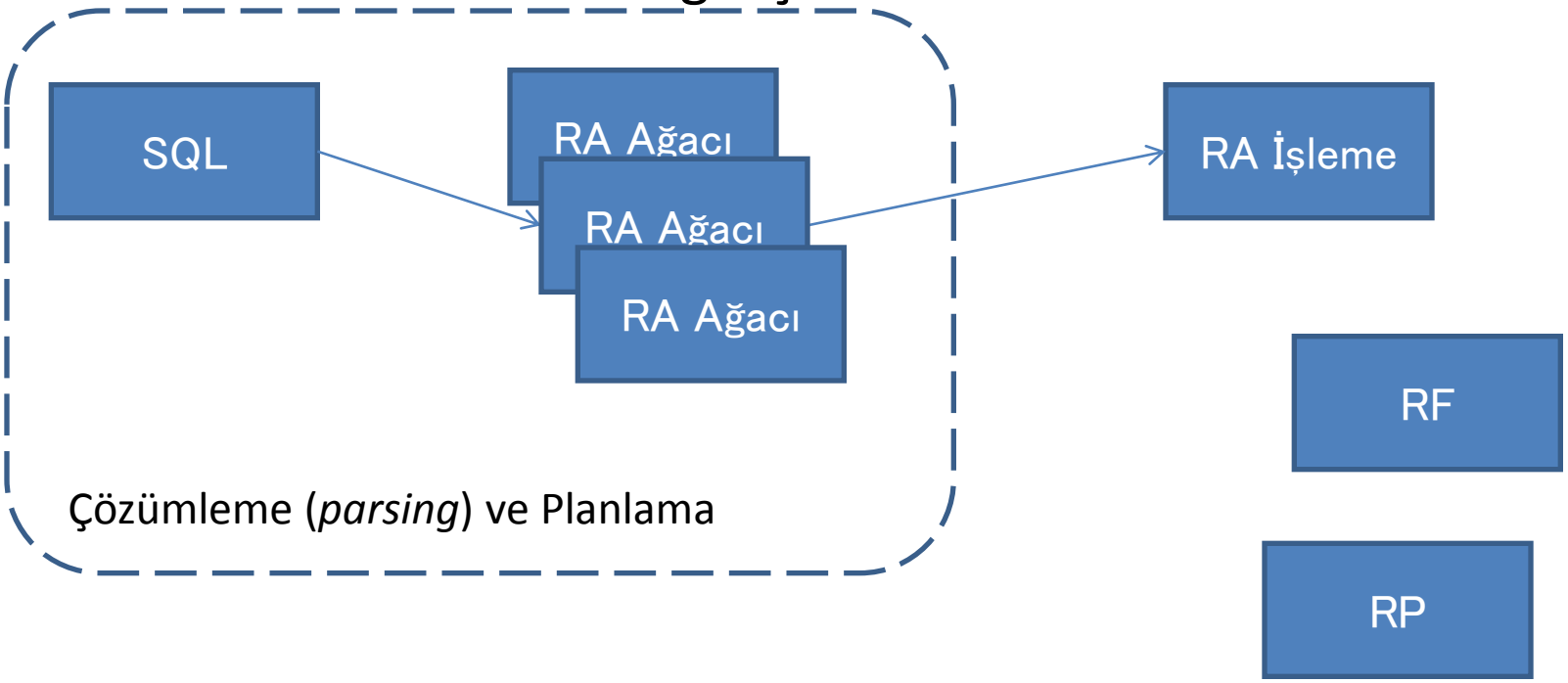
File: Dosya blokları ile ana hafıza sayfaları arasında bilgi transferini organize eder.

Sorgu İşleme

- Tarama (*Scan*), Yenileme Taramaları (*Update Scan*)
- *Table, Select, Project, Product* Taramaları
- Boru hattı Sorgu İşleme (*Pipelined Q Proc*)
- Tarama Maaliyetleri
- Planlama
- Yüklemler (*Predicates*)

Neredeyiz?

- Çok sayıda hareketin, kısıtlı depolama kaynaklarını (tampon) kullanarak, kayıt dosyalarına eşzamanlı/güvenli veri ve üst-veri erişimi gerçekleştirildi.
- Bundan sonra SQL sorgu işleme:



Tarama arayüzü (*Scan*)

•Kayıt dosyası (*RecordFile*) üzerinde sorgu işlemeyi kontrol eden arayüzdür.

•İlişkisel algebra ağacındaki her bir düğüm = bir ilişkisel operatör

•Her bir ilişkisel operatör, Scan ara yüzünü gerçekler.

SELECT --------> PROJECTSCAN

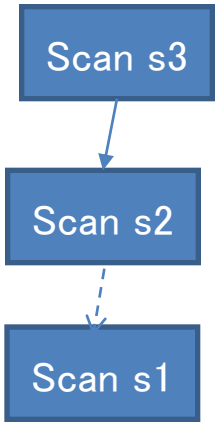
FROM ---------> TABLESCAN, PRODUCTSCAN

```
public SelectScan(Scan s, Predicate pred);
public ProjectScan(Scan s, Collection<String> fldlist);
public ProductScan(Scan s1, Scan s2);
public TableScan(TableInfo ti, Transaction tx);
```

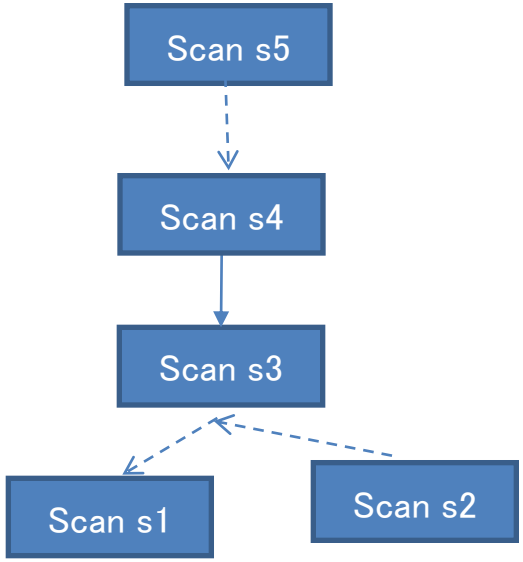
Figure 17-3
The API of the constructors in SimpleDB that implement *Scan*

• Sorgu işleme, ilişkisel cebir ağacında yapraklardan yukarıya doğru gerçekleşir. Yapraklarda her zaman **TableScan** operatörü, diğer düğümlerdeki operatörler ise sorgu cümlesine göre belirlenir.

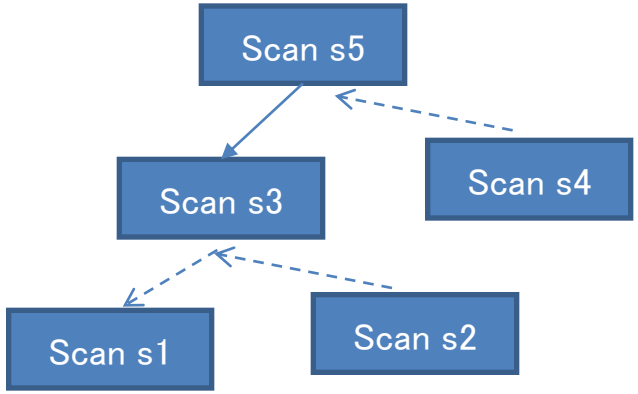
örnek İlşkisel cebir ağaçları



- s1:tablescan
- s2,s3: selectscan, projectscan



- s1,s2:tablescan
- s3:productscan
- s4,s5: selectscan, projectscan



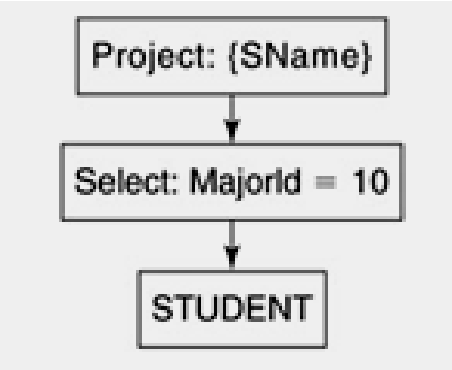
- s1,s2,s4:tablescan
- s3,s5:productscan

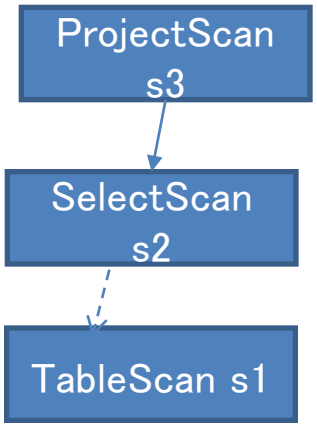
Örnek-1

SELECT SName

FROM STUDENT

WHERE MajorId=10



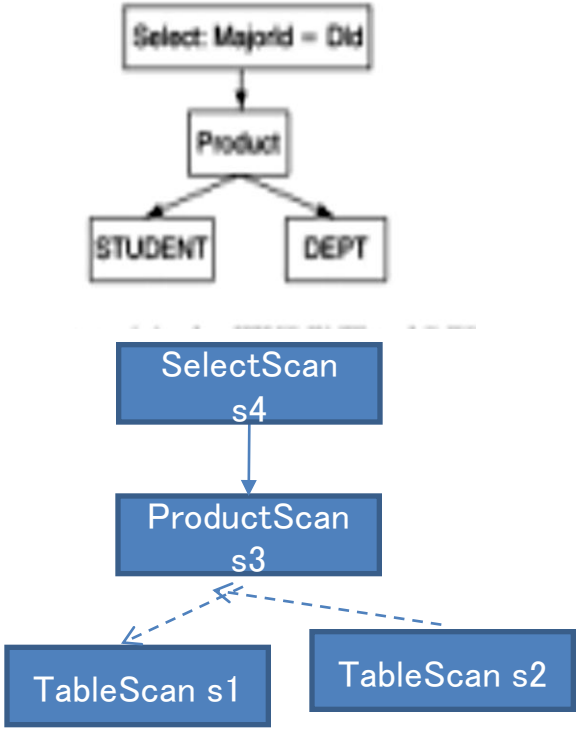



```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();

// the STUDENT node
TableInfo ti = SimpleDB.mdMgr().getTableInfo
                                    ("student",tx);
Scan s1 = new TableScan(ti, tx);

// the Select node
Predicate pred = new Predicate(. . .); //majorid=10
Scan s2 = new SelectScan(s1, pred);

// the Project node
Collection<String> c = Arrays.asList("sname");
Scan s3 = new ProjectScan(s2, c);

while (s3.next())
   System.out.println(s3.getString("sname"));
s3.close();
```

Örnek-2

SELECT SName,SId, MajorId,DId,DName
FROM STUDENT, DEPT

WHERE MajorId=DId




```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
MetadataMgr mdMgr = SimpleDB.mdMgr();

// the STUDENT node
TableInfo sti = mdMgr.getTableInfo("student", tx);
Scan s1 = new TableScan(sti, tx);

// the DEPT node
TableInfo dti = mdMgr.getTableInfo("dept", tx);
Scan s2 = new TableScan(dti, tx);

// the Product node
Scan s3 = new ProductScan(s1, s2);

// the Select node
Predicate pred = new Predicate(. . .); //majorid=did
Scan s4 = new SelectScan(s3, pred);

while (s4.next())
   System.out.println(s4.getString("sname")
            + ", " + s4.getString("gradyear")
            + ", " + s4.getString("dname") );
s4.close();
```

Örnek-3 (*güncellenebilir Scan*)

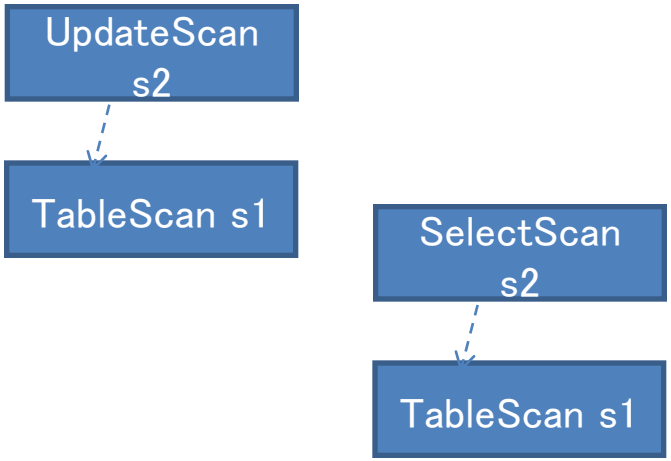
```
update ENROLL
set Grade = 'C'
where SectionId = 53
```

(a) An SQL statement to modify the grades of students in section 53

```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
TableInfo ti = SimpleDB.mdMgr().getTableInfo
                                        ("student",tx);
Scan s1 = new TableScan(ti, tx);

Predicate pred = new Predicate(. . .); //SectionId=53
UpdateScan s2  = new SelectScan(s1, pred);

while (s2.next())
   s2.setString("grade", "C");
s2.close();
```

(b) The SimpleDB code corresponding to the update

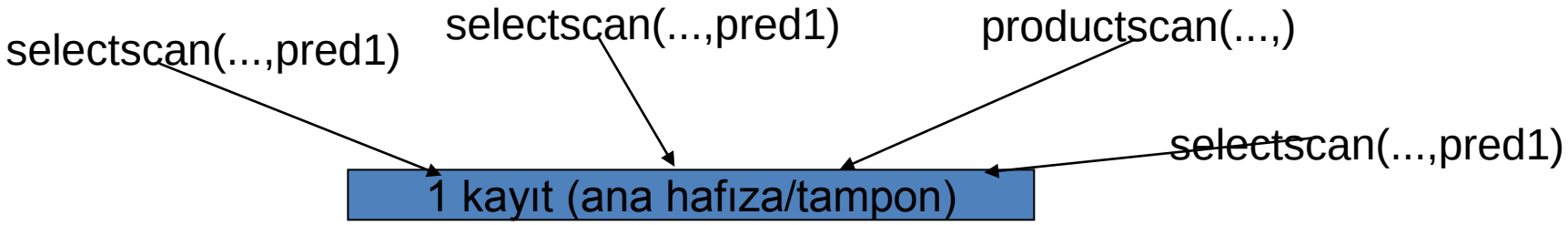
Figure 17-7
Representing an SQL update statement as an update scan



- Hangi Scan gerçeklemeleri güncellenebilir (*updatable*)?
  - S2deki her bir r kaydı, s1 TableScan'de bir r'kaydına denk geliyorsa s2 güncellenebilir.
  - TableScan, SelectScan ve ProjectScan güncellenebilir Scan gerceklemeleridir. ProductScan güncellenemez.

Sorgu ağacının işlenmesi

- 1.yol: Somutlaştırma (*materilization*): işlenen her bir düğümün sonucu diskte saklanır. Bir sonraki (ağaçta üstteki) düğüme girdi (*input*) olur.
- 2.yol: Boru hattı (pipelined): bütün düğümlerdeki operasyonların içiçe girmesi (*interleave*) olayıdır. TableScan ile taranan kayıtlar, --diske kaydedilmeden-- ağaçtaki operasyonlarda dolaşırlar. Bu ağacı temsil eden, "aktif tarama(*scan,iterator*) ağı" ile gerçeklenir. (network of scans).



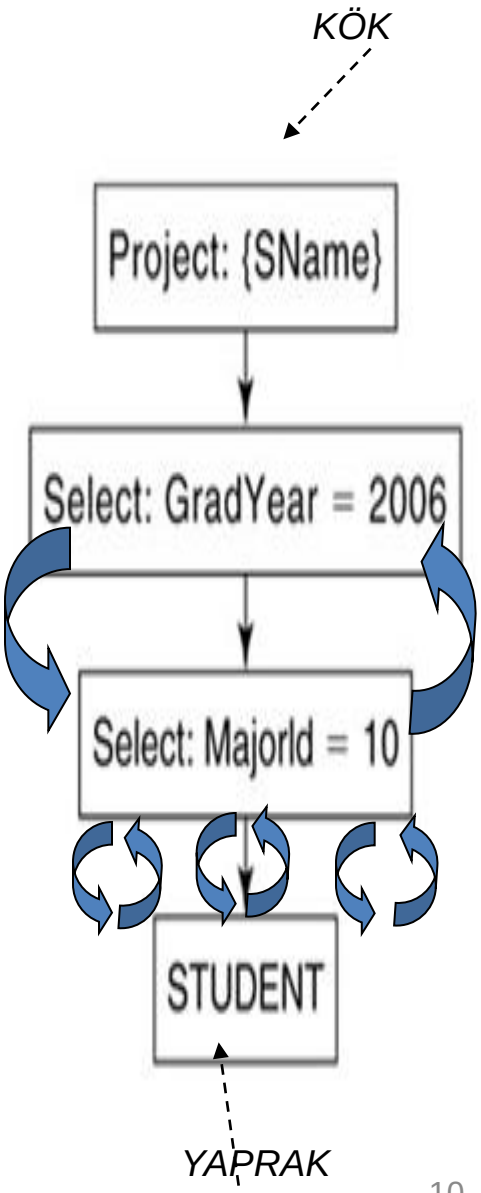
Tarama (*Scan,iterator*) arayüzü

RecordFile'e benzer. Fakat , Scan sorgu işlemenin parçaları olan operasyonlar (Select,Project,Product,TableScan) "**o operasyona özgü işlevi yerine getirerek**" bu SCAN'ı gerçekler.

```
public interface Scan {
   public void    beforeFirst();
   public boolean next();
   public void    close();
   public Constant getVal(String fldname);
   public int      getInt(String fldname);
   public String   getString(String fldname);
   public boolean  hasField(String fldname);
}
public interface UpdateScan extends Scan {
   public void setVal(String fldname, Constant val);
   public void setInt(String fldname, int val);
   public void setString(String fldname, String val);
   public void insert();
   public void delete();

   public RID  getRid();
   public void moveToRid(RID rid);
}
```

Boru hattı sorgu işleme *(pipelined Query Proc.)*

- Botu hattı işlem: yapraklarda bulunan kayıtlardan herbirini, enine ve/veya boyuna filtrelerde geçirerek ağacın köküne(yukarıya doğru), herhangi bir saklama yapmadan çıkartacak şekilde işlemektir.
  - Kökteki herbir next(), (varsa) bir sonuç kaydı üretiyor.
  - Ara sonuçlar ve Sonuç kayıtları kaydedilmiyor.
- TableScan ve ProjectScan next(), temel sınıfta sadece 1 next() çağırır.
- SelectScan sınıfında next(), bu sınıfın temel sınıfında yüklem sağlanıncaya kadar bir veya daha çok next() komutunu tetikleyecektir.
- ProductScan next(), temel sınıfta 1, enfazla 2 next() çağırır.



```
public class TableScan implements UpdateScan {
   private RecordFile rf;
   private Schema sch;

   public TableScan(TableInfo ti, Transaction tx) {
      rf  = new RecordFile(ti, tx);
      sch = ti.schema();
   }

   // Scan methods

   public void beforeFirst() {
      rf.beforeFirst();
   }

   public boolean next() {
      return rf.next();
   }

   public void close() {
      rf.close();
   }

   public Constant getVal(String fldname) {
      if (sch.type(fldname) == INTEGER)
         return new IntConstant(rf.getInt(fldname));
      else
         return new StringConstant(rf.getString(fldname));
   }

   public int getInt(String fldname) {
      return rf.getInt(fldname);
   }

   public String getString(String fldname) {
      return rf.getString(fldname);
   }

   public boolean hasField(String fldname) {
      return sch.hasField(fldname);
   }

   // UpdateScan methods

   public void setVal(String fldname, Constant val) {
      if (sch.type(fldname) == INTEGER)
         rf.setInt(fldname, (Integer)val.asJavaVal());
      else
         rf.setString(fldname, (String)val.asJavaVal());
   }

   public void setInt(String fldname, int val) {
      rf.setInt(fldname, val);
   }

   public void setString(String fldname, String val) {
      rf.setString(fldname, val);
   }

   public void delete() {
      rf.delete();
   }

   public void insert() {
      rf.insert();
   }

   public RID getRid() {
      return rf.currentRid();
   }

   public void moveToRid(RID rid) {
      rf.moveToRid(rid);
   }
}
```

Figure 17-8
The code for the SimpleDB class *TableScan*

Figure 17-8 *(Continued)*

```
public class SelectScan implements UpdateScan {
   private Scan s;
   private Predicate pred;

   public SelectScan(Scan s, Predicate pred) {
      this.s = s;
      this.pred = pred;
   }

   // Scan methods

   public void beforeFirst() {
      s.beforeFirst();
   }

   public boolean next() {
      while (s.next())
         if (pred.isSatisfied(s))
            return true;
      return false;
   }

   public void close() {
      s.close();
   }

   public Constant getVal(String fldname) {
      return s.getVal(fldname);
   }

   public int getInt(String fldname) {
      return s.getInt(fldname);
   }

   public String getString(String fldname) {
      return s.getString(fldname);
   }

   public boolean hasField(String fldname) {
      return s.hasField(fldname);
   }

   // UpdateScan methods

   public void setVal(String fldname, Constant val) {
      UpdateScan us = (UpdateScan) s;
      us.setVal(fldname, val);
   }

   public void setInt(String fldname, int val) {
      UpdateScan us = (UpdateScan) s;
      us.setInt(fldname, val);
   }

   public void setString(String fldname, String val) {
      UpdateScan us = (UpdateScan) s;
      us.setString(fldname, val);
   }

   public void delete() {
      UpdateScan us = (UpdateScan) s;
      us.delete();
   }

   public void insert() {
      UpdateScan us = (UpdateScan) s;
      us.insert();
   }

   public RID getRid() {
      UpdateScan us = (UpdateScan) s;
      return us.getRid();
   }

   public void moveToRid(RID rid) {
      UpdateScan us = (UpdateScan) s;
      us.moveToRid(rid);
   }
}
```

Figure 17-9
The code for the SimpleDB class *SelectScan*

Figure 17-9 *(Continued)*

```
public class ProjectScan implements Scan {
   private Scan s;
   private Collection<String> fieldlist;

   public ProjectScan(Scan s,
                      Collection<String> fieldlist) {
      this.s = s;
      this.fieldlist = fieldlist;
   }

   public void beforeFirst() {
      s.beforeFirst();
   }

   public boolean next() {
      return s.next();
   }

   public void close() {
      s.close();
   }

   public Constant getVal(String fldname) {
      if (hasField(fldname))
         return s.getVal(fldname);
      else
         throw new RuntimeException("field not found.");
   }

   public int getInt(String fldname) {
      if (hasField(fldname))
         return s.getInt(fldname);
      else
         throw new RuntimeException("field not found.");
   }

   public String getString(String fldname) {
      if (hasField(fldname))
         return s.getString(fldname);
      else
         throw new RuntimeException("field not found.");
   }

   public boolean hasField(String fldname) {
      return fieldlist.contains(fldname);
   }
}
```

Figure 17-10
The code for the SimpleDB class *ProjectScan*

Figure 17-10 *(Continued)*

```
public class ProductScan implements Scan {
   private Scan s1, s2;

   public ProductScan(Scan s1, Scan s2) {
      this.s1 = s1;
      this.s2 = s2;
      s1.next();
   }

   public void beforeFirst() {
      s1.beforeFirst();
      s1.next();
      s2.beforeFirst();
   }

   public boolean next() {
      if (s2.next())
         return true;
      else {
         s2.beforeFirst();
         return s2.next() && s1.next();
      }
   }

   public void close() {
      s1.close();
      s2.close();
   }

   public Constant getVal(String fldname) {
      if (s1.hasField(fldname))
         return s1.getVal(fldname);
      else
         return s2.getVal(fldname);
   }

   public int getInt(String fldname) {
      if (s1.hasField(fldname))
         return s1.getInt(fldname);
      else
         return s2.getInt(fldname);
   }

   public String getString(String fldname) {
      if (s1.hasField(fldname))
         return s1.getString(fldname);
      else
         return s2.getString(fldname);
   }

   public boolean hasField(String fldname) {
      return s1.hasField(fldname) || s2.hasField(fldname);
   }
}
```

Figure 17-11
The code for the SimpleDB class *ProductScan*

Figure 17-11 *(Continued)*

Tarama Maliyetleri

- Scan S için aşağıdaki maliyet tanımları yapılır:
  - B(S): S'nin sonlanması için gerekli blok erişim sayısı
  - R(S): S'nin sonlanması ile ortaya çıkan toplam kayıt sayısı
  - V(S,F): S'nin sonlanması ile ortaya çıkan kayıtların F niteliklerindeki farklı toplam değer sayısı

s	B(s)	R(s)	V(s, F)
TableScan(T)	B(T)	R(T)	V(T,F)
SelectScan(s_1,A=c)	B(s_1)	R(s_1)/V(s_1,A)	1 if F = A min{R(s), V(s_1,F)} if F ≠ A
SelectScan(s_1,A=B)	B(s_1)	R(s_1) / max{V(s_1,A), V(s_1,B)}	min{V(s_1,A), V(s_1,B)} if F = A,B min{R(s), V(s_1,F)} if F ≠ A,B
ProjectScan(s_1,L)	B(s_1)	R(s_1)	V(s_1,F)
ProductScan(s_1,s_2)	B(s_1) + R(s_1)*B(s_2)	R(s_1)*R(s_2)	V(s_1,F) if F is in s_1 V(s_2,F) if F is in s_2

Figure 17-13
The statistical cost formulas for scans

Varsayım: V(s1,A) > V(s1,B) ⇒ her bir B-değeri A niteliğinde mutlaka gözüküyor.

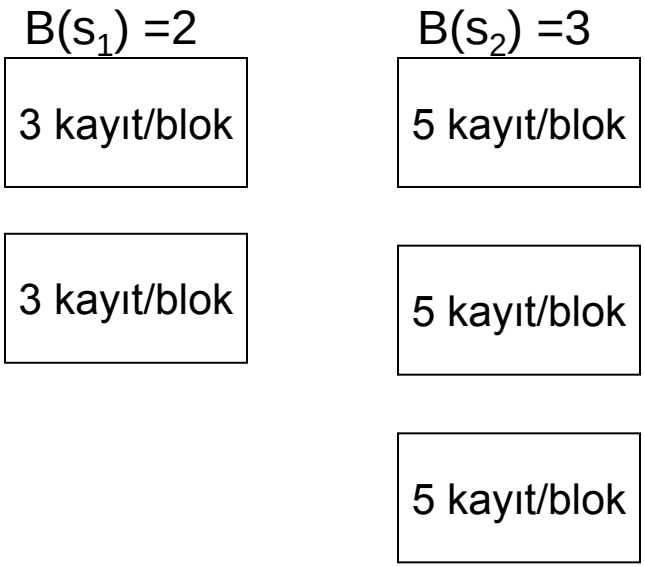
R(s):SelectScan(s_1,A=B) operatörü için, R(s) ve V(s,F)

- Bu operator nerede kullanılıyor ?
- R(s_1) =100
- V(s_1,A)= 25
- V(s_1,B)= 2
- R(s), V(s,F) değerleri nasıl tahminedilir?
- V(s_1,A) > V(s_1,B) ==> *B niteliği A'ya işaret ediyor. (B: yabancı anahtar.) Bir kayıtta, B değerinin A değerinine eşit olması olasılığı 1/*V(s_1,A) olur.
- Buna göre, R(s)=4 olur.

V(s,F) =?

- F=A,B ise; *(yukarıdaki varsayımı dikkate alırsak..)*
  - min(V(s_1,A), V(s_1,B)) = 2
- F≠ A,B
  - min(R(s), V(s_1,F))

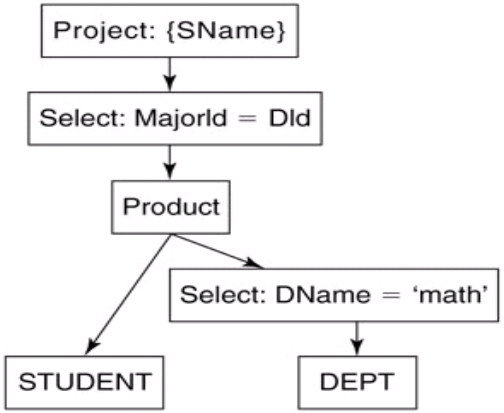
$R(s):ProductScan(s_1, s_2)$ operatörünün simetrik **olmaması**



- SONUÇ :
  - Daha verimli bir Product (Join) işlemi için; RPB(s) değeri küçük olan sol tarafta olmalı...Fakat mutlak değil, Sunum 20'deki örnek gibi

- s: ProductScan(s_1, s_2)
  - $B(s) = B(s_1) + (R(s_1)*B(s_2))$
    $= 2 + 6*3 = 20$
- s: ProductScan(s_2, s_1)
  - $B(s) = B(s_2) + (R(s_2)*B(s_1))$
    $= 3 + 15*2 = 33$
- tanım: **RPB(s)** *(record/block)*
  - *RPB(s) = R(s) / B(s)*
  - *R(s) = RPB(s) * B(s)*
- s: ProductScan(s_1, s_2) için
  - $B(s) = B(s_1) + (RPB(s_1)*B(s_1)*B(s_2))$
- s: ProductScan(s_2, s_1) için
  - $B(s) = B(s_2) + (RPB(s_2)*B(s_2)*B(s_1))$

Örnek



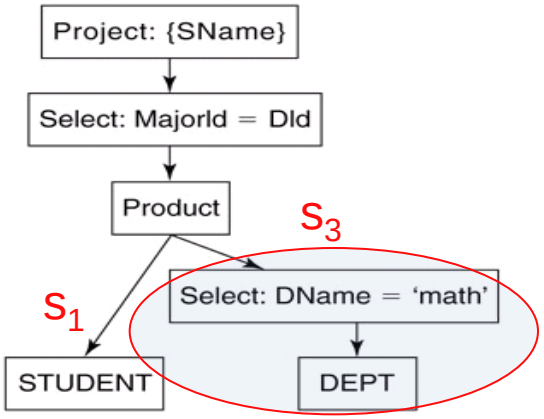

```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
MetadataMgr mdMgr = SimpleDB.mdMgr();
TableInfo sti = mdMgr.getTableInfo("student", tx);
TableInfo dti = mdMgr.getTableInfo("dept", tx);
Scan s1 = new TableScan(sti, tx);
Scan s2 = new TableScan(dti, tx);
Predicate pred1 = new Predicate(. . .); //DName='math'
Scan s3 = new SelectScan(s2, pred1);
Scan s4 = new ProductScan(s1, s3);
Predicate pred2 = new Predicate(. . .); //majorid=DId
Scan s5 = new SelectScan(s4, pred2);
Collection<String> fields = Arrays.asList("SName");
Scan s6 = new ProjectScan(s5, fields);
```

s	B(s)	R(s)	V(s,F)
s1	4,500	45,000	45,000 for F=SId 44,960 for F=SName 50 for F=GradYear 40 for F=MajorId
s2	2	40	40 for F=DId, DName
s3	2	1	1 for F=DId, DName
s4	94,500	45,000	45,000 for F=SId 44,960 for F=SName 50 for F=GradYear 40 for F=MajorId 1 for F=DId, DName
s5	94,500	1,125	1,125 for F=SId 1,124 for F=SName 50 for F=GradYear 1 for F=MajorId, DId, DName
s6	94,500	1,125	1,124 for F=SName

R(s4) / max(V(s4,majorId),V(s4,DId))

(c) The cost of the code

Örnek (devam ...)



- *ProductScan (s_1,s_3)* işlemini *ProductScan (s_3,s_1)* ile karşılaştıralım:
  - RPB (s_3) = ½ = 0,5
  - RPB(s_1) = 45000 / 4500 = 10
- O zaman; *ProductScan (s_3,s_1)* daha iyi:
  - 2 + 0,5*2*4500 = 4502 < 94.500
- *ProductScan (s_1,s_3)* işleminde *select(Dept, math')* operatörü 45.000 kez çalıştırılır... *ProductScan (s_3,s_1)*'de ise *select(Dept, math')* işlemi 1 kez çalıştırılır...

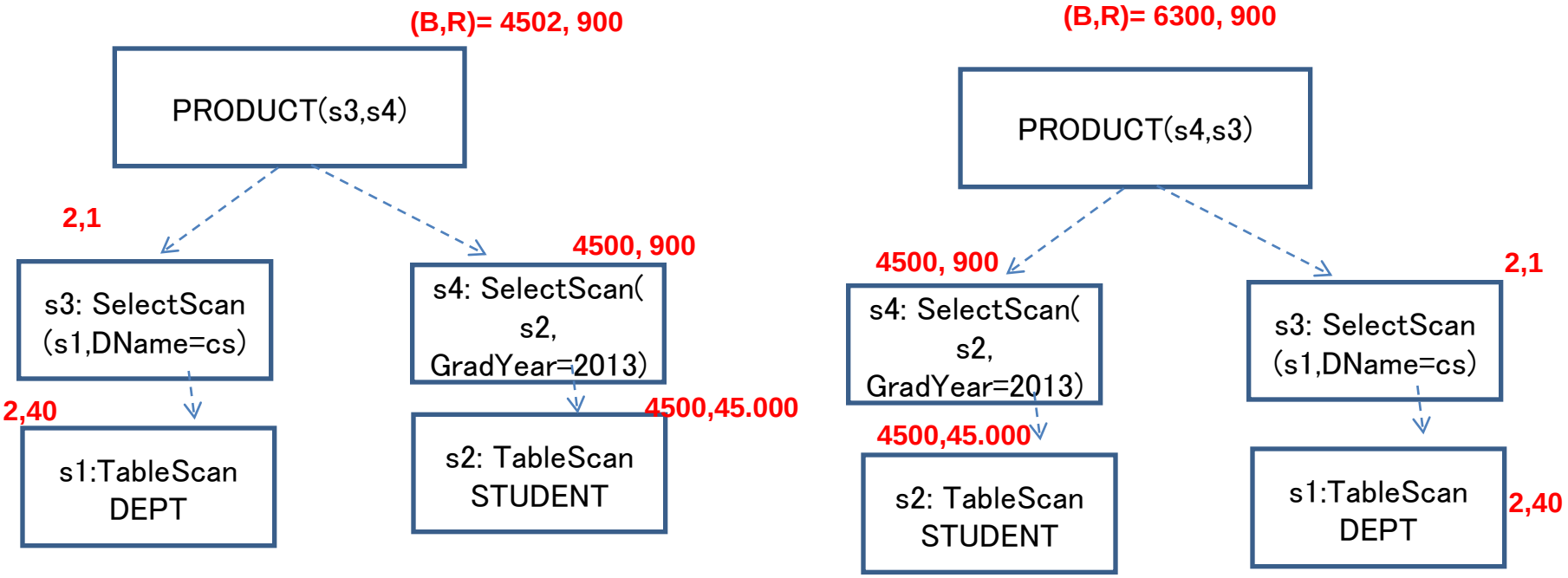
Örnek

T3: select (DEPT, DName="cs")

T4: select (STUDENT, GradYear=2013)

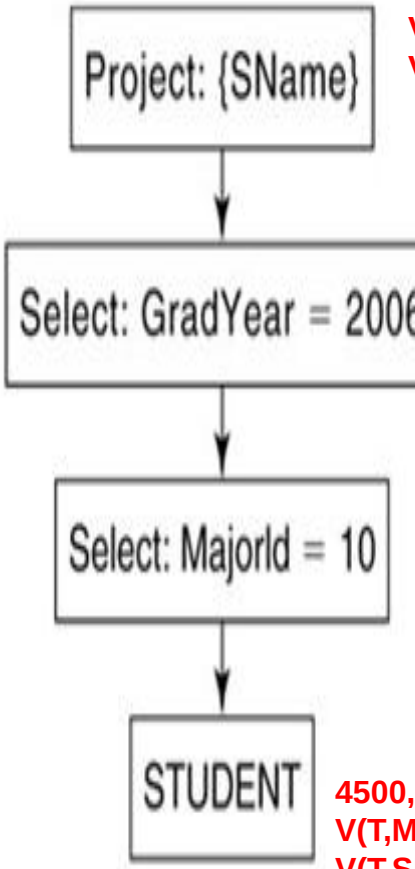
product (T3,T4) =?

product (T4,T3) =?



- rpb (s4) < rpb (s3) olmasına rağmen PRODUCT(s4,s3) daha yüksek maliyet verdi. Bunun sebebi, B(s4) >> B(s3) olması.

Örnek



s3

4500,23
V(s3,MajorId) = -
V(s3,SId)= -
V(s3,SNAme)= 23
V(s3,GradYear)= -

s2

4500, 23
V(s2,MajorId) = 1
V(s2,SId)= 23
V(s2,SNAme)= 23
V(s2,GradYear)= 1

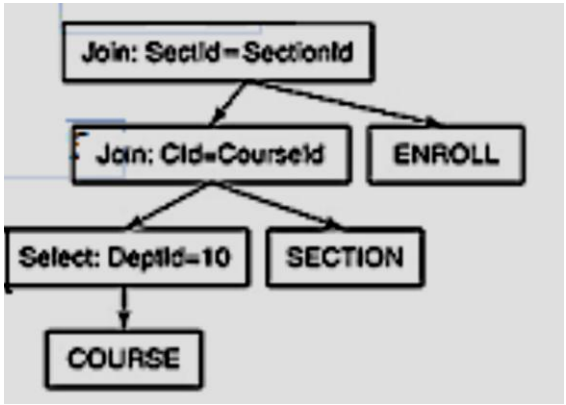
s1

4500, 1125
V(s1,MajorId) =1
V(s1,SId)= 1125
V(s1,SNAme)= 1125
V(s1,GradYear)= 50

T

4500,45.000
V(T,MajorId) = 40
V(T,SId)= 45000
V(T,SNAme)= 44960
V(T,GradYear)=50

Örnek



COURSE	25	500	500 40	for F=CId, Title for F=DeptId
SECTION	2,500	25,000	25,000 500 250 50	for F=SectId for F=CourseId for F=Prof for F=YearOffered
ENROLL	50,000	1,500,000	1,500,000 25,000 45,000 14	for F=EId for F=SectionId for F=StudentId for F=Grade

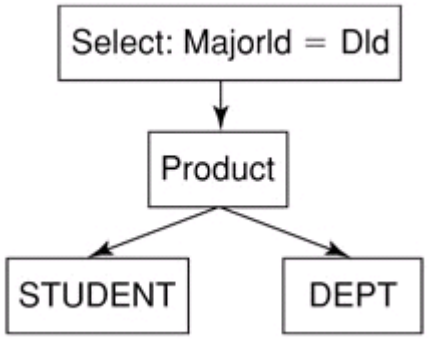
	B(s)	R(s)
s1	25	500/40 =13
s2	25 + 13*2500=32,525	13*25000=325,000
s3	32,525	325000 / max(13,500) = 650
s4	32525 + 650*50000 = 32,532,525	650*1,500,000 = 975,000,000
s5	32,532,525	975,000,000 / 25,000 = 39,000

Planlama

- Maliyet karşılaştırması amacıyla oluşturulan sorgu ağacına PLAN ismi verilir.
- PLAN / SCAN karşılaştırma:
  - İkisi de sorgu ağacını temsil eder..
  - PLAN üstveriye ulaşarak maliyet hesabı yapar; SCAN ise (RF sınıfı ile) verinin kendisine erişerek, sorguyu çalıştırır...
  - Bir çok PLAN arasından en az maliyeti olan seçilerek karşılık gelen SCAN oluşturulur. Bu Planlayıcı (PLANNER) modülünün görevidir.
  - PLAN arayüzünü her bir operatör (TABLE, SELECT, PROJECT, PRODUCT) gerçekler ve aşağıdaki istatistikleri hesaplaması gerekir:
    - B(s)
    - R(s)
    - V(s,F)

```
public interface Plan {
   public Scan    open();
   public int     blocksAccessed();
   public int     recordsOutput();
   public int     distinctValues(String fldname);
   public Schema  schema();
}
```

Örnek




```
SimpleDB.init("studentdb");
Transaction tx = new Transaction();
Plan p1 = new TablePlan("student", tx);
Plan p2 = new TablePlan("dept", tx);
Plan p3 = new ProductPlan(p1, p2);
Predicate pred = new Predicate(. . .); //majorid=DId
Plan p4 = new SelectPlan(p3, pred);

Scan s = p4.open();
while (s.next())
   System.out.println(s.getString("sname"));
s.close();
```

```
public interface Plan {
   public Scan    open();
   public int     blocksAccessed();
   public int     recordsOutput();
   public int     distinctValues(String fldname);
   public Schema  schema();
}
```

PLAN gerçeklemeleri

- TABLEPLAN
  - Tabloya ait StatInfo sınıfındaki istatistiksel bilgileri kullanır.
- SELECTPLAN, PROJECTPLAN ve PRODUCTPLAN Sunu12, Şekil 17.13'deki maliyet hesaplama tablosundaki formülleri kullanır.
- SELECTPLAN için maliyet yükleme (*predicate*) bağlı olduğu için; yüklem kısmı aşağıdaki fonksiyonları kullanır:
  - reductionFactor(): recordsOutput() tarafından kullanılır...
  - equatesWithConstant(): distinctValues() tarafından kullanılır..

TABLEPLAN gerçeklemesi

```
public class TablePlan implements Plan {
   private Transaction tx;
   private TableInfo ti;
   private StatInfo si;

   public TablePlan(String tblname, Transaction tx) {
      this.tx = tx;
      ti = SimpleDB.mdMgr().getTableInfo(tblname, tx);
      si = SimpleDB.mdMgr().getStatInfo(tblname, ti, tx);
   }

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

   public int blocksAccessed() {
      return si.blocksAccessed();
   }

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

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

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

Figure 17-17
The code for the SimpleDB class *TablePlan*

SELECTPLAN gerceklemesi

```
public class SelectPlan implements Plan {
   private Plan p;
   private Predicate pred;

   public SelectPlan(Plan p, Predicate pred) {
      this.p = p;
      this.pred = pred;
   }

   public Scan open() {
      Scan s = p.open();
      return new SelectScan(s, pred);
   }

   public int blocksAccessed() {
      return p.blocksAccessed();
   }

   public int recordsOutput() {
      return p.recordsOutput() / pred.reductionFactor(p);
   }
```

```
   public int distinctValues(String fldname) {
      if (pred.equatesWithConstant(fldname) != null)
        return 1;
      else {
        String fldname2 = pred.equatesWithField(fldname);
        if (fldname2 != null)
          return Math.min(p.distinctValues(fldname), p.distinctValues(fldname2));
        else
          return Math.min(p.distinctValues(fldname), recordsOutput());
      }
   }
```

- reductionFactor(): yüklemin temel plandaki (*underlying plan, p*) kayıt sayısını ne oranda azalttığını bulur.
- equatesWithConstant(): ise yüklem koşulunun şeklini belirler. Yani;
  - A=*c ?*
  - A=B ?
  - B= *c*

  A *fieldname* niteliğinin sabit bir değere eşitlik mi yoksa başka bir niteliğe eşitlik mi olduğunu belirler.

PROJECTPLAN gerçeklemesi

```
public class ProjectPlan implements Plan {
   private Plan p;
   private Schema schema = new Schema();

   public ProjectPlan(Plan p,
                Collection <String> fieldlist) {
      this.p = p;
      for (String fldname : fieldlist)
            schema.add(fldname, p.schema());
   }

   public Scan open() {
      Scan s = p.open();
      return new ProjectScan(s, schema.fields());
   }

   public int blocksAccessed() {
      return p.blocksAccessed();
   }

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

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

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

Figure 17-19
The code for the SimpleDB class *ProjectPlan*

- Yeni bir schema oluşturuluyor...

PRODUCTPLAN gerçeklemesi

```
public class ProductPlan implements Plan {
   private Plan p1, p2;
   private Schema schema = new Schema();

   public ProductPlan(Plan p1, Plan p2) {
      this.p1 = p1;
      this.p2 = p2;
      schema.addAll(p1.schema());
      schema.addAll(p2.schema());
   }

   public Scan open() {
      Scan s1 = p1.open();
      Scan s2 = p2.open();
      return new ProductScan(s1, s2);
   }

   public int blocksAccessed() {
      return p1.blocksAccessed() +
            (p1.recordsOutput() * p2.blocksAccessed());
   }

   public int recordsOutput() {
      return p1.recordsOutput() * p2.recordsOutput();
   }

   public int distinctValues(String fldname) {
      if (p1.schema().hasField(fldname))
         return p1.distinctValues(fldname);
      else
         return p2.distinctValues(fldname);
   }

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

Figure 17-20
The code for the SimpleDB class *ProductPlan*

- Yeni bir schema oluşturuluyor...

Örnek

```
Plan p = new TablePlan ("dept",tx)
Scan s= p.open();
boolean ok = s.next();
```

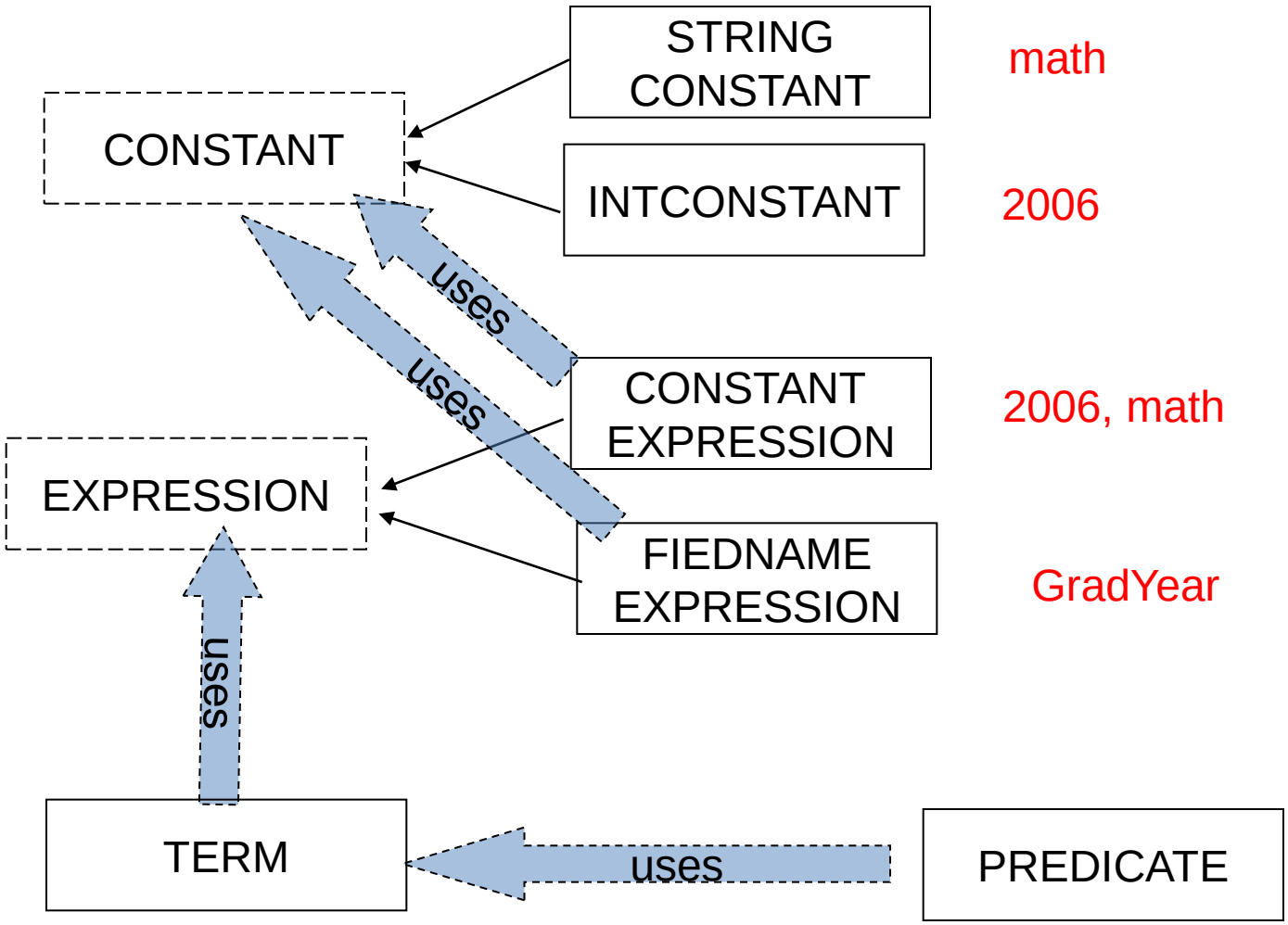
- Yukarıdaki işlemlerde sırayla tampon "pin" ve "kilit" durumları nasıldır?
- 1. adımda: tblcat ve fldcat tabloları için pin ve slock
- 2.adımda: dept tablosunda "RecordFile-->RecordPage--> tx.pin(..)" için sadece pin
- 3.adımda: dept tablsunun ilk bloğu için slock

Yüklemler (*Predicates*)

- (GradYear=2006 or GradYear=2006) and MajorId=DId

- Expression : sabit sayı (*constant*) veya nitelik veya bunların üzerindeki fonksiyonlar..
- Term: 2 expression'ın karşılaştırılması (=,≠,≤,≥,<,>)
- **Yüklem**: Birden çok Term'ün boolean kombinasyonu
- SimpleDB'de mevcut gerçeklenenler:
  - Expression: sadece sabit sayı ve nitelik
  - Term: sadece eşitlik
  - Yüklem: Term'lerin sadece AND ile birleşimi

yüklem gerçekleme sınıf hiyerarşisi



math

2006

2006, math

GradYear

GradYear> 2006

GradYear> 2006 and MajorId=math

Yüklem API

```
// used by the parser:
public void       conjoinWith(Predicate pred);

// used by a scan:
public boolean    isSatisfied(Scan s);

// used by a plan:
public int        reductionFactor();

// used by the query planner:
public Predicate  selectPred(Schema sch);
public Predicate  joinPred(Schema sch1, Schema sch2);
public Constant   equatesWithConstant(String fldname);
public String     equatesWithField(String fldname);
```

Figure 17-21
The API for the SimpleDB class *Predicate*

A=B ?

A=c ?

SimpleDB'de Sabit sayı (*constant*)

- Tamsayı (*integer*) sabitleri
- String sabitleri

- Yüklem içerisindeki termlerin doğru/yanlış olmaları için, sadece karşılaştırılabilir olmaları yeterlidir..bunun dışında herhangi bir tip kontrolü yok. *(➔Basitleştirilmiş sorgu işleme)*

```
public interface Constant extends Comparable<Constant> {
   public Object asJavaVal();
}
```

- Aşağıdaki dönüşümler ambalaj (*wrapper)* sınıflar ile gerçeklenir:
  - SimpleDB Tamsayı (*integer*) ⍰⍰Java Integer tipi : **IntConstant** sınıfı
  - SimpleDB String sabitleri ⍰⍰Java String tipi : **StringConstant** sınıfı

```
public class StringConstant implements Constant {
   private String val;

   public StringConstant(String s) {
      val = s;
   }

   public Object asJavaVal() {
      return val;
   }

   public boolean equals(Object obj) {
      StringConstant sc = (StringConstant) obj;
      return sc != null && val.equals(sc.val);
   }

   public int compareTo(Constant c) {
      StringConstant sc = (StringConstant) c;
      return val.compareTo(sc.val);
   }

   public int hashCode() {
      return val.hashCode();
   }

   public String toString() {
      return val;
   }
}
```

Expression arayüzü ve gerçekleme

```
public interface Expression {
   public boolean  isConstant();
   public boolean  isFieldName();
   public Constant asConstant();
   public String   asFieldName();
   public Constant evaluate(Scan s);
   public boolean  appliesTo(Schema sch);
}
```

```
public class ConstantExpression implements Expression {
   private Constant val;

   public ConstantExpression(Constant c) {
      val = c;
   }

   public boolean isConstant() {
      return true;
   }

   public boolean isFieldName() {
      return false;
   }

   public Constant asConstant() {
      return val;
   }

   public String asFieldName() {
      throw new ClassCastException();
   }

   public Constant evaluate(Scan s) {
      return val;
   }

   public boolean appliesTo(Schema sch) {
      return true;
   }

   public String toString() {
      return val.toString();
   }
}
```

Figure 17-25
The code for the SimpleDB class *ConstantExpression*

```
public class FieldNameExpression implements Expression {
   private String fldname;

   public FieldNameExpression(String fldname) {
      this.fldname = fldname;
   }

   public boolean isConstant() {
      return false;
   }

   public boolean isFieldName() {
      return true;
   }

   public Constant asConstant() {
      throw new ClassCastException();
   }

   public String asFieldName() {
      return fldname;
   }

   public Constant evaluate(Scan s) {
      return s.getVal(fldname);
   }

   public boolean appliesTo(Schema sch) {
      return sch.hasField(fldname);
   }

   public String toString() {
      return fldname;
   }
}
```

Figure 17-26
The code for the SimpleDB class *FieldNameExpression*

Term sınıfı gerçekleme

```
public class Term {
   private Expression lhs, rhs;

   public Term(Expression lhs, Expression rhs) {
      this.lhs = lhs;
      this.rhs = rhs;
   }

   public int reductionFactor(Plan p) {
      String lhsName, rhsName;
      if (lhs.isFieldName() && rhs.isFieldName()) {
         lhsName = lhs.asFieldName();
         rhsName = rhs.asFieldName();
         return Math.max(p.distinctValues(lhsName),
                         p.distinctValues(rhsName));
      }
      if (lhs.isFieldName()) {
         lhsName = lhs.asFieldName();
         return p.distinctValues(lhsName);
      }
      if (rhs.isFieldName()) {
         rhsName = rhs.asFieldName();
         return p.distinctValues(rhsName);
      }
      // otherwise, the term equates constants
      if (lhs.asConstant().equals(rhs.asConstant()))
         return 1;
      else
         return Integer.MAX_VALUE;
   }

   public Constant equatesWithConstant(String fldname) {
      if (lhs.isFieldName() && rhs.isConstant()
       && lhs.asFieldName().equals(fldname))
         return rhs.asConstant();
      else if (rhs.isFieldName() && lhs.isConstant()
            && rhs.asFieldName().equals(fldname))
         return lhs.asConstant();
      else
         return null;
   }

   public String equatesWithField(String fldname) {
      if (lhs.isFieldName() && rhs.isFieldName()
       && lhs.asFieldName().equals(fldname))
         return rhs.asFieldName();
      else if (rhs.isFieldName() && lhs.isFieldName()
            && rhs.asFieldName().equals(fldname))
         return lhs.asFieldName();
      else
         return null;
   }

   public boolean appliesTo(Schema sch) {
      return lhs.appliesTo(sch) && rhs.appliesTo(sch);
   }

   public boolean isSatisfied(Scan s) {
      Constant lhsval = lhs.evaluate(s);
      Constant rhsval = rhs.evaluate(s);
      return rhsval.equals(lhsval);
   }

   public String toString() {
      return lhs.toString() + "=" + rhs.toString();
   }
}
```

Figure 17-27
The code for the SimpleDB class *Term*

Figure 17-27 *(Continued)*

Predicate sınıfı gerçeklemesi

```
public class Predicate {
   private List<Term> terms = new ArrayList<Term>();

   public Predicate() {}

   public Predicate(Term t) {
      terms.add(t);
   }

   public void conjoinWith(Predicate pred) {
      terms.addAll(pred.terms);
   }

   public boolean isSatisfied(Scan s) {
      for (Term t : terms)
         if (!t.isSatisfied(s))
            return false;
      return true;
   }

   public int reductionFactor(Plan p) {
      int factor = 1;
      for (Term t : terms)
         factor *= t.reductionFactor(p);
      return factor;
   }

   public Predicate selectPred(Schema sch) {
      Predicate result = new Predicate();
      for (Term t : terms)
         if (t.appliesTo(sch))
            result.terms.add(t);
      if (result.terms.size() == 0)
         return null;
      else
         return result;
   }

   public Predicate joinPred(Schema sch1, Schema sch2)
      Predicate result = new Predicate();
      Schema newsch = new Schema();
```

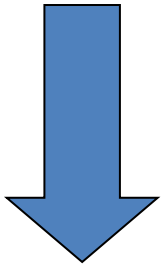
Figure 17-28
The code for the SimpleDB class *Predicate*

```
      newsch.addAll(sch1);
      newsch.addAll(sch2);
      for (Term t : terms)
         if (!t.appliesTo(sch1) && !t.appliesTo(sch2)
            && t.appliesTo(newsch))
               result.terms.add(t);
      if (result.terms.size() == 0)
         return null;
      else
         return result;
   }

   public Constant equatesWithConstant(String fldname) {
      for (Term t : terms) {
         Constant c = t.equatesWithConstant(fldname);
         if (c != null)
            return c;
      }
      return null;
   }

   public String equatesWithField(String fldname) {
      for (Term t : terms) {
         String s = t.equatesWithField(fldname);
         if (s != null)
            return s;
      }
      return null;
   }

   public String toString() {
      Iterator<Term> iter = terms.iterator();
      if (!iter.hasNext())
         return "";
      String result = iter.next().toString();
      while (iter.hasNext())
         result += " and " + iter.next().toString();
      return result;
   }
}
```

Figure 17-28 *(Continued)*

Örnek

SName='joe' and MajorId=DId



Çözümleyici
(*Parser*)

```
Expression lhs1 = new FieldNameExpression("SName");
Constant c = new StringConstant("joe");
Expression rhs1 = new ConstantExpression(c);
Term t1 = new Term(lhs1, rhs1);

Expression lhs2 = new FieldNameExpression("MajorId");
Expression rhs2 = new FieldNameExpression("DId");
Term t2 = new Term(lhs2, rhs2);

Predicate pred1 = new Predicate(t1);
pred1.conjoinWith(new Predicate(t2));
```

Figure 17-29
SimpleDB code to create a predicate