VT Gerçeklenmesi Ders Notları- #6

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.

Katalog, Üstveri (*Metadata*) yönetimi

- Tablo (*Table*)
- Görüntü (*View*)
- İndeks (*Index*)
- Statistiksel (*Statistical*)

Üstveri (Metadata)

- Bundan önce; veri tabanı tablosu (*relation*) yoktu. Kayıtların oluşturduğu yeni yapı: TABLO (relation)
- Üstveri tipleri:
  - **Tablo üstverisi:** Şema ve fiziksel özellikleri. Kayıt yöneticisi tarafından tutulan bilgi.
  - **Görüntü üstverisi**: Görüntü tanımı ve kimin oluşturduğu. Planlamada kullanılır.
  - **İndeks üstverisi:** Tablo üzerinde oluşturulan indeksler hakkında bilgi. Planlamada kullanılır.
  - **İstatistik üstverisi:** Tablo büyüklüğü, niteliklerdeki değer dağılımı. Sorgu maliyeti tahmininde kullanılır.
- Üstveri bilgileri KATALOG adı verilen tablolarda saklanır.
  - Tablo üstverisi:
    - **tblcat** (tblname, reclength)
    - **fldcat** (tblname, fldname, type, length, offset)
  - Görüntü üstverisi:
    - **viewcat** (viewname, viewdef)
  - İstatistik üstverisi:
    - **tblstats**(tblname, numblocks, numrecords)
    - **fldstats** (tblname, fldname, numvalues)
  - İndeks üstverisi:
    - **idxcat** (indexname, tablename, fieldname)

SimpleDB'de disk yerine, ana hafızada gerçekleniyor.

MetadataMgr sinifi için API

```
public void createTable(String tblname, Schema sch,
                        Transaction tx);
public TableInfo getTableInfo(String tblname,
                              Transaction tx);
public void createView(String viewname, String viewdef,
                       Transaction tx);
public String getViewDef(String viewname,
                         Transaction tx);
public void createIndex(String idxname, String tblname,
                        String fldname, Transaction tx);
public Map<String,IndexInfo> getIndexinfo(String tblname,
          Transaction tx);
public StatInfo getStatInfo(String tblname, TableInfo ti,
                            Transaction tx);
```

Figure 16-1
The API for the SimpleDB class *MetadataMgr*

- SimpleDB sistemi içinde 1 adet «mdMgr» sınıfı var.

TableMgr

- **tblcat** (tblname, reclength)
- **fldcat** (tblname, fldname, type, length, offset)

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

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

// Part 1: Create the DEPT table
Transaction tx1 = new Transaction();
Schema sch = new Schema();
sch.addIntField("did");
sch.addStringField("dname", 8);
mdMgr.createTable("dept", sch, tx1);
tx1.commit();

// Part 2: Print the name of each department
Transaction tx2 = new Transaction();
TableInfo ti = mdMgr.getTableInfo("dept", tx2);
RecordFile rf = new RecordFile(ti, tx2);
while (rf.next())
      System.out.println(rf.getString("dname"));
rf.close();
tx2.commit();
```

Figure 16-2
Using the table metadata methods

tblcat	TblName	RecLength
	student	26
	dept	16
	course	32
	section	24
	enroll	18

in char

fldcat	TblName	FldName	Type	Length	Offset
	student	sid	4	0	0
	student	sname	12	10	12
	student	majorid	4	0	8
	student	gradyear	4	0	4
	dept	did	4	0	12
	dept	dname	12	8	0
	course	cid	4	0	28
	course	title	12	20	0
	course	deptid	4	0	24
	section	sectid	4	0	12
	section	courseid	4	0	20
	section	prof	12	8	0
	section	yearoffered	4	0	16
	enroll	eid	4	0	10
	enroll	studentid	4	0	14
	enroll	sectionid	4	0	0
	enroll	grade	12	2	4

Figure 16-3
Catalog tables for the university database

JDBC types. 4:int, 12: string

TableMgr gerçeklenmesi

```
public class TableMgr {
   // table and field names are varchar(16)
   public static final int MAX_NAME = 16;

   private TableInfo tcatInfo, fcatInfo;

   public TableMgr(boolean isNew, Transaction tx) {
      Schema tcatSchema = new Schema();
      tcatSchema.addStringField("tblname", MAX_NAME);
      tcatSchema.addIntField("reclength");
      tcatInfo = new TableInfo("tblcat", tcatSchema);

      Schema fcatSchema = new Schema();
      fcatSchema.addStringField("tblname", MAX_NAME);
      fcatSchema.addStringField("fldname", MAX_NAME);
      fcatSchema.addIntField("type");
      fcatSchema.addIntField("length");
      fcatSchema.addIntField("offset");
      fcatInfo = new TableInfo("fldcat", fcatSchema);

      if (isNew) {
         createTable("tblcat", tcatSchema, tx);
         createTable("fldcat", fcatSchema, tx);
      }
   }

   public void createTable(String tblname, Schema sch,
                           Transaction tx){
      TableInfo ti = new TableInfo(tblname, sch);

      // insert one record into tblcat
      RecordFile tcatfile = new RecordFile(tcatInfo, tx);
      tcatfile.insert();
      tcatfile.setString("tblname", tblname);
      tcatfile.setInt("reclength", ti.recordLength());
      tcatfile.close();

      // insert a record into fldcat for each field
      RecordFile fcatfile = new RecordFile(fcatInfo, tx);
      for (String fldname : sch.fields()) {
         fcatfile.insert();
         fcatfile.setString("tblname", tblname);
         fcatfile.setString("fldname", fldname);
         fcatfile.setInt("type", sch.type(fldname));
         fcatfile.setInt("length", sch.length(fldname));
         fcatfile.setInt("offset", ti.offset(fldname));
      }
      fcatfile.close();
   }
```

```
public TableInfo getTableInfo(String tblname,
                              Transaction tx) {
   RecordFile tcatfile = new RecordFile(tcatInfo, tx);
   int reclen = -1;
   while (tcatfile.next())
      if(tcatfile.getString("tblname").equals(tblname)){
         reclen = tcatfile.getInt("reclength");
         break;
      }
   tcatfile.close();
   RecordFile fcatfile = new RecordFile(fcatInfo, tx);
   Schema sch = new Schema();
   Map<String,Integer> offsets =
                        new HashMap<String,Integer>();
   while (fcatfile.next())
      if (fcatfile.getString("tblname")
                  .equals(tblname)) {
         String fldname = fcatfile.getString("fldname");
         int fldtype    = fcatfile.getInt("type");
         int fldlen     = fcatfile.getInt("length");
         int offset     = fcatfile.getInt("offset");
         offsets.put(fldname, offset);
         sch.addField(fldname, fldtype, fldlen);
      }
   fcatfile.close();
   return new TableInfo(tblname, sch, offsets, reclen);
  }
}
```

ViewMgr gerçeklenmesi

```
class ViewMgr {
   private static final int MAX_VIEWDEF = 100;
   TableMgr tblMgr;

   public ViewMgr(boolean isNew, TableMgr tblMgr,
                  Transaction tx) {
      this.tblMgr = tblMgr;
      if (isNew) {
         Schema sch = new Schema();
         sch.addStringField("viewname", TableMgr.MAX_NAME);
         sch.addStringField("viewdef", MAX_VIEWDEF);
         tblMgr.createTable("viewcat", sch, tx);
      }
   }

   public void createView(String vname, String vdef,
                          Transaction tx) {
      TableInfo ti = tblMgr.getTableInfo("viewcat", tx);
      RecordFile rf = new RecordFile(ti, tx);
      rf.insert();
      rf.setString("viewname", vname);
      rf.setString("viewdef", vdef);
      rf.close();
   }

   public String getViewDef(String vname, Transaction tx) {
      String result = null;
      TableInfo ti = tblMgr.getTableInfo("viewcat", tx);
      RecordFile rf = new RecordFile(ti, tx);
      while (rf.next())
         if (rf.getString("viewname").equals(vname)) {
            result = rf.getString("viewdef");
            break;
         }
      rf.close();
      return result;
   }
}
```

Figure 16-6
The code for the SimpleDB class *ViewMgr*

- **viewcat** (viewname, viewdef)

StatMgr

- Sorgu işleyicinin ihtiyacı olan bilgiler:
  - Tablodaki kayıt sayısı
  - Nitelik değer dağılımı
  - Nitelik/değer histogramları
  - Nitelikler arası korelasyon değerleri
- SimpleDB'de:
  - B(T) : T, Tablosundaki blok sayısı
  - R(T) : T, Tablosundaki kayıt sayısı
  - V(T,F): T, Tablosunun, F niteliğindeki farklı değer sayısı

University veri tabanına ait bazı istatistikler

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

- Elimizdeki bilgiler:
  - 40 depts
  - 500 courses
  - 900 students/year
  - 500 sections/year
  - 50 years
  - 10 studentrecord/block
  - 20 dept_record/block

T	B(T)	R(T)	V(T, F)	
STUDENT	4,500	45,000	45,000 44,960 50 40	for F=SId for F=SName for F=GradYear for F=MajorId
DEPT	2	40	40	for F=DId, DName
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

Figure 16-7
Example statistics about the university database

StatMgr gerçeklenmesi

```
public int blocksAccessed();
public int recordsOutput();
public int distinctValues(String fldname);
```

Figure 16-8
The API for the SimpleDB class *StatInfo*

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

Transaction tx = new Transaction();
TableInfo ti   = mdMgr.getTableInfo("student", tx);
StatInfo si    = mdMgr.getStatInfo(ti, tx);
System.out.println(si.blocksAccessed() + " " +
                   si.recordOutput()  + " " +
                   si.distinctValues("majorid"));
tx.commit();
```

Figure 16-9
Obtaining and printing statistics about a table

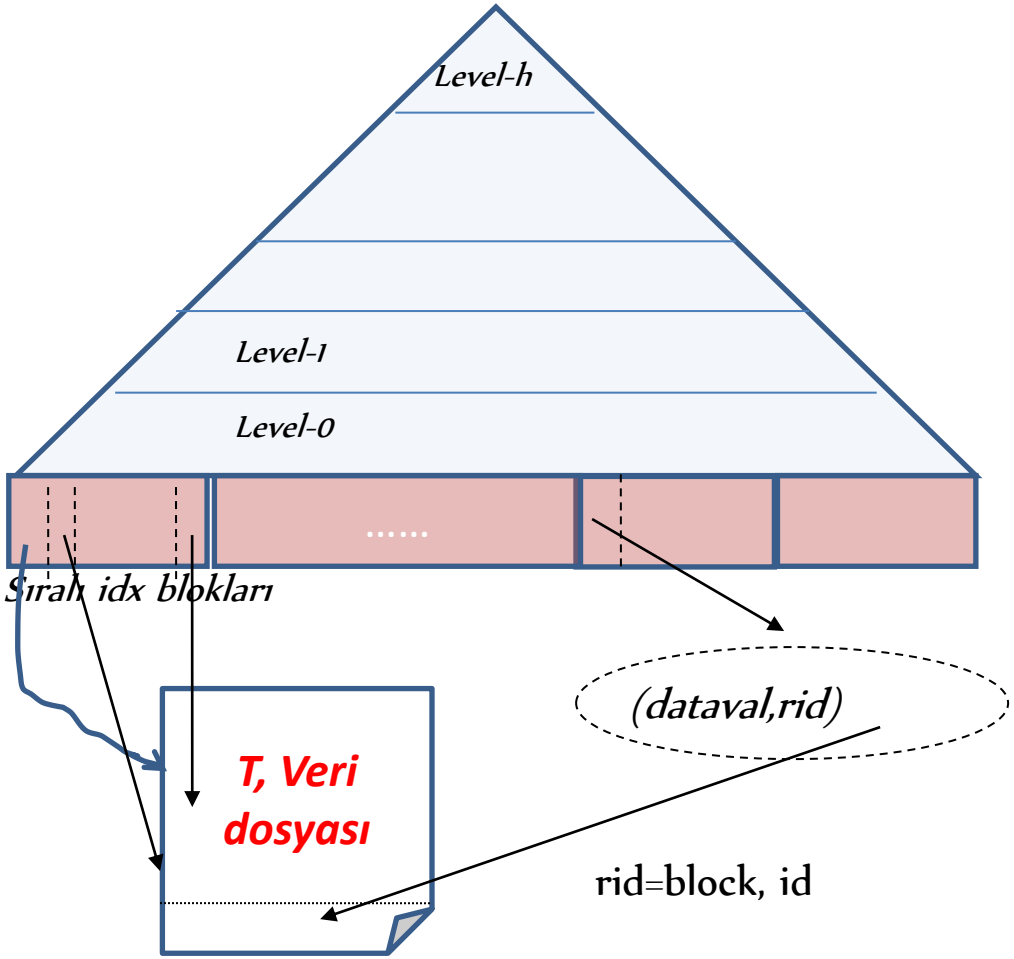
- **İstatistik katalogları:**
  - **tblstats**(tblname, numblocks, numrecords)
  - **fldstats** (tblname, fldname, numvalues)
- SimpleDB'de sistem ilk çalıştığında istatistikler hesaplanıp ana hafızada tutulur. Bu *StatMgr* sınıfı ile gerçeklenir..
- Bilginin yenilenmesi
  - Her eylem sonrası
  - Biriktirip tazeleme

StatMgr gerçeklenmesi

```
class StatMgr {
   private TableMgr tblMgr;
   private Map<String,StatInfo> tablestats;
   private int numcalls;

   public StatMgr(TableMgr tblMgr, Transaction tx) {
      this.tblMgr = tblMgr;
      refreshStatistics(tx);
   }

   public synchronized StatInfo getStatInfo (String tblname,
                           TableInfo ti, Transaction tx) {
      numcalls++;
      if (numcalls > 100)
         refreshStatistics(tx);
      StatInfo si = tablestats.get(tblname);
      if (si == null) {
         si = calcTableStats(ti, tx);
         tablestats.put(tblname, si);
      }
      return si;
   }

   private synchronized void refreshStatistics(
                                          Transaction tx) {
      tablestats = new HashMap<String,StatInfo>();
      numcalls = 0;
      TableInfo tcatinfo =
                        tblMgr.getTableInfo("tblcat", tx);
      RecordFile tcatfile = new RecordFile(tcatinfo, tx);
      while(tcatfile.next()) {
         String tablname = tcatfile.getString("tblname");
         TableInfo ti = tblMgr.getTableInfo(tblname, tx);
         StatInfo si = calcTableStats(ti, tx);
         tablestats.put(tblname, si);
      }
      tcatfile.close();
   }

   private synchronized StatInfo
         calcTableStats(TableInfo ti, Transaction tx) {
      int numRecs = 0;
      RecordFile rf = new RecordFile(ti, tx);
      int numblocks = 0;
      while (rf.next()) {
         numRecs++;
         numblocks = rf.currentRid().blockNumber() + 1;
      }
      rf.close();
      return new StatInfo(numblocks, numRecs);
   }
}
```

```
public class StatInfo {
   private int numBlocks;
   private int numRecs;

   public StatInfo(int numblocks, int numrecs) {
      this.numBlocks = numblocks;
      this.numRecs   = numrecs;
   }

   public int blocksAccessed() {
      return numBlocks;
   }

   public int recordsOutput() {
      return numRecs;
   }

   public int distinctValues(String fldname) {
      return 1 + (numRecs / 3);
   }
}
```

Indeksleme



- "T" tablosunda, "n" niteliği üzerinde bir indeks görüntüsü. Indeks dosyasının şeması:
- (dataval, block,id)
  - dataval: "n" niteliğine ait dosyadaki değerler
  - block: "T" dosyasındaki blok numarası
  - id: blok içerisindeki slot id.

IndexMgr

- **idxcat** (indexname, tablename, fieldname)
- Her bir index'e ait üstveri, *IndexInfo* sınıfı ile temsil edilir..

```
public IndexInfo(String idxname, String tblname,
                 String fldname, Transaction tx);
public int blocksAccessed();
public int recordsOutput();
public int distinctValues(String fldname);
public Index open();
```

Figure 16-12
The API for the SimpleDB class *IndexInfo*

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

MetadataMgr mdMgr = SimpleDB.mdMgr();
Map<String,IndexInfo> indexes =
                      mdMgr.getIndexInfo("student", tx);

// Part 1: Print the name and cost of each index on STUDENT
for (String fldname : indexes.keySet()) {
   IndexInfo ii = indexes.get(fldname);
   System.out.println(fldname + "\t" +
                      ii.blocksAccessed());
}

// Part 2: Open the index on MajorId
IndexInfo ii = indexes.get("majorid");
Index idx = ii.open();
```

Figure 16-13
Using the SimpleDB index manager

IndexMgr gerçeklenmesi

```
public class IndexMgr {
   private TableInfo ti;

   public IndexMgr(boolean isnew, TableMgr tblmgr,
                   Transaction tx) {
      if (isnew) {
         Schema sch = new Schema();
         sch.addStringField("indexname", MAX_NAME);
         sch.addStringField("tablename", MAX_NAME);
         sch.addStringField("fieldname", MAX_NAME);
         tblmgr.createTable("idxcat", sch, tx);
      }
      ti = tblmgr.getTableInfo("idxcat", tx);
   }

   public void createIndex(String idxname, String tblname,
                           String fldname, Transaction tx) {
      RecordFile rf = new RecordFile(ti, tx);
      rf.insert();
      rf.setString("indexname", idxname);
      rf.setString("tablename", tblname);
      rf.setString("fieldname", fldname);
      rf.close();
   }
```

```
   public Map<String,IndexInfo>
            getIndexInfo(String tblname, Transaction tx) {
      Map<String,IndexInfo> result =
                        new HashMap<String, IndexInfo>();
      RecordFile rf = new RecordFile(ti, tx);
      while (rf.next())
         if (rf.getString("tablename").equals(tblname)) {
            String idxname = rf.getString("indexname");
            String fldname = rf.getString("fieldname");
            IndexInfo ii =
              new IndexInfo(idxname, tblname, fldname, tx);
            result.put(fldname, ii);
         }
      rf.close();
      return result;
   }
}
```

IndexInfo

- "I": "T" veri tablosunun "n1" niteliği üzerindeki indeks tablosu olsun.
- "I" için IndexInfo: "I" hakkındaki istatistiksel bilgi tutan nesne. Ihtiyaç duyulduğu zaman run-time'da oluşturuluyor.
- blocksAccessed(): Bu Indeks üzerinden veri dosyasındaki bir konuma ulaşmak için indekste kaç disk blok erişimi oldu?
- recordsOutput(): Bu indeks üzerinden bir "selection" (n=*constant*) ile kaç *rid* ortaya çıkar (kaç kayda erişilebilir)?
- distinctValues(n2): Bu indeks üzerinden bir "selection" (n1=*constant*) ile kaç farklı "n2" nitelik değeri ortaya çıkar?

IndexMgr-IndexInfo gerçeklenmesi

```
public class IndexInfo {
   private String idxname, fldname;
   private int idxtype;
   private Transaction tx;
   private TableInfo ti;
   private StatInfo si;

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

   public Index open() {
      Schema sch = schema();
      return new BTreeIndex(idxname, sch, tx);
   }
```

```
   public int blocksAccessed() {
      TableInfo idxti = new TableInfo("", schema());
      int rpb = BLOCK_SIZE / idxti.recordLength();
      int numblocks = si.recordsOutput() / rpb;
      // Call HashIndex.searchCost for hash indexing
      return HashIndex.searchCost(numblocks, rpb);
   }
```

```
   public int recordsOutput() {
      return si.recordsOutput() /
             si.distinctValues (fldname);
   }

   public int distinctValues(String fname) {
      if (fldname.equals(fname))
         return 1;
      else
         return Math.min(si.distinctValues(fldname),
                         recordsOutput());
   }

   private Schema schema() {
      Schema sch = new Schema();
      sch.addIntField("block");
      sch.addIntField("id");
      if (ti.schema().type(fldname) == INTEGER)
         sch.addIntField("dataval");
      else {
         int fldlen = ti.schema().length(fldname);
         sch.addStringField("dataval", fldlen);
      }
      return sch;
   }
}
```

recordsOutput(), distinctValues(*fieldname*)

x	. . .	y
2		a
2		b
2		a
2		b

- Tabloya ait StatInfo = si
  - si.recordsOutput() = 100
  - si.distinctValues('x') = 25
  - si.distinctValues('y') =2 olsun...
- Fieldname='x' üzerindeki index'e ait:
  - recordsOutput()= 100/25 = 4
  - distinctValues('x') = 1
  - distinctValues('y') = 2
- Fieldname='y' üzerindeki index'e ait:
  - recordsOutput()= 100/2 = 50
  - distinctValues('x') = 25
  - distinctValues('y') = 1

MetadataMgr gerçeklenmesi

```
public class MetadataMgr {
   private static TableMgr tblmgr;
   private static ViewMgr viewmgr;
   private static IndexMgr idxmgr;
   private static StatMgr statmgr;

   public MetadataMgr(boolean isnew, Transaction tx) {
      tblmgr  = new TableMgr(isnew, tx);
      viewmgr = new ViewMgr(isnew, tblmgr, tx);
      idxmgr  = new IndexMgr(isnew, tblmgr, tx);
      statmgr = new StatMgr(tblmgr, tx);
   }

   public void createTable(String tblname, Schema sch,
                           Transaction tx){
      tblmgr.createTable(tblname, sch, tx);
   }

   public TableInfo getTableInfo(String tblname,
                                 Transaction tx) {
      return tblmgr.getTableInfo(tblname, tx);
   }

   public void createView(String viewname, String viewdef,
                          Transaction tx) {
      viewmgr.createView(viewname, viewdef, tx);
   }
```

```
   public String getViewDef(String viewname,
                            Transaction tx) {
      return viewmgr.getViewDef(viewname, tx);
   }

   public void createIndex(String idxname, String tblname,
                           String fldname, Transaction tx) {
      idxmgr.createIndex(idxname, tblname, fldname, tx);
   }

   public Map<String,IndexInfo>
          getIndexInfo(String tblname, Transaction tx) {
      return idxmgr.getIndexInfo(tblname, tx);
   }

   public StatInfo getStatInfo(String tblname,
                               TableInfo ti, Transaction tx) {
      return statmgr.getStatInfo(tblname, ti, tx);
   }
}
```